ABSTRACT

Title of Document:	METHODS FOR HIGH-THROUGHPUT COMPARATIVE GENOMICS AND DISTRIBUTED SEQUENCE ANALYSIS
	Samuel Vincent Angiuoli, Ph.D., 2011
Directed By:	Professor S.L. Salzberg, Department of Computer Science

High-throughput sequencing has accelerated applications of genomics throughout the world. The increased production and decentralization of sequencing has also created bottlenecks in computational analysis. In this dissertation, I provide novel computational methods to improve analysis throughput in three areas: whole genome multiple alignment, pan-genome annotation, and bioinformatics workflows.

To aid in the study of populations, tools are needed that can quickly compare multiple genome sequences, millions of nucleotides in length. I present a new multiple alignment tool for whole genomes, named Mugsy, that implements a novel method for identifying syntenic regions. Mugsy is computationally efficient, does not require a reference genome, and is robust in identifying a rich complement of genetic variation including duplications, rearrangements, and large-scale gain and loss of sequence in mixtures of draft and completed genome data. Mugsy is evaluated on the alignment of several dozen bacterial chromosomes on a single computer and was the fastest program evaluated for the alignment of assembled human chromosome sequences from four individuals. A distributed version of the algorithm is also described and provides increased processing throughput using multiple CPUs.

Numerous individual genomes are sequenced to study diversity, evolution and classify pan-genomes. Pan-genome annotations contain inconsistencies and errors that hinder comparative analysis, even within a single species. I introduce a new tool, Mugsy-Annotator, that identifies orthologs and anomalous gene structure across a pan-genome using whole genome multiple alignments. Identified anomalies include inconsistently located translation initiation sites and disrupted genes due to draft genome sequencing or pseudogenes. An evaluation of pan-genomes indicates that such anomalies are common and alternative annotations suggested by the tool can improve annotation consistency and quality.

Finally, I describe the Cloud Virtual Resource, CloVR, a desktop application for automated sequence analysis that improves usability and accessibility of bioinformatics software and cloud computing resources. CloVR is installed on a personal computer as a virtual machine and requires minimal installation, addressing challenges in deploying bioinformatics workflows. CloVR also seamlessly accesses remote cloud computing resources for improved processing throughput. In a case study, I demonstrate the portability and scalability of CloVR and evaluate the costs and resources for microbial sequence analysis.

METHODS FOR HIGH-THROUGHPUT COMPARATIVE GENOMICS AND DISTRIBUTED SEQUENCE ANALYSIS

By

Samuel Vincent Angiuoli

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Foreword

A version of the content described in Chapter 4, Mugsy, was published as Angiuoli, SV and Salzberg, SL in *Bioinformatics* [1]. A version of the content described in Chapter 6, Mugsy-Annotator, was submitted for publication to *BMC Bioinformatics* as Angiuoli *et al*. and was pending review at the time of preparing this dissertation.

Dedication

To Gilda and Vincent James Frattarelli

Acknowledgements

I thank my family who has taught me the most important lessons of life and for which I owe the most gratitude. I am especially thankful for Emily and Samuel J. Angiuoli, who have inspired me through their lives, the family they raised, and all their strength and hard work. I thank my wonderful fiancé Soraia whose patience is unmatched; may we enjoy our life together now that school is out. To my parents, I am forever grateful and indebted to your endless and unwavering support. Dad, you've worked hard; it's a good time for you to retire, you really deserve it.

I thank my advisor, Steven Salzberg, who provided me a unique opportunity to pursue this research and has provided invaluable mentorship throughout my career. I thank my committee members for their support, encouragement, and guidance. I also thank countless colleagues at the Institute for Genome Sciences, JCVI, and TIGR for their support and feedback through the years. I'm especially thankful to Owen White for his mentorship and support for my work and study.

Table of Contents

List of Tables

List of Figures

Chapter 1 - Introduction

Advances in genome sequencing and a bioinformatics bottleneck

Genome sequencing has widespread applications, including basic science, biosafety and biomedical research, and is expected to become part of the service sector, e.g. in the form of personalized health care [2,3,4]. Numerous individual genomes are sequenced to study genetic diversity of populations in depth, as no single reference sequence can fully describe the biology of a species [5,6,7].

The interpretation of genome sequence is reliant on computation and comparison. As a consequence, high-throughput sequencing technologies [8,9,10], while aiding in sequence acquisition, have also given rise to a “bioinformatics bottleneck”. Contributing to this bottleneck, the rate of sequence acquisition is exceeding improvements in computer performance and predictions from Moore's law [11,12]. For particular computational methods, such as multiple sequence alignment, optimal solutions are infeasible for large data sets [13] requiring heuristic solutions that improve computational efficiency.

In conjunction with increased throughput, the introduction of "benchtop" sequencing that aims at integrating medium-scale, affordable sequence generation into the standard laboratory equipment [14] is following a decentralization trend where sequence generation is becoming available for any size laboratory all over the world [15]. This democratization of sequencing has resulted in a larger, more diverse set of users that require bioinformatics software for analysis [16,17]. This trend has also exposed practical and technical challenges in sequence analysis using bioinformatics software (software requirements) [17,18] and computational resources (hardware requirements)

[12,19], contributing to analysis bottlenecks. In combination, bottlenecks in analysis have made the feasibility and affordability of applications of genomics increasingly dependent on bioinformatics tools and methods rather than sequence generation itself.

Novel methods for efficient whole genome comparison and high-throughput sequence analysis

This dissertation describes novel computational methods and software that improve analysis throughput for whole genome alignment and microbial sequence analysis. The original contributions include:

Algorithms for efficient comparisons of whole genomes

In Chapters 3-5, a series of novel algorithms and new tools are presented for alignment and comparisons of multiple whole genomes. In Chapter 4, we describe a new tool, Mugsy, for the multiple alignment of whole genomes. Mugsy includes a novel algorithm for identifying collinear regions (synteny) across multiple genomes without the need for a reference genome [1]. Mugsy is computationally efficient for the multiple alignment of closely related genomes that share fractions of identical or nearly identical DNA. Mugsy is not biased towards a reference genome, is robust in handling draft genomes, and can identify a wide range of genome-scale diversity, including rearrangements and duplications.

In Chapter 5, we extend the work on whole genome alignment to provide a distributed version of the algorithm, enabling faster computation and calculation of larger alignments than can be generated on a single computer.

Algorithms for improving annotation of pan-genomes

In Chapter 6, we describe a novel method for identifying orthologs and annotation inconsistencies across a pan-genome using whole genome multiple alignment. The method, implemented in a tool Mugsy-Annotator, is computationally efficient. We use the tool to evaluate the quality of annotated gene structures in several species pan-genomes and improve annotated gene structures across the bacterium *Neisseria meningitidis*.

A novel software platform for portable and automated sequence analysis

In Chapter 7, we describe a novel software platform, Cloud Virtual Resource (CloVR), for portable and automated analysis of microbial genomes. CloVR provides a single software executable that runs on a personal computer and seamlessly accesses cloud computing resources over the Internet for increased processing throughput. In Chapter 8, we utilize the CloVR software to evaluate the required resources and costs for microbial sequence analysis.

Dissertation organization

The remainder of this document is organized as follows. Background information relevant to the dissertation is described in Chapter 2. A review of literature related to the contributions is described in Chapter 3- Related Works. The contributions are presented and evaluated in Chapters 4-8. Finally, discussion, conclusions drawn, and areas for future work are described in Chapter 9.

Chapter 2 - Background

Genome sequencing

A wide-range of genome sequencing projects across the tree of life have broadened understanding of life and evolution [20]. For microbial organisms, there are already several thousand bacterial genomes in public databases [21], and hundreds of individual genomes for some medically relevant species and model organisms [22,23]. The Cancer Genome Atlas [24], 1000 Genomes Project [25] and the Personal Genome Project [26] provide genome sequences from at least several thousand people.

The popularity of genomics applications has largely been driven by the introduction of second generation sequencing technologies that offer increasing sequencing throughput at a decreasing cost per nucleotide [8,9]. As third-generation sequencing platforms [10] are now available, the cost of sequence generation is likely to decrease even further.

Pan-genomics

The availability of sequence data propels work in comparative genomics to study genome populations and their evolution. In Chapter 5, we describe a method for comparison and annotation of populations of genomes, so called pan-genomes. For many species, tremendous intra-species diversity results in a pan-genome much larger than any individual genome [7,27,28]. The pan-genome describes the genetic complement that is accessible to an organism, comprised of a core genome that is largely conserved and an accessory genome that is more variable [5,6,29]. Characterization of a pan-genome sheds light on an organisms' biology, life style and has implications for the definition of the species itself [30,31]. As hundreds of genomes are now available for some species [21], high-throughput methods are needed for comparison and annotation of closely related genomes.

<u>*Sequence Alignment*</u>

The interpretation of this genetic information is reliant on computation for comparison. In Chapters 4-5, we describe a novel methodology for alignment of multiple whole genomes. Sequence alignment, particularly multiple sequence alignment, is one of the most basic and studied problems in computational biology [32,33]. The primary motivation for aligning biological sequences, such as DNA, is identifying characters of the sequence that have evolved from a common ancestor. For DNA, the characters of a sequence are nucleotides, also called base pairs, which have a standard alphabet of 4 characters {A,C,T,G}. The methods for sequence alignment are closely related to general sequence search algorithms, such as longest common subsequence [34], and have long since been adapted to biological sequences, propelled by the work of Needleman-Wunsch [35] and Smith-Waterman [36]. To improve runtimes on searches of large sequences, seed and extend heuristics are widely utilized for pairwise alignment, including BLAST [37], MEGABLAST [38], and BLAT [39].

Computation of an optimally scoring alignment for multiple sequences is NP-hard [40] using simple scoring schemes [13]. As a result, a variety of heuristic algorithms have been employed for multiple sequence alignment of collinear sequences, including those in [41,42,43]. Progressive alignment [44] is a popular heuristic for iteratively building a multiple alignment from the conjunction of pairwise alignments, although this method is greedy and subject to the propogation of errors during the progression resulting in suboptimally scoring alignments. To improve quality, iterative refinement methods that re-align low-scoring regions are often utilized [45].

Genome annotation

In Chapter 5, we describe a method for identifying annotation problem areas. To aid in the whole genome analysis, an annotation process is typically performed using computational methods that include prediction of genes and their functions [46]. Protein coding genes in prokaryotes are frequently comprised of a single open reading frame that begins with a translation initiation site and ends with a termination codon. Gene prediction algorithms for prokaryotes have been shown to perform well with relatively low error rates [47,48,49]. Limitations of gene prediction include accurate identification of the translation initiation start (TIS) [50] sites and pseudogenes, and over-annotation in GC-rich genomes [51]. Specialized tools have addressed these issues, such as for improved TIS prediction [50].

Some common errors in gene prediction, such as inaccurate identification of TIS, are problematic for experimentation work, making identification and correction important. Correctly annotated gene structures and translation initiation sites are critical for proteomics studies, including N-terminal protein sequencing [52], and construction of DNA and protein microarrays. Computational protein structure prediction also relies on accurate gene structures.

The problem of annotation errors is heightened in pan-genomic studies involving many individually annotated genomes [53]. Rare errors, including missed gene predictions, are compounded as more genomes are added obscuring identification of core genes that are critical for a species and present in every sequenced isolate [54]. Missing genes can lead to false biological inferences if such missed genes are critical components of metabolic pathways.

Virtual machines

In Chapter 7, we describe an architecture for distributed sequence analysis that relies on virtual machines. A Virtual Machine (VM) is a piece of software that emulates an entire operating system and can be bundled with pre-installed and pre-configured software. Upon execution, the VM has the appearance of booting a new computer through a process called virtualization. On a host computer, the VM runs inside a software application called a hypervisor (also called VM player) that supports virtualization. There are VM players [55,56] available for all major operating systems, including Microsoft Windows, Apple Mac OS X, and Linux.

A VM is portable and can be distributed over the Internet and executed anywhere in the world, without further need for complex installations and adaptations. As a result, the VM provides a means to eliminate complex software installations and adaptations for portable execution, directly addressing one of the challenges involved with using bioinformatics tools and pipelines. Most importantly, the developer of a VM has super-user access and complete control over the operating system configuration, so there are few limitations to installing and configuring additional software on the VM.

Cloud computing

In Chapters 7-8, we utilize cloud computing platforms to improve processing throughput of pre-packaged analysis pipelines. Despite performance increases of modern CPUs, single desktops or even small computer clusters are limited in the volume of sequencing data that they can analyze. Distributed computing platforms that provide many computers for processing data in parallel are commonly used to improve analysis throughput. Such systems, include clusters of machines on a local network [57] and grids that connect machines over wide area networks [58].

Cloud computing is a distributed computing platform which offers on-demand leases to computational resources over a network [59]. Cloud computing can provide access to a variety of computing architectures, including large memory machines, while eliminating the need to build or administer a local computer network, addressing challenges in access and deployment of infrastructure for bioinformatics [12,60]. Cloud computing platforms have been emerging in the commercial sector, including the Amazon Elastic Compute Cloud (EC2) [61], and in the public sector to support research [62,63]. Amazon EC2 provides on-demand compute (priced per CPU hour) and charges additionally for network transfers to and from the cloud (bandwidth priced per GB) and persistent data storage (priced per GB and per month). Importantly these cloud platforms support user-provided virtual machines, allowing for extensive customization of operation system and software that is executed on the servers.

Chapter 3 - Related Works

Whole genome multiple alignment

In Chapter 4-5, we describe a novel methodology for whole genome multiple alignment. Whole genome alignment has become instrumental for studying genome evolution and genetic diversity [33,64], with applications in construction of phylogenies, study of gene families, and characterization of a pan-genomic and species specific DNA. Whole genome alignment tools are distinguished from collinear sequence alignment tools, such as tools of [41,42,43], in that they can align very long sequences, millions of base pairs in length, and identify both large-scale and smaller scale variation. Large-scale mutations include rearrangement, duplication, gain, and loss of genetic segments. Small-scale variation includes in local substitution, insertion, and deletion of individual nucleotides. Biological processes, such as homologous recombination [65], produce these mutations during species evolution resulting in extensive flux of genetic elements within and between chromosomes, even for very closely related species. Whole genome alignment describes these variations by identifying matching nucleotides in two or more organisms that are derived from a common ancestral sequence.

There are numerous methods to compare a single pair of whole genome sequences [66,67]. The Nucmer and MUMmer package are fast whole genome alignment method that utilizes a suffix tree to seed an alignment with maximal unique matches (MUMs) [68]. The suffix tree implementation of MUMmer is especially efficient and can be both built and searched in time and space that is linear in proportion to the input sequence length.

Beyond pairwise comparisons, there are numerous tools for the alignment for multiple of whole genomes [69,70,71,72,73]. Alignment accuracy and assessment of quality remains a challenge in whole genome alignment [74,75,76]. For divergent sequences, alignment accuracy is difficult to assess and popular methods disagree, such was demonstrated by the relatively low level of agreement between alignment of non-coding regions in mammals [77,78]. Given the difficulties in assessing accuracy, recent development has included methods that are statistically motivated and show improved specificity [42,69].

At shorter evolutionary distances with large fractions of identical sequences, there is less ambiguity in alignment outcomes. Yet, despite relatively short chromosome lengths for bacteria, typically a few million base pairs, the computational complexity of multiple sequence alignment makes it a formidable challenge. Calculation of multiple alignments with a simple sum of pairs scoring scheme is known to be an NP-hard problem [13], which makes calculation of an exact solution infeasible for large inputs. All multiple genome alignment tools rely on heuristics to achieve reasonable run times. Popular heuristics include calculation of multi-genome anchors [69,79] followed by chaining of syntenic anchors [69,79,80,81].

Computational complexity is only one challenge for the comparison of numerous whole genomes. Alignment tools must handle a rich complement of genetic variation, including mutations, rearrangements, gain and loss events, and duplications. In this dissertation, we are interested in tools that do not require a reference genome and can readily accept mixtures of completed and assembled draft genome data. The requirement for a single reference genome is not always practical given sampling bias and intra-species diversity [82]. Among current tools, Enredo-Pecan [69] and MLAGAN [71] are the only ones that both report duplications and do not require a reference genome. The Threaded Blockset Aligner (TBA) [70] also does not require a reference genome for calculating the alignment, but it produces many short local alignments that require ordering against a reference genome. Progressive Mauve [73,83] utilizes maximally unique matches (MUMs) and does not require a reference, however Mauve does not currently report duplications. M-GCAT is a whole genome alignment tool that also utilizes MUMs and has been shown to be computationally efficient for the alignment of closely related genomes [84] but is biased towards a reference genome.

Graph-based methods have been widely employed for pairwise and multiple alignment of long sequences [85,86]. The segment-based progressive alignment approach implemented in SeqAn::T-Coffee [87] utilizes an alignment graph scored for consistency and a progressive alignment scheme to calculate multiple alignments. In brief, an alignment graph is composed of vertices corresponding to non-overlapping genomic regions with edges indicating matches between regions. The alignment graph can be built efficiently for multiple sequences from a set of pairwise alignments and is scored for consistency. Consistency scoring has been demonstrated to perform well in resolving problems in progressive alignment [88,89]. A multiple alignment can then be derived from the graph using an efficient heaviest common subsequence algorithm [90]. A noteworthy property of the alignment graph is that it is efficient for representing highly similar sequences. Each genomic segment that aligns without gaps in all pairwise alignments is represented as a single vertex in the graph. This property offers an advantage for comparisons of genomes with significant sequence identity because long gap-free regions are stored as a single vertex in the alignment graph. Since the number of vertices and edges in the alignment graph is a function of the genetic diversity of the sequences and not the sequence lengths, this method allows for a compact representation and fast alignment of very long and highly similar sequences. A limitation of the SeqAn::T-Coffee tool is that it is restricted to aligning collinear sequences that are free of rearrangements.

In Chapter 5, we describe a method for distributed whole genome multiple alignment to enable parallel computation. There are few tools for distributed multiple alignment, none of which are described for whole genomes. Cloud-Coffee implements a distributed consistency-based scoring for parallel multiple alignment of collinear regions and has been benchmarked on Amazon EC2 [91].

(Pan-)genome annotation and analysis

In Chapter 6, we describe a novel method for improving annotation across a species pan-genome. While there are several tools for gene prediction of single genomes [47,49,92], relatively few tools exist to deal specifically with the simultaneous annotation of large numbers of nearly identical sequenced isolates, such as a species pan-genome. Also, despite low error rates in gene calling, the accumulation of errors across many genomes can cause problems for comparative analysis, such as identification of the conserved core genome [54]. Additionally, as genomes are sequenced and annotated by diverse scientists, annotations can vary due to choice of gene predictions algorithm or annotation procedures [53,93,94,95]. Post-processing can be used to identify annotation anomalies, as in GenePrimp [93].

Re-annotation efforts have been used to standardize annotation across many genomes to a single protocol [96]. This approach is particularly useful for updating out-dated annotation with the latest available evidence. A challenge for standardization efforts is combining automated re-annotation while preserving curated edits, which may include corrections of gene prediction errors. This process requires integration of both manually curated structures and *ab-initio* gene predictions.

Comparative analysis of closely related sequences forms the basis of many annotation approaches [97]. Reference-based approaches that map annotation onto new genomes using a reference [98] are particularly well-suited to annotation within a species where many genes are expected to be identical in each sequenced isolate. For some species, the use of a single reference genome can be limiting and as a result, researchers often need to integrate annotations from multiple sources. While fully automated approaches for comparison and annotation are of heightened interest as genome sequencing throughput has increased, the need for combining manual, expert curation with high-throughput automated approaches has been recognized [99].

<u>Platforms and packages for automated sequence analysis</u>

In Chapters 7-8, we describe a novel software platform for automated sequence analysis pipelines. The installation, operation, and maintenance of software tools for bioinformatics analysis can be cumbersome and require significant technical expertise leading to efforts that pre-package and bundle bioinformatics tools [17]. While, many bioinformatics software tools routinely used in sequence analysis are open source and freely available, the installation, operation, and maintenance can be cumbersome and require significant technical expertise [17,100]. In addition, individual tools are often insufficient for sequence analysis and, rather, need to be integrated with others into multi-step pipelines for thorough analysis. To aid with this, bioinformatics workflows systems and workbenches, such as Galaxy [101], Ergatis [102], GenePattern [103], Taverna [104] provide user interfaces to simplify execution of tools and pipelines on centralized servers. Prior to analysis, researchers utilizing genomics approaches are faced with a multitude of choices of analysis protocol and best practices are often poorly documented [105]. Complexities of analysis pipelines and lack of transparent protocol can limit reproducibility of computed results [18]. Use of workbenches that store pipeline metadata and track data provenance can improve reproducibility [101].

Bioinformatics service providers, such as RAST [106], MG-RAST [107], ISGA [108], and the IGS Annotation engine [109], have attempted to address challenges in microbial genome analysis by providing centralized services, where users submit sequence data to a web site for analysis using standardized pipelines. In this model, the service provider operates the online resource, dedicating the necessary personnel and computational resources to support a community of users. Bioinformatics workflow systems [101,102,103,104] also operate on central servers, utilizing dedicated or shared network based storage, and clusters of computers for improved processing throughput.

Other efforts have bundled tools into portable software packages for installation on a local computer, including Mother [110] and Qiime [111] for 16S ribosomal RNA analysis and DIYA [112] for bacterial genome annotation pipeline.

There is considerable enthusiasm in the bioinformatics community for use of cloud computing in sequence analysis [12,60,113,114]. Map-Reduce algorithms [115] using the cloud-ready frameworks Hadoop are available for sequence alignment and short read mapping [116], SNP identification [117], RNA expression analysis [118], amongst others demonstrating the usability of cloud services to support large-scale sequence processing.

In Chapter 8, we evaluate the cost and resources required for typical applications of microbial genomics. Cost considerations of using the cloud have generated debate as to the affordability of cloud based analysis [19,119]. Case studies using cloud computing platforms have been published with varying results, either favoring cloud-based over local computing in both performance and cost for microarray-based transcriptomic analysis [120] or demonstrating comparable performance parameters for cloud-based and local computing and cost advantages of local executions for metagenomics BLAST analysis [121].

Chapter 4 – Mugsy: Fast whole genome multiple alignment

Multiple sequence alignment is amongst the most widely used and studied methods for comparative analysis, providing a rich description of evolutionary relationships between sequences [33]. Yet, multiple alignment of whole genomes presents significant challenges as genome evolution introduces large-scale genetic flux and multiple alignment is NP-hard [13], making exact solutions infeasible for large data sets .

In this chapter, we present a new whole genome alignment tool, named Mugsy, which can rapidly align DNA from multiple whole genomes on a single computer. Mugsy implements a novel algorithm for identifying locally collinear blocks (LCBs) that define the regions from two or more genomes that are collinear, free of rearrangements, and suitable for multiple alignment. We demonstrate the performance of Mugsy on up to 57 bacterial genomes from the same species and the alignment of chromosomes from multiple human genomes. Mugsy accepts draft genome sequences and does not require a reference genome for calculating the alignment or interpretation of output. Mugsy integrates the fast whole genome pairwise aligner, Nucmer, for identifying homology, including rearrangements and duplications, with the segment-based multiple alignment method provided by the SeqAn C++ library. Mugsy is run as a single command line invocation that accepts a set of multi-FASTA files, one per genome and outputs a multiple alignment in MAF format. The Mugsy aligner is open source software and available for download at http://mugsy.sf.net.

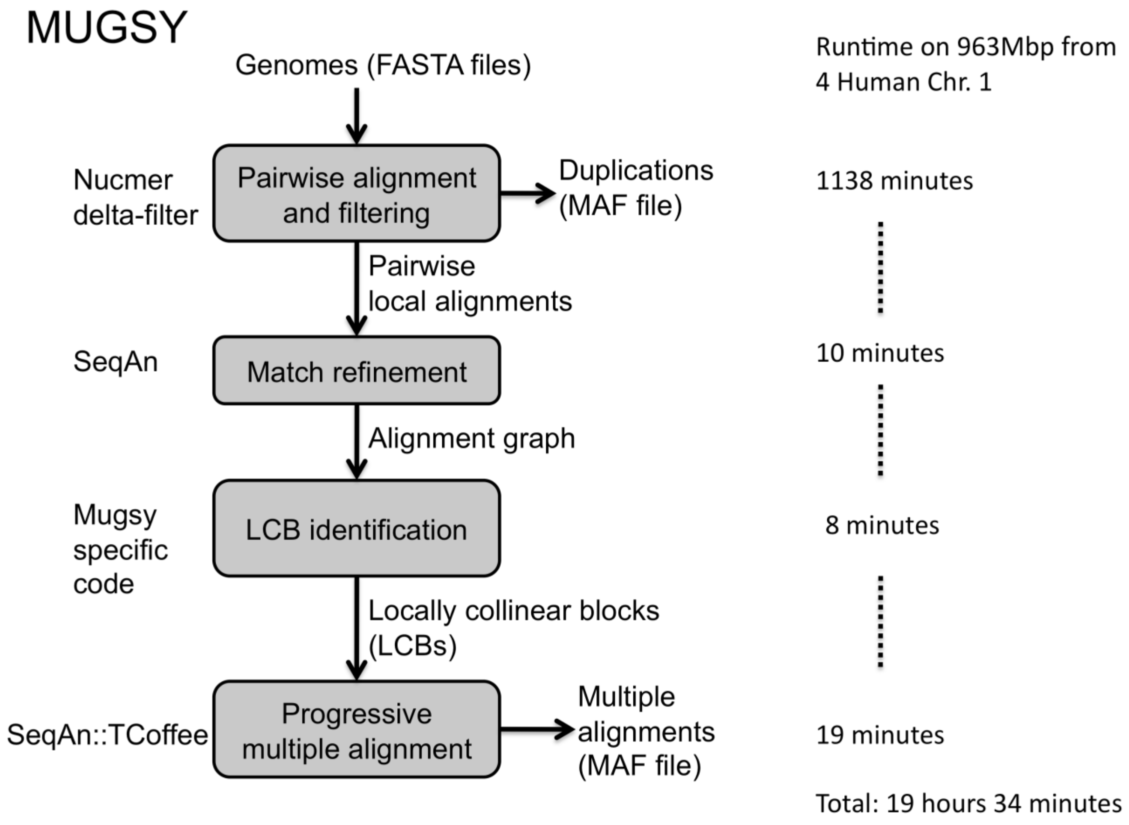

MUGSY
Genomes (FASTA files)
Runtime on 963Mbp from 4 Human Chr. 1
Nucmer delta-filter
Pairwise alignment and filtering
Duplications (MAF file)
1138 minutes
Pairwise local alignments
SeqAn
Match refinement
10 minutes
Alignment graph
Mugsy specific code
LCB identification
8 minutes
Locally collinear blocks (LCBs)
SeqAn::TCoffee
Progressive multiple alignment
Multiple alignments (MAF file)
19 minutes
Total: 19 hours 34 minutes

Figure 4.1: The process flow and primary steps of Mugsy

The key steps are listed in boxes and data types that are input and output at each step are shown adjacent to the arrows. Software used to implement parts of each step is listed on the left. The execution time of each step from an alignment of 4 human chromosomes is provided on the right. The component timings include parsing input and writing outputs. Tests were run on a single CPU of an Intel Xeon 5570 processor with 16GB of RAM

Methodology

The Mugsy alignment tool is comprised of four primary steps (Figure 4.1):

i. An all-against-all pairwise alignment using Nucmer, refined with delta-filter [68];

ii. Construction of an alignment graph and refinement [87] using SeqAn [122];

iii. Identification of locally collinear blocks (LCBs) in the graph using code we developed; and

iv. Calculation of a multiple alignment for each LCB using SeqAn::TCoffee [87].

Mugsy includes a Perl wrapper script that runs all the steps. The primary input consists of one file per genome, which may contain more than one sequence for draft genomes (i.e., a multi-FASTA file). The SeqAn library provided functions to build an alignment graph from pairwise alignments. We made three extensions to the alignment graph approach that enabled us to use it for whole genome alignments with rearrangements and genome flux. First, we utilized the pairwise alignments from Nucmer to define the segments allowing for gaps and mismatches. Second, we modified the data structure of the alignment graph to store the orientation between matching segments so that we could detect inversions. Lastly, we implement a novel method for calculating locally collinear subgraphs from the input alignment graph. These subgraphs represent locally collinear blocks (LCBs) and can correspond to inversions and regions that have been gained or lost in a subset of genomes.

Pairwise alignment and identification of duplications

The input genomes are searched using Nucmer in an all-against-all manner using a minimum match length of 15 nucleotides and a cluster length of 60 (-l 15, -c 60). Each pairwise search is subsequently processed with the "delta-filter" utility to identify matches likely to be orthologous. Delta-filter, a program included with Nucmer, limits pairwise matches to those contained in the highest scoring chain of matches calculated using a modified longest increasing subsequence (LIS) [123]. Each match is given a score corresponding to the match length multiplied by the square of the pairwise sequence identity. Pairwise matches that are present in the LIS chain for both the reference and query sequences (delta-filter -1) are saved for use in the multiple alignment and can include inversions. This filtering is critical for excluding homology to repetitive sequences. The output of delta-filter is converted to MAF format for subsequent processing.

We modified the source code of delta-filter to report duplicated segments that are present in the LIS chain of either the reference or the query genome, but not both (delta-filter –b). The duplicated segments identified for each pairwise alignment are saved as an output file in MAF format. The chaining algorithm in delta-filter is similar to Supermap which has been used to identify orthologous segments in the presence of duplications [71].

Following Nucmer and delta-filter, the remaining pairwise alignments are passed to the mugsyWGA program for multiple alignment. mugsyWGA first builds an alignment graph using the refinement approach described in SeqAn::T-Coffee [87], with the addition that the orientation of the alignment between segments is also saved. The alignment graph stores all the pairwise homology information calculated by Nucmer. Each vertex represents an ungapped genomic segment (Figure 4.2 top). Edges represent pairwise homology statements from Nucmer that pass the orthology filtering criteria from delta-filter as described above. The refinement procedure produces a minimal subdivision of segments from all pairwise comparisons ensuring the segments are non-overlapping. We modified the alignment graph to store the relative orientation of the matches as reported by Nucmer for each edge. The alignment graph is then processed to identify locally collinear blocks (LCBs).

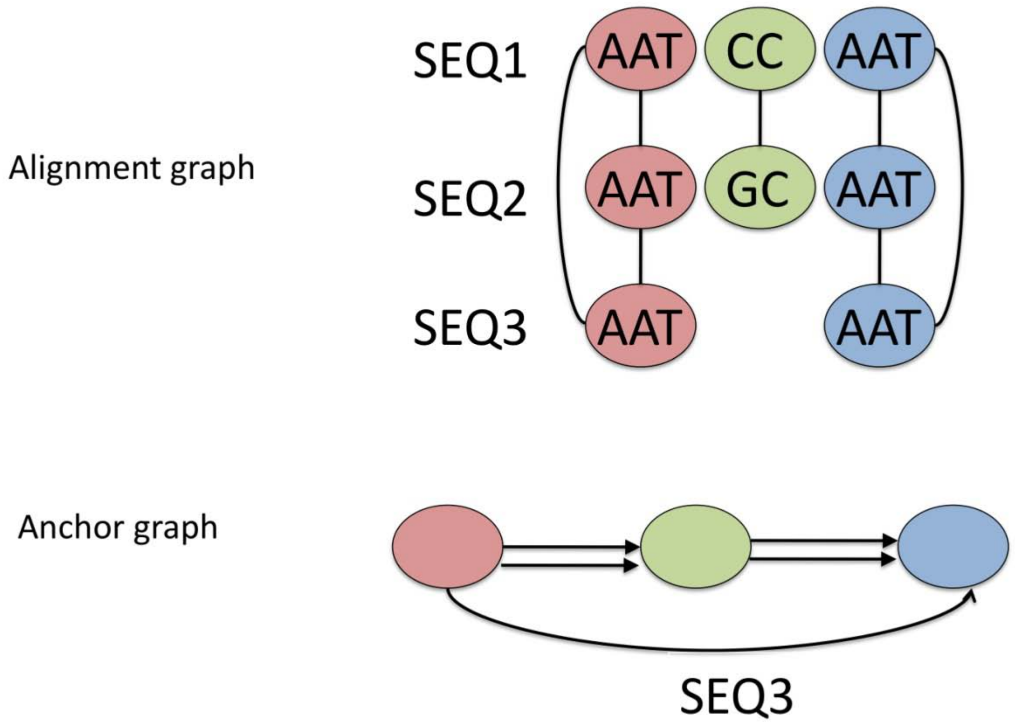


Figure 4.2: Generation of multi-genome anchors from the alignment graph
Three sequences are shown (SEQ1,SEQ2,SEQ3) with matching segments from the alignment graph (top). Connected components define three multi-genome anchors (bottom). Adjacent anchors along a sequence are connected by edges and labeled with the sequence identifier. To handle inconsistencies in the alignment graph, connected components are built in a greedy fashion traversing the most consistent edges first and restricting anchors to one alignment segment per genome (not shown). Multiple segments from the same genome are allowed only if they are within a configurable distance along the sequence.

Determination of locally collinear blocks (LCBs)

A critical step in whole genome alignment is the determination of genomic regions that are homologous, collinear, free of rearrangements, and suitable for multiple alignment. Following the terminology of Mauve [73], we refer to these segments as locally collinear blocks (LCBs). Chaining procedures are widely utilized to define genomic intervals that are consistently ordered and oriented in multiple genomes and are often labeled as syntenic [69,71,79,80,81,124,125]. In Mugsy, we implement a new graph-based chaining procedure that looks for LCBs in the alignment graph and has similarities with previous methods for defining syntenic regions. The procedure uses heuristics to define collinear regions that are free of rearrangements and large gaps, correspond to LCBs, and are suitable for multiple alignment. The procedure first builds a graph, termed the anchor graph (Figure 4.2 bottom), that enables easy identification of collinear regions by traversing simple paths comprised of anchors with exactly two incident edges (Figure 4.3a).

Micro-rearrangements and repetitive elements limit the length of these regions by introducing breakpoints in the graph. Our method attempts to extend these regions by a series of merges and filtering of short LCBs (Figure 4.3b). Our construction of the anchor graph joins anchors if any two genomes comprising the anchor are syntenic. This does not ensure all paths in the graph correspond to LCBs because of genome gain, loss, duplications and rearrangements. To resolve this, a cutting procedure is used to ensure LCBs do not traverse large-scale rearrangements and indels. The cutting procedure interprets the anchor graph as a flow network and a maximum flow, minimum cut algorithm is used to trim edges from the graph to define LCBs (Figure 4.3c). This procedure breaks the anchor graph at locations of reduced synteny and limits the length of an insertion or deletion described within an LCB.

The procedure takes two input parameters, a maximum genomic distance between adjacent anchors, *G*, and a minimum block length, *L*. The method will not identify rearrangements, including inversions, shorter than *L*. *G* and *L* are set in Mugsy using –distance and –minlength with defaults 1,000 and 30 nucleotides, respectively. The default settings were determined empirically by varying options and comparing output to other tools on limited test data (Figure 4.8-Figure 4.10). Increasing the value of *G* can help avoid fragmentation of LCBs in comparisons of divergent genomes but only had slight effect on datasets in this paper (Figure 4.10). In alignments of 11 *S. pneumoniae* genomes, the aligned core varied by 1,904 nucleotides out of ~1.59M core nucleotides aligned for values of *G* between 1,000-10,000. In the same experiment, the total aligned nucleotides varied by 141,898 out of ~63.3M nucleotides. The value of *L* can have a greater impact on results with larger values excluding short regions of homology that cannot be chained into LCBs leading to reduced sensitivity.

Identification of multi-genome anchors

The first step in determining LCBs consists of producing a set of multi-genome anchors from the alignment graph. To simplify identification of synteny, we are interested in defining anchors with a single location per genomic sequence. The anchors will be subsequently chained together to define syntenic regions. The pairwise alignments used to define segments in the anchor graph have already been filtered for orthology (using delta-filter as described in Section 2.1) but inconsistencies between pairwise alignments arising from repeats and duplications can produce paths in the alignment graph with multiple segments from the same genome. As a result, connected components in the alignment graph may contain multiple segments from a single genome. Some of these copies may be close to each other on the genome while others are not. We identify duplications during pairwise alignment, and so we are interested in generating multi-genome anchors that contain only a single segment per genome.

These anchors are calculated using a greedy depth-first search of the alignment graph ordered by consistency score, traversing the highest consistency edges first (Figure 4.2).

In cases where there are inconsistencies in the anchor graph, we track the genomic extent of each connected component and only allow multiple segments from the same genome if they are separated by less than a configurable genomic distance, --*anchorwin*. The default value for --*anchorwin* is 100 nucleotides. Other copies explored during the search define new anchors or are excluded as singletons if no incident edges remain. By setting this parameter, we are able to reduce the size of the anchor graph for further processing. In the comparison of 31 *S. pneumonia*, the number of multi-genome anchors was 264,133 using –anchorwin=0 and 239,259 using –anchorwin=100. With –anchorwin=0, each inconsistency in the alignment graph introduces a new anchor and potential breakpoint in the anchor graph. Subsequent processing of the anchor graph attempts to merge anchors that are syntenic, including anchor fragments produced by inconsistencies in the alignment graph.

The relative orientations of segments that comprise an anchor are also saved during the greedy anchor traversal. For each LCB, the edge with the highest consistency score determines the relative match orientation for its incident genomic segments. Remaining edges are considered in descending order of consistency score, assigning a relative orientation based on the Nucmer alignment orientation. The resulting anchors consist of oriented genomic segments in two or more genomes that can contain mismatches, but no gaps, as provided by the alignment graph.

Anchors derived from this method can be very short since the refinement procedure used to build the alignment graph will produce segments as short as a single base per sequence, such as in the case of a single base indel. In the comparisons of closely related genomes, segments are often much longer and the alignment graph will often have significantly fewer vertices than the total number of base pairs in the genome. The alignment graph for ~963Mbp from 4 human sequences of chromosome 1 consisted of 1,024,728 vertices with an average length of 868bp and 1,450,084 edges. The connected components in this graph resulted in 185,537 multi-genome anchors. By comparison, the alignment graph for the 31-way comparison of *S. pneumoniae* strains, comprising 65.7Mbp in total, contained 2,717,087 vertices with an average length of 23bp and 264,133 multi-genome anchors.

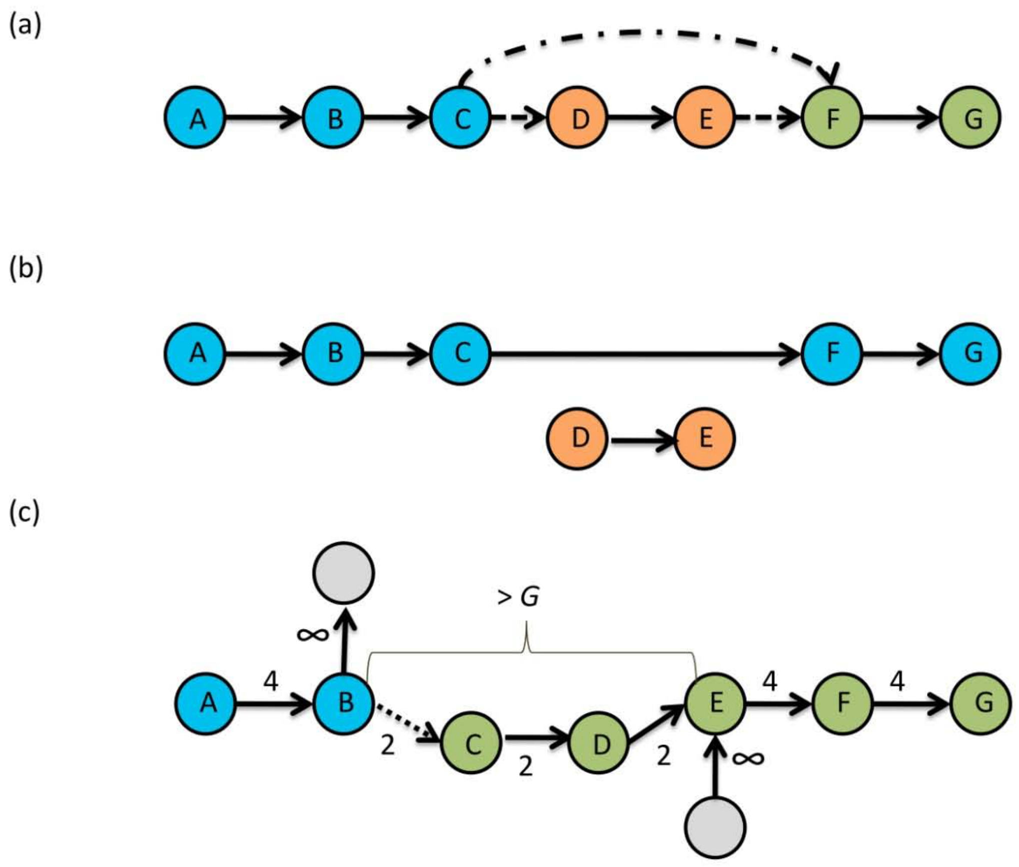


Figure 4.3: Identification of LCBs in the anchor graph

A set of multi-genome anchors labeled A-G are shown. Anchors adjacent along one or more sequences are connected by an edge. (a) Simple paths with exactly one incoming and outgoing edge correspond to collinear regions and branches correspond to syntenic breakpoints (dotted edges) resulting in three collinear regions colored blue, orange, green. (b) Merging of adjacent regions. A short component (D,E) with a genomic extent less than a configurable parameter *L* is removed from the graph. The remaining anchors form a single collinear region colored blue. (c) Cutting of paths that violate LCBs constraints with max-flow, min-cut. Anchors B and E are adjacent but non-syntenic separated by a genomic extent greater than the configurable parameter *G* in at least one sequence. The graph forms a single connected component that is an invalid LCB. To resolve this, the anchor graph is interpreted as a flow network. Edges are labeled with an edge capacity indicating the number of sequences for which the incident anchors are collinear. Source and sink vertices (grey) are added to the graph incident to vertices that violate the distance criteria. Maximum flow, minimum cut identifies the cut (dotted edge B,C) to produce two collinear regions colored blue and green. Max-flow, min-cut ensures the graph is cut to produce collinear regions that fulfill the distance constraint *G* regardless of cycles or branches in the graph.

Identification of syntenic anchors

The multi-genome anchors are used to define vertices in a new directed graph, termed the anchor graph that is used to identify boundaries of LCBs. Edges in the anchor graph connect adjacent anchors along a genomic sequence. To determine edges, the vertices are first ordered along each of the member sequences. Anchors that are immediately adjacent on at least one sequence and separated by a genomic extent less than the configurable distance *G* are linked by an edge. The edges are labeled with the names of the sequences for which the anchors are adjacent. Simple paths through this graph, comprised of vertices with exactly two incident edges, represent runs of anchors that are consistently ordered and syntenic in two or more genomes. Branches in the graph produced by vertices with more than two edges represent breakpoints in synteny. The beginning and end of an assembled contig or changes in relative orientation between anchors also represent breakpoints. An initial set of LCBs is calculated by finding simple paths in the anchor graph that do not cross any breakpoints using a depth-first search (Figure 4.3a). Some of these breakpoints will arise from micro-rearrangements, repetitive elements, or from our greedy construction of multi-genome anchors. The remaining steps of the algorithm attempt to extend the LCBs into longer regions that span these breakpoints by removing branches from the graph.

We merge LCBs that are connected by at least one edge in the anchor graph and do not traverse an inversion, indicated by a change in relative orientation between sequences in an anchor and do not introduce gaps greater than *G* in the projection along any member sequence (Figure 4.3b). Next, anchors comprising short LCBs that span less than the minimum block length, *L*, are removed from the graph. A new set of LCBs is calculated after adding new edges between adjacent anchors separated by less than the genomic distance, *G*, on two or more genomes. This resulting graph can include branches between anchors that are adjacent on some genomes but not others due lineage specific rearrangements or indels. Repetitive elements can also give rise to branches and cycles in the graph that link anchors that are not syntenic.

An additional step is used to break edges in the anchor graph so that we ensure valid LCBs. This step models the anchor graph as a flow network and uses a maximum flow, minimum cut algorithm [126] to find bottlenecks in the graph that are used to partition connected components that violate criteria for LCBs. Flow networks have been previously used in other areas of alignment, including the consistency problem in multiple alignment [127]. To build the flow network, the LCBs are ordered on each member sequence and checked for gaps greater than distance *G* or paths that join multiple contigs from the same genome.

Sets of vertices that violate these criteria are deemed non-syntenic and added to opposing source and sink vertices in the flow network (Figure 4.3c). We define the edge capacity of the network as the number of sequences for which any two incident anchors are adjacent and syntenic. We compute maximum flow, minimum cut using an implementation of the Ford-Fulkerson algorithm [128] to identify a minimum set of cut edges that partitions the graph ensuring the non-syntenic source and sink vertices are disconnected. This in turn ensures the LCBs consist of anchors that fulfill the maximum gap criteria and contain a single contig per genome in the case of draft genomes. The use of the max-flow, min-cut provides a valid partition even if multiple cuts are required to ensure a valid LCB due to branching in the anchor graph. This max-flow, min-cut procedure using conserved synteny as the edge capacity has the property that it will attempt to split the LCB at bottlenecks represented by edges with reduced synteny.

The max-flow, min-cut, calculation accounted for ~12.5 minutes of 116 total minutes for the LCB identification in 57 E. coli. For these genomes, the anchor graph was composed of 675,780 multi-genome vertices and 1,258,603 edges.

Finally, the extent of the LCBs is determined from the coordinates of the minimum and maximum anchor coordinates on each member sequence. The subset of vertices in the alignment graph that overlap the extent and connected edges are passed to SeqAn::T-Coffee to align each LCB. The LCB identification procedure can produce overlapping LCB boundaries with the extent of the overlap determined by the distance parameter *G*. To place each anchor in exactly one LCB, the LCBs are sorted by length in descending order and anchors are removed from the anchor graph as they are aligned into LCBs.

The resulting multiple alignments are saved in MAF format for each LCB. The construction of the alignment graph and progressive alignment algorithm using SeqAn::TCoffee is implemented in C++ using the SeqAn library [122]. The LCB identification procedure is written in C++ using the Boost library (http://boost.org).

Evaluation of whole genome alignment tools

To compare Mugsy to other multiple whole genome alignment tools, we downloaded Mauve, TBA, FSA, MLAGAN, and Pecan from their project web sites. The MLAGAN/SLAGAN and Pecan/Enredo tools do not provide scripts that automate all of the steps required to generate whole genome alignments from a set of input FASTA files. Also, previous analyses of mammalian genomes using these tools in [69,71] utilized a compute grid to execute the pairwise alignment step. This makes generation of whole genome alignments from a set of genomic FASTA files cumbersome. To compare these tools with Mugsy on a single computer, we limited our evaluation to only the collinear alignment components, MLAGAN and Pecan, and used Mugsy to define a common set of LCBs for evaluation. The extents of the LCBs were first calculated by Mugsy and saved as multi-FASTA files that were passed as input to MLAGAN or Pecan. MLAGAN and Pecan were run with default parameters. We did not attempt to execute the SLAGAN that defines collinear regions for MLAGAN.

Mugsy LCBs were also used to define the genomic extent of the regions passed to the multiple alignment program FSA. FSA was run using the recommended fast alignment options –fast, -noindel2, -refinement. Mugsy includes an option to invoke the FSA aligner on each LCB as a part of a post-processing step.

Mauve alignments were generated directly from the genomic FASTA files using progressiveMauve 64-bit binary version 2.3.1 with default command line options [83]. The Mauve output format was converted to MAF format to compare with the outputs of Mugsy.

TBA was run with default options using MAF formatted pairwise alignments from Nucmer instead of BLASTZ. The Nucmer alignments were processed with delta-filter and identical to those used as inputs for Mugsy. By using the same pairwise alignments, we were able to focus our evaluation on the multiple alignment portion of Mugsy compared to TBA. The runtime values generated are the shortest successful runtime of three tests for all tools evaluated.

For comparing outputs between Mugsy, Mauve, TBA, comparisons were restricted to completed genomes to simplify projecting pairwise alignments onto a reference coordinate system. Output files were converted to MAF format if necessary. The utility "compare" downloaded from http://www.bx.psu.edu/miller_lab was used to calculate precision, recall, and percentage agreement between alignment outputs.

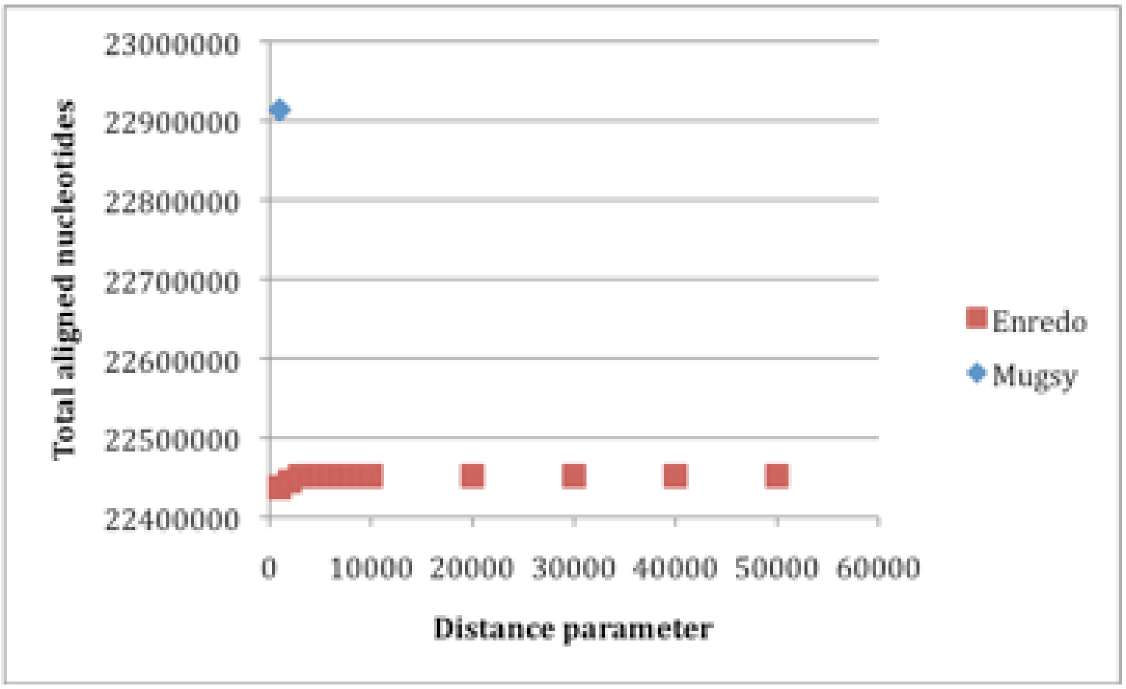


Figure 4.4: Comparison of total aligned nucleotides between Enredo and Mugsy
Alignment anchors from 31 *S. pneumoniae* genomes were used as input. Enredo was run with varying values of –max-gap between 1,000-50,000 in an attempt to recover more of the alignment. For comparison, a single Mugsy run with –distance 1000 is displayed on the plot in blue. Other parameters to Enredo were also evaluated but did not improve the results (Not shown).

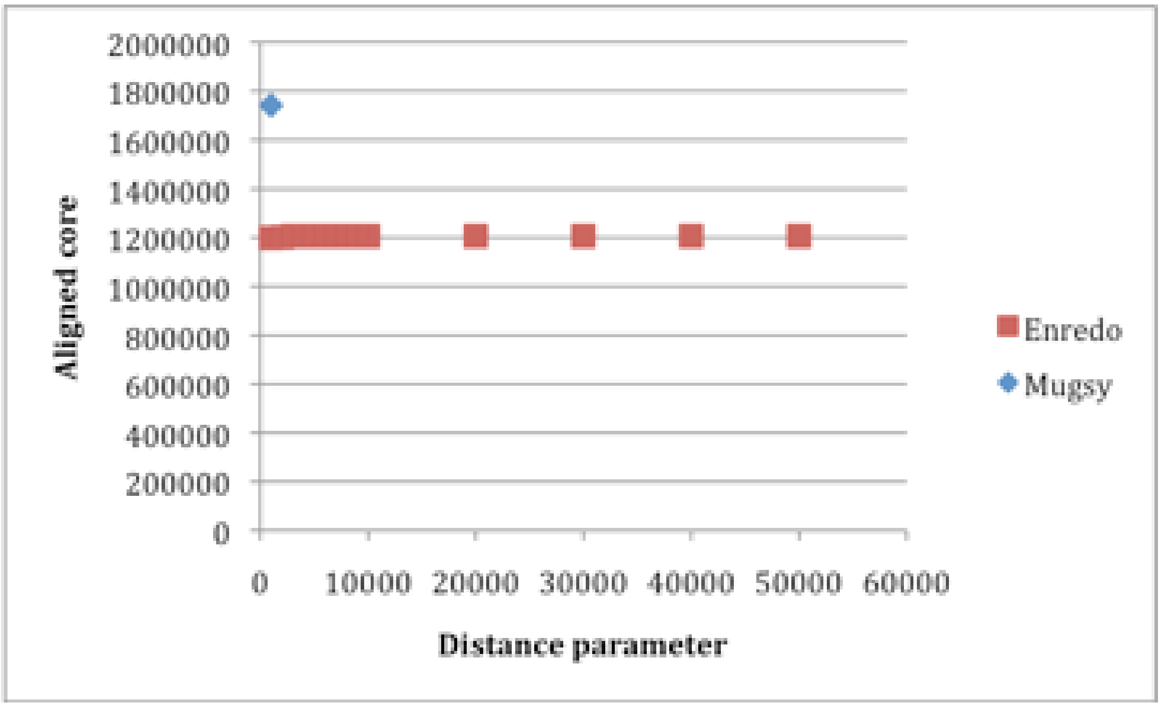


Figure 4.5: Comparison of total aligned core nucleotides between Enredo and Mugsy
Alignment anchors from 31 *S. pneumoniae* genomes were used as input. Enredo was run with varying values of –max-gap between 1,000-50,000 in an attempt to recover additional alignment. For comparison, a single Mugsy run with –distance 1000 is displayed on the plot in blue. Other parameters to Enredo were also evaluated but did not improve the results (Not shown).

In a separate analysis, we compared the extent of LCBs calculated by Mugsy with the segmentation produced by Enredo [69]. Enredo reports locally collinear blocks (LCBs) from a set of externally generated anchors that occur in two or more input genomes. We first calculated multi-genome anchors from the alignment graph of 11 completed *S. pneumoniae* genomes as described in Methods. The set of multi-genome anchors was used as input to both Enredo and Mugsy. Enredo was run with options –min-score=0, –min-length=0 and –max-gap=3000. Additional runs were performed varying–min-length between 0 and 100 and varying –max-gap-length between 1,000 and 50,000 (Figure 4.4, Figure 4.5).

Table 4-1. Summary of genomes compared using whole genome alignment

Organism	Number of genomes	Number of sequences	Total bases (Mbp)
N. meningitidis	5	5	10.9
S. pneumoniae	31	1 906	65.7
E. coli	57	4 213	299.1
Human Chr I	4	4	963.2

For genomes in draft form, the total number of assembled contigs or scaffolds is provided in column 3.

Data sets

The *S. pneumoniae*, *E. coli* and *N. meningitidis* genomes were downloaded from the NCBI Entrez website [129]. The accessions and species names are provided in (Supplemental Table S1 in [1]). The human genome sequences were downloaded from the individual project web sites: the NCBI reference GRCh37 available from UCSC as hg19 from http://genome.ucsc.edu, the Venter genome (JCV) from http://huref.jcvi.org [130], the Kim Sungjin (SJK) genome from http://koreagenome.kobic.re.kr/en/ [131], and the YanHuang project (YH) from http://yh.genomics.org.cn [132]. The SJK genome utilized the NCBI reference to build consensus sequences as described in [131]. The *de novo* assembly of YH Li *et al.* [133] was not available as a consensus scaffold that spans chromosome 1. Instead, we utilized a consensus sequence for YH from Beijing Genomics Institute that was based on the UCSC build hg18 (NCBI v36) and is available as a single scaffold spanning chromosome 1 on the project web site (http://yh.genomics.org.cn). We choose to align these sequences to demonstrate the performance of Mugsy on the multiple alignment of very long sequences.

Single nucleotide variants (SNVs) were obtained from the personal variant tracks of UCSC browser [134] and included these sources: JCV [130], YH [132], SJK [131], and dbSNP 130 [135]. The personal variant tracks provided the variant data in a common format with coordinates on a single version of the reference genome, hg19, which was used for multiple alignment with Mugsy. This allowed for comparison of the published variants for each individual even though some of the published studies were generated on consensus sequences prior to hg19.

Table 4-2 Processing time to calculate whole genome multiple alignments using three methods

	5 *N. men*	31 *S. pneumo*	57 *E. coli*	4 Human Chr 1
Pairwise search	3 min	44 min	435 min	1138 min
+Mugsy	3 min	56 min	720 min	37 min
+TBA	<1 min	36 min	381 min	71 min
Mauve v2.3.1	5 min	377 min	DNF (1)	DNF (2)

The runtime in minutes for the pairwise search includes aligning all pairs of genome sequences with Nucmer, post-processing with delta-filter, and converting output formats to MAF as described in Methods. The time provided for Mugsy and TBA is the runtime for generating the multiple alignment from the pairwise search results. The time for Mauve is the total runtime. Nucmer was run with parameters MUM length –l 10, cluster length –c 60 and all other default options. Mugsy was run with parameters --distance=1000 and --minlength=30. Mauve and TBA were run with default options. Tests were run on a single CPU of an Intel Xeon 5570 processor with 16GB of RAM. DNF(1): did not finish after 2 days of processing. DNF(2): generated an allocation error.

Results

Alignment of multiple bacterial genomes

We computed whole genome alignments using Mugsy and compared runtimes to other popular whole genome alignment tools. The input genomes consisted of a mixture of completed and draft sequences with most genomes represented in multiple contigs (Table 4-1). Mugsy had the second fastest runtime, requiring less than 2 hours for the alignment of 31 *Streptococcus pneumoniae* genomes and ~19 hours for the alignment of 57 *Escherichia coli* genomes (Table 4-2). Nucmer+TBA had the fastest total runtime on this same dataset. Mugsy and TBA were the only two tools evaluated that completed the alignment of 57 *E. coli* in less than two days of processing on a single CPU. The step in Mugsy that identifies locally collinear blocks (LCBs) contributed ~15 of 56 minutes for the *S. pneumoniae* multiple alignments and ~116 of 720 minutes for the *E. coli* multiple alignment.

We ran additional comparisons of runtimes with MLAGAN[71] and Pecan [69] whole genome multiple alignment tools and the collinear alignment tool FSA[42]. For this comparison, a single set of LCBs was first calculated by Mugsy to define genomic extents for multiple alignment by MLAGAN, Pecan, and FSA. Of these three tools, only FSA completed the alignment of all LCBs in the 57 *E. coli* genomes in less than two days of processing on a single CPU. FSA is a fast method for aligning long sequences [42] but it is restricted to aligning collinear segments that are free of rearrangements. The runtime of FSA was slightly faster (896 minutes) than the combined runtime of Nucmer and Mugsy (1155 minutes).

Table 4-3 Precision and recall in aligned positions in a comparison of tools across 11 complete S. pneumoniae genomes

	Mugsy as truth	Mauve as truth	TBA as truth
Mugsy		0.97,0.99 (0.96)	0.99,0.99 (0.98)
Mauve	0.99,0.97 (0.95)		0.98,0.97 (0.95)
TBA	0.99,0.99 (0.98)	0.97,0.98 (0.96)	

The precision and recall for each row was calculated using the output of the tool listed in the column as a hypothetical true alignment. The percentage agreement is provided in parenthesis calculated as the fraction of aligned positions that are identical in the projection of all pairwise alignments inferred from the multiple alignment.

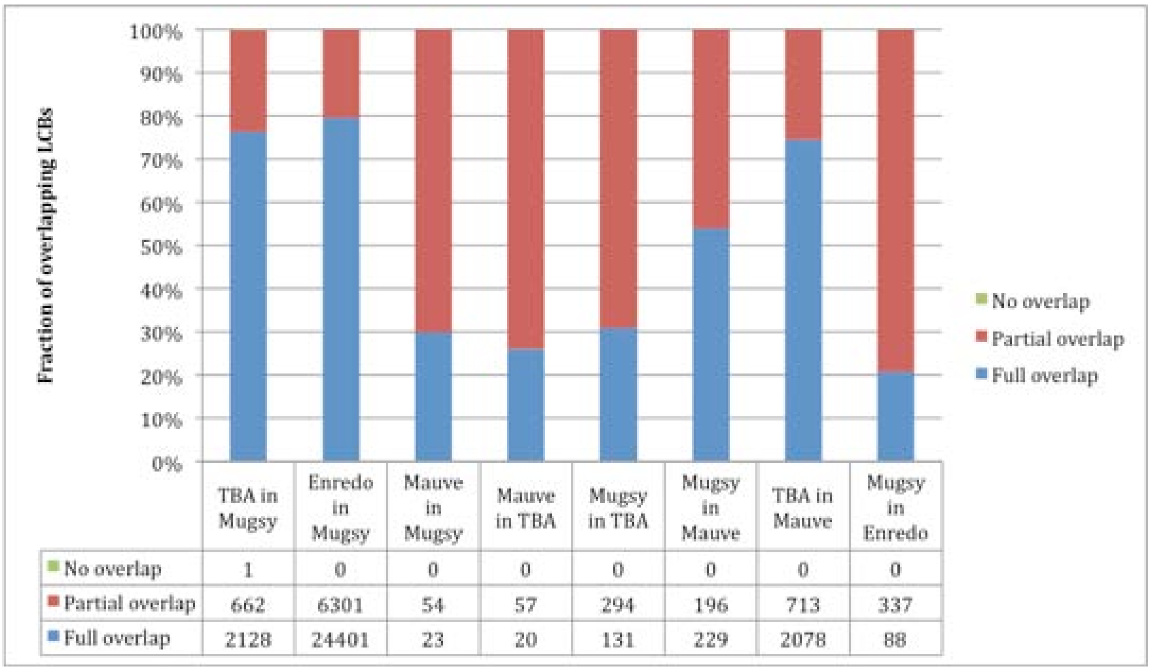
The alignment positions calculated by Mugsy show agreement with those reported by Mauve and TBA. We evaluated the agreement using a projection of pairwise alignments using one of the reported outputs as a hypothetical true alignment in a comparison of 11 complete genomes in the *S. pneumoniae* data set. Mugsy alignments scored a precision and recall of 0.99, 0.99 and 0.97, 0.99 using TBA and Mauve respectively as truth in this comparison (Table 4-3).

Table 4-4. Summary of the whole genome multiple alignment of 31 strains of S. pneumoniae using three different methods

	Number of LCBs	Length core (bp)	Core LCB N50 (bp)	Nucleotides aligned
Mugsy	2 394	1 590 820	2 044	63 294 709
Mauve v2.31	1 366	1 568 715	2 759	62 714 295
Nucmer+TBA	27 075	1 475 575	705	64 698 581

Each method reports a series of alignments that correspond to locally collinear blocks (LCBs). The length of the aligned core is the total number of alignment columns that contain all input genomes and no gap characters. Half of the aligned core is contained in LCBs spanning genomic regions longer than the core LCB N50 length. The total number of aligned nucleotides is obtained by counting bases aligned to at least one other genome in the multiple alignment.

Mugsy aligned slightly more nucleotides than Mauve in almost double the number of LCBs for the full *S. pneumoniae* dataset (Table 4-1). Mugsy also identified a slightly longer core alignment. The aligned core is comprised of alignment columns that contain all input genomes and no gaps. The combination of Nucmer+TBA aligned more total nucleotides but a shorter and more fragmented core (Table 4-4, core N50).



	TBA in Mugsy	Enredo in Mugsy	Mauve in Mugsy	Mauve in TBA	Mugsy in TBA	Mugsy in Mauve	TBA in Mauve	Mugsy in Enredo
🟩 No overlap	1	0	0	0	0	0	0	0
🟥 Partial overlap	662	6301	54	57	294	196	713	337
🟦 Full overlap	2128	24401	23	20	131	229	2078	88

Figure 4.6 - Overlap of LCBs between tools

The number of LCBs is reported in a comparison of 11 complete *S. pneumoniae* genomes that are either partially or fully contained in LCBs in each of Mugsy , Mauve, TBA, and Enredo as a reference. Mugsy was run with –minlength=30, --distance 1000. Mauve and TBA were run with defaults. Enredo was run with options –min-score=0, –min-length=0, and –max-gap-length=3000.

The length and number of aligned regions was the primary difference in output between Mugsy, Mauve, and TBA in our evaluations. Mauve produced LCBs with the longest average length (Table 4-4, Figure 4.6) but did not complete the alignment of the largest data sets used in this evaluation in the allotted time. In the comparison of 11 completed *S. pneumoniae* genomes, Mugsy LCBs either shared boundaries or partially overlapped all of the Mauve LCBs (Figure 4.6)

Mugsy reported longer alignments than TBA on average (Table 4-4, Figure 4.7). Mugsy LCBs contained all but one of the shorter TBA blocks in a comparison of 11 completed *S. pneumoniae* genomes. 76% of all TBA blocks (2128 of 2791) were fully contained within longer Mugsy LCBs (Figure 4.6). By comparison, 25% of Mugsy LCBs (20 of

77) shared identical boundaries or were spanned by longer blocks in TBA. Slightly fewer TBA blocks were contained in Mauve than Mugsy, 2078 versus 2128.

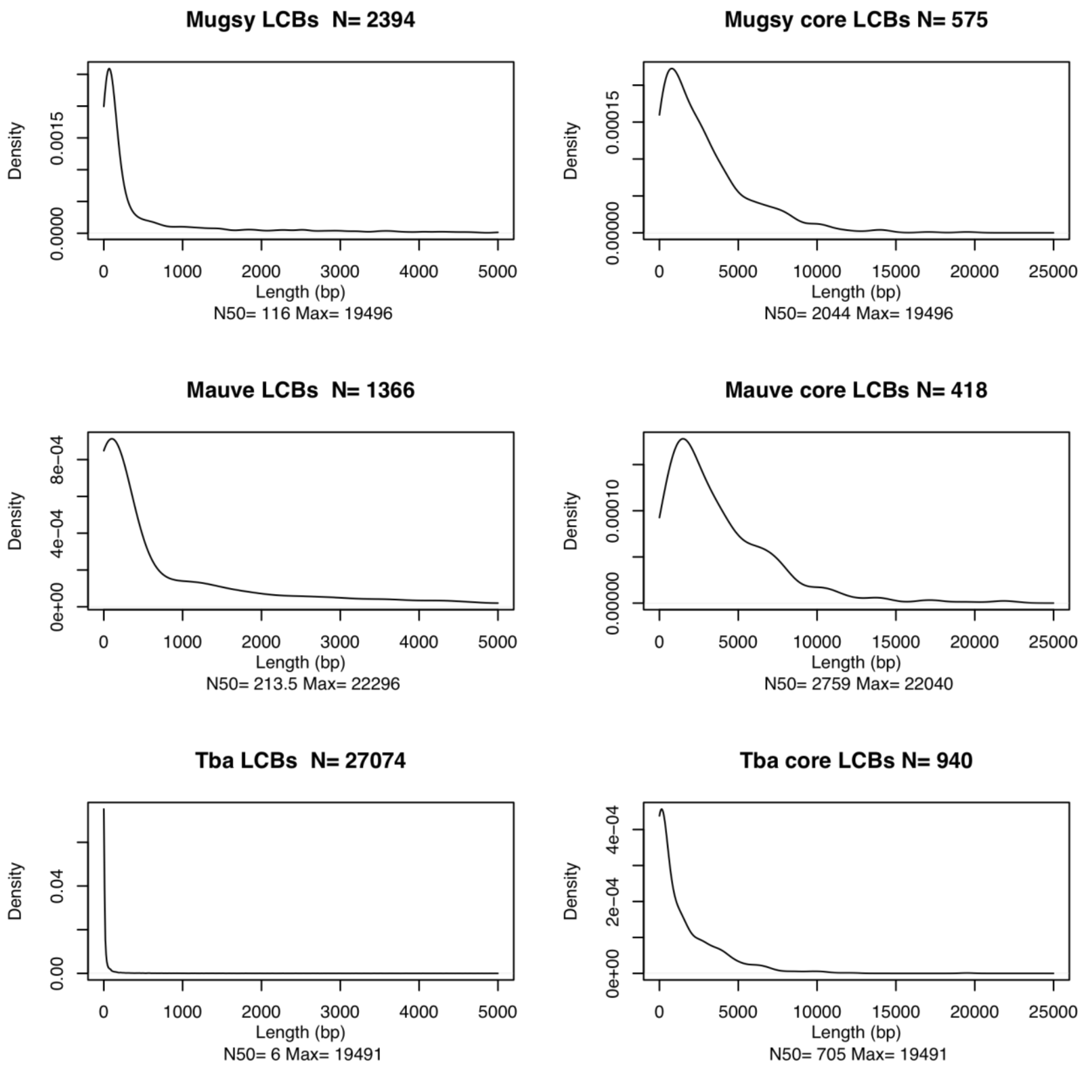


Figure 4.7 - Length distribution of total and core LCBs
Calculated by Mugsy, Mauve, and TBA in the alignment of 31 *S. pneumoniae* genomes

The differences in LCB composition and boundaries are also indicated by the lengths of the contained gaps (indicating an insertion or deletion event) reported by each tool. The longest gap lengths present in a LCB for Mugsy, Mauve and TBA were 31,130 bp, 34,910 bp, and 177 bp respectively in the comparison of 31 S. pneumonia genomes.

To further evaluate our method, we compared the LCB identification step in Mugsy with Enredo, another graph based method that has been demonstrated on comparisons of mammalian genomes [69]. Mugsy calculated a longer aligned core and incorporated more anchors into LCBs than Enredo using a set of anchors from 11 completed *S. pneumoniae* genomes. Mugsy calculated a total of 425 LCBs comprising 22,913,396 aligned nucleotides (98.6% of input) compared to 30,710 LCBs from Enredo comprising 22,451,622 aligned nucleotides (95.4%) (Figure 4.4). The Mugsy core LCBs consisted of 1,741,704 nucleotides versus 1,229,583 nucleotides with Enredo (Figure 4.5). The Mugsy LCBs were also longer than Enredo on average, with 79% (24,401 of 30,710) of Enredo LCBs sharing identical boundaries or fully contained within longer Mugsy LCBs (Figure 4.6). By comparison, 20% (88 of 425) of Mugsy LCBs shared boundaries or were fully contained in longer LCBs reported by Enredo. Increasing the distance parameter in Enredo did not improve results (Figure 4.4). The relatively short and fragmented regions reported by Enredo may be due to the composition of the multi-genome anchors used in our comparison. As described in Methods, the multi-genome anchors vary in length and can be subdivided during the segment refinement procedure to as short as a single base. Enredo has been previously reported to work well on longer anchors (>50bp in [69]).

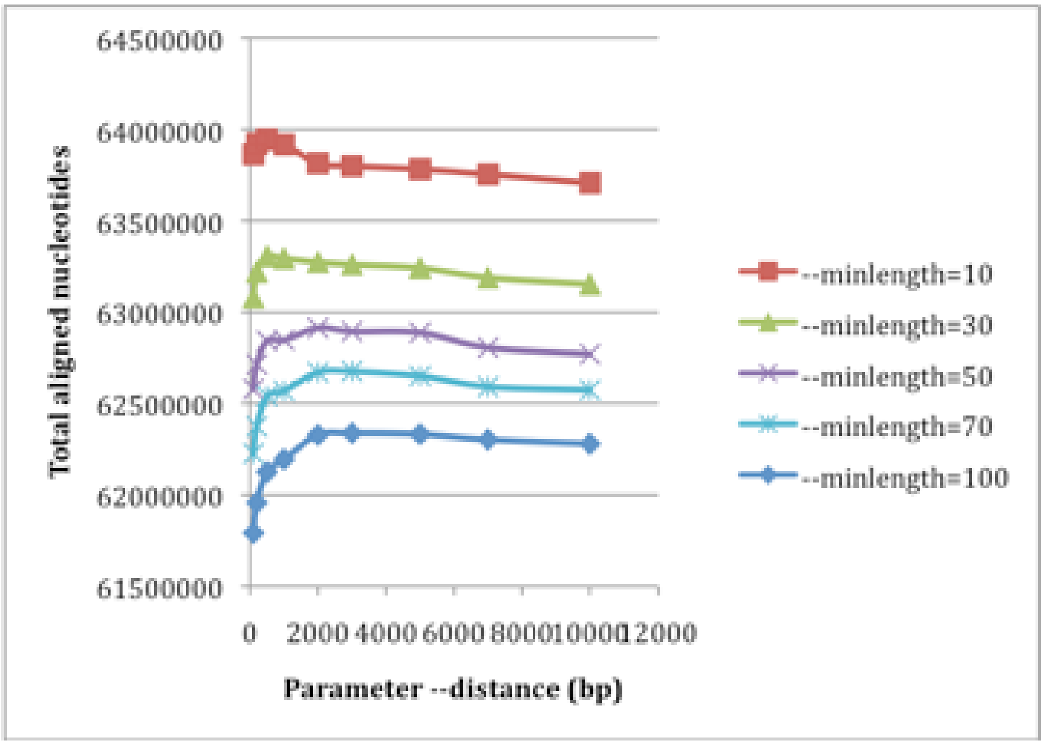


Figure 4.8 - Total aligned nucleotides for varying parameters

As calculated by Mugsy for values of *G* (-distance) and *L* (-minlength) in an alignment of 31 *S. pneumoniae* genomes. The algorithm loses sensitivity for increasing values of -minlength as short regions of homology are excluded from consideration. Sensitivity initially increases with values of –distance and then decreases slightly. –minlength=30, -distance=1000 are the defaults for Mugsy and were used for the evaluations in paper.

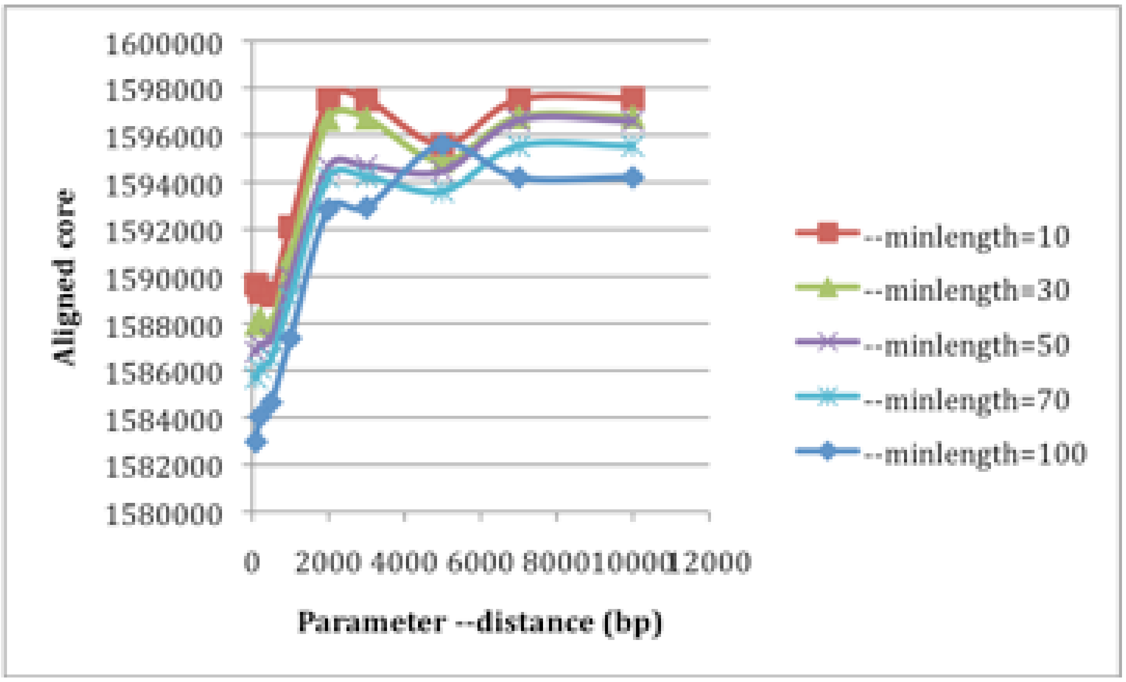


Figure 4.9 - Nucleotides in core LCBs for varying parameters

As calculated by Mugsy for values of *G* (–distance) and *L* (--minlength) in an alignment of 31 *S. pneumoniae* genomes.

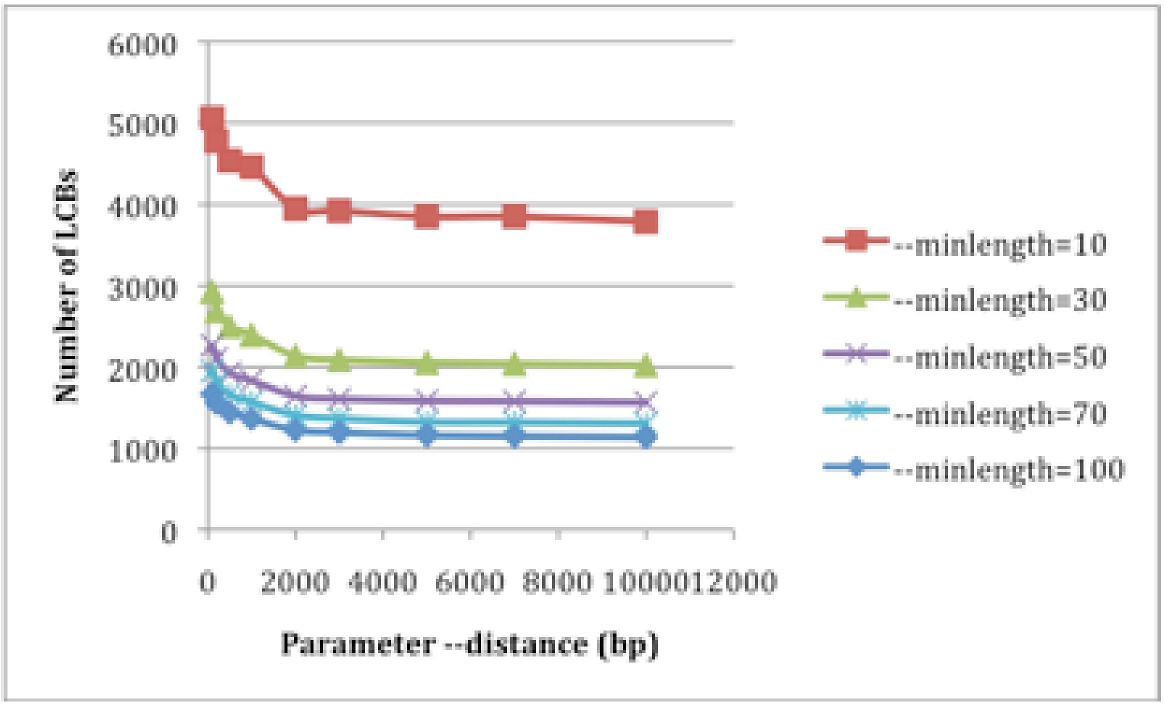


Figure 4.10 - Number of LCBs for varying parameters

As calculated by Mugsy for values of *G* (–distance) and *L* (--minlength) in an alignment of 31 *S. pneumoniae* genomes.

Mugsy includes a step for building longer syntenic regions, LCBs, from shorter multi-genome anchors. The longer aligned regions simplify some downstream analysis, such as the identification of orthologous genes and mapping of annotations, thereby minimizing the need for a reference genome. Longer alignments also aid the inspection of genomic regions that have been gained or lost and span multiple genes without requiring a reference genome. Increasing the value of the –distance parameters in Mugsy produces longer LCBs, although with slight loss of sensitivity (Figure 4.8). Our greedy method for building multi-genome anchors can introduce branches in the anchor graph in cases where there are inconsistencies in combining pairwise alignment. Our LCB identification algorithm aims to reduce this fragmentation but remains an area that can be improved. Contig boundaries will also cause fragmentation in Mugsy LCBs. As a result, introducing draft genomes will automatically increase the number of LCBs.

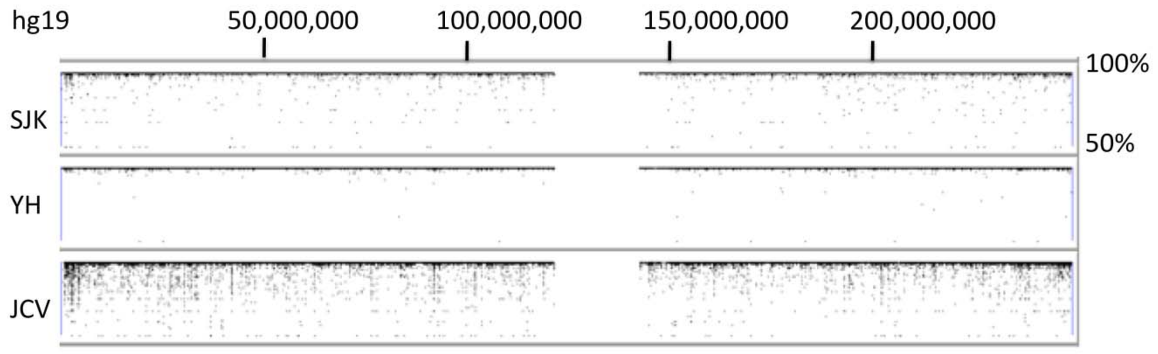


Figure 4.11 - Percent identity plots for the multiple alignment of human chromosome 1
Mugsy generated alignment for four individuals. The plots were obtained from the alignment viewer, GMAJ, using hg19 as reference for the display (top coordinates). A percent identity plot is displayed in subsequent rows for each of the three other genomes SJK, YH, JCV. The alignments span 99.9% of the nucleotides on chromosome 1 of the NCBI reference hg19, excluding the centromere, which is shown as a gap in the middle of the figure. The percent identity in each row ranges from 50 to 100 from the bottom to top of each row

Alignment of multiple human genomes

To evaluate the performance of Mugsy on larger sequences, we aligned multiple individual chromosomes from human genomes. We identified four human genomes for which consensus sequences are available for each chromosome: the NCBI reference human genome build GRCh37 (hg19 at the UCSC genome browser) [136], a western European individual (JCV) [130], a Korean individual (SJK) [131], and a Han Chinese individual (YH). Mugsy was able to align all four copies of chromosome 1 in less than one day using a single CPU (Table 4-1). Mugsy computed the multiple alignments in less than one hour (37 minutes) after completing the pairwise searches with Nucmer. The contribution of the LCB identification step in Mugsy was ~7 minutes. By comparison, TBA ran in 71 minutes using the same pairwise alignments as input. Realignment of all the LCBs from Mugsy with the FSA aligner ran in 358 minutes. Three other whole genome alignment tools evaluated (MLAGAN, Pecan, and Mauve) failed to complete an alignment of the four human chromosomes in less than two days of processing time. The length of the chromosomes (>219Mbp each) and amount of repetitive DNA in the human genome makes whole genome alignment especially challenging. The genomes were not masked for repetitive elements.

Mugsy calculated 526 locally collinear blocks (LCB) on chromosome 1 with the longest LCB spanning 5.97 Mbp on all four individuals. The LCBs covered 224,975,484 of 225,280,621 (99.86%) nucleotides in the NCBI reference sequence, hg19. The alignment viewer GMAJ was used to generate pairwise percent identity plots projected from the Mugsy multiple alignment (Figure 4.11). The plots show variation in JCV, SJK, and YH versus the reference sequence, hg19. The sequences for YH and SJK both utilized the

NCBI reference in building the consensus sequences, and therefore this comparison may under-represent the variation in these genomes. The percent identity plots indicate this possible artifact, showing relatively low variation in the comparison of YH and SJK versus hg19.

Table 4-5 Number of single nucleotide variants (SNVs) detected by Mugsy in the multiple alignment of human chromosome 1 from four individuals

Individual genomes	Mugsy SNVs	SNVs at UCSC	Recall from UCSC	Precision from UCSC or dbSNP
JCV	216 201	108 767	104 684 (90%)	194 616 (90%)
SJK	135 070	113 708	112 032 (98%)	128 473 (95%)
YH	114 871	104 590	103 106 (98%)	113 641 (99%)

Mugsy alignments were performed on consensus sequences of human chromosome 1 as provided by each source. SNVs were obtained from alignment columns where the consensus nucleotide in JCV, SJK, or YH differed from the nucleotide in hg19. An additional filter was applied to screen out alignment columns that contained gaps within 5 positions on either side. Published SNVs for JCV, SJK, or YH were obtained from UCSC personal variant tracks, restricted to homozygous variants where annotated. Recall (column 4) is the number of Mugsy variants that match UCSC divided by the total number of UCSC variants. Precision measures the number of Mugsy variants that match either UCSC or dbSNP variants divided by the total number of variants reported by Mugsy.

The whole genome multiple alignments produced by Mugsy were parsed to extract variants, including mutations, insertions, and deletions. Single nucleotide variants (SNVs) were extracted from ungapped alignment columns with more than one allele and compared to published variations in the personal variant tracks of the UCSC genome browser. Many of the SNVs calculated by Mugsy are also reported at the UCSC browser or dbSNP (Table 4-5). Mugsy calculates variation on assembled consensus sequence and does not consider the composition or quality of the underlying sequencing reads that contributed to the assembly. We restricted the comparison to variants annotated as homozygous for the individual using the UCSC browser. Additional variation identified by Mugsy may be due to differences in detection methods or assembly artifacts in the consensus.

Discussion

Mugsy implements a new procedure that identifies locally collinear blocks (LCBs). The graph utilized for the LCB identification and segment-based multiple alignment is compact for highly conserved sequences allowing for efficient computation. This makes Mugsy especially well suited to classification of species pan-genomes and other intra-species comparisons where there is a high degree of sequence conservation. Alignment of many large, highly conserved sequences, such as human chromosomes, is likely to become increasingly popular as improvements in sequencing and assembly technologies allow for de-novo assembly of human genomes, including assembly of haplotypes.

Our work relies heavily on two open source software packages, the suffix tree-based pairwise aligner Nucmer [68] and the segment-based alignment approach of SeqAn::TCoffee [87]. We utilized Nucmer to quickly build a library of pairwise homology across all input genomes. Our work extends methods in SeqAn::TCoffee to accommodate whole genome multiple alignment with rearrangements and duplications.

Mugsy relies on a number of parameters including minimum MUM length in Nucmer and the LCB chaining parameters. Careful choice of parameters is likely to be important for alignments at longer evolutionary distances. Automatically determining parameters or providing user guidance on parameter choice is an area that needs improvement.

Also, for more divergent genomes, the performance advantages of the segment-based alignment approach decrease as the length of the conserved segments shorten and the size of the alignment graph grows. The alignment of 57 *E. coli* strains required slightly more than 12 GB of RAM to build and process the alignment graph. The larger memory requirement of Mugsy on more divergent genomes is a limitation of the tool and an area that we may be able to improve at the expense of longer runtimes.

Chapter 5 - Distributed whole genome multiple alignment

The increase of sequencing data has allowed for in-depth sequencing of populations, generating hundreds of genomes for some species [23]. Interpretations of this data require comparisons using methods such as whole genome alignment. Extensive runtime or memory consumption limit the number of genomes that can be readily aligned simultaneously with whole genome alignment tools. Current implementations are designed to perform work in serial using only one CPU, requiring a day or more to align more than a few dozen bacterial chromosome [1,83].

In this Chapter, we describe preliminary results of new algorithm and tool, Para-Mugsy, for parallel and distributed whole genome multiple alignment. We demonstrate Para-Mugsy on 145 *E. coli* genomes (725Mbp) completing an alignment in ~38 hours using 8 CPUs on an Amazon EC2 virtual machine instance. Para-Mugsy is also space efficient requiring less than 16 GB RAM for the 145-way alignment. An examination of a subset of the output indicates the alignment is of comparable quality as generated by other tools.

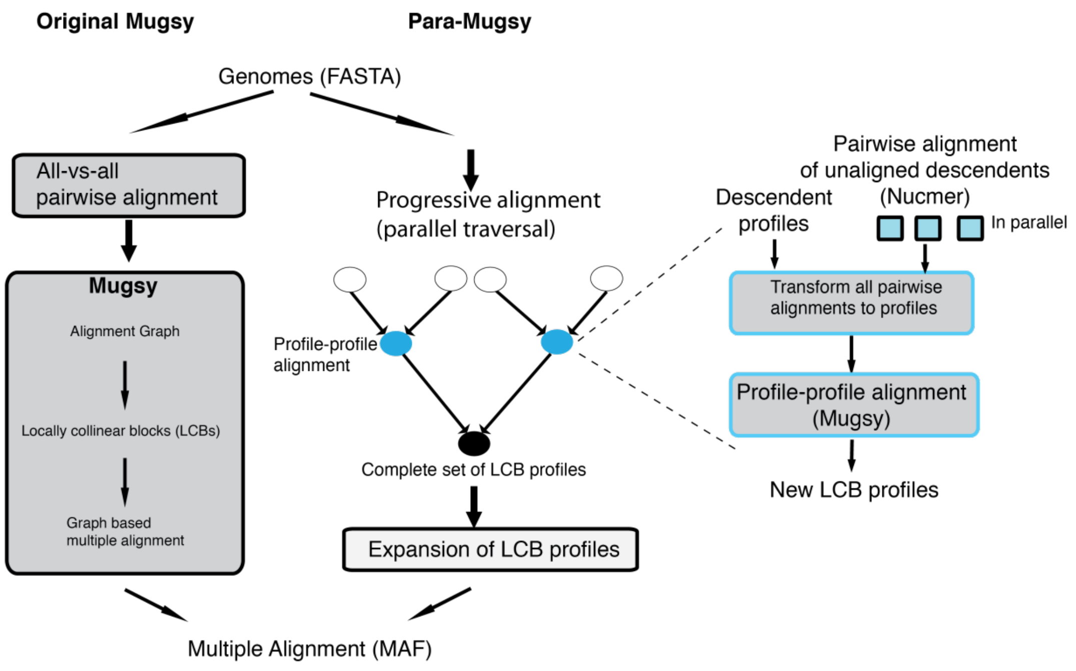

Original Mugsy
Para-Mugsy
Genomes (FASTA)
All-vs-all pairwise alignment
Mugsy
Alignment Graph
Locally collinear blocks (LCBs)
Graph based multiple alignment
Progressive alignment (parallel traversal)
Profile-profile alignment
Complete set of LCB profiles
Expansion of LCB profiles
Pairwise alignment of unaligned descendents (Nucmer)
Descendent profiles
In parallel
Transform all pairwise alignments to profiles
Profile-profile alignment (Mugsy)
New LCB profiles
Multiple Alignment (MAF)

Figure 5.1 – Overview of Para-Mugsy compared to the serial Mugsy algorithm for whole genome multiple alignment

Methodology

The method is based on progressive multiple alignment [44] using many of the same codes from Mugsy [1] with the addition that pairwise sequence alignments and profile-profile alignments are computed in parallel while traversing a phylogenetic guide tree (Figure 5.1).

Para-Mugsy reads a set of genomes in FASTA format and produces multiple alignments in Multiple Alignment Format (MAF file). Para-Mugsy executes on a CloVR virtual machine (Chapter 7) and all processing jobs are submitted to a GridEngine queue for scheduling over multiple CPUs. A description of each of the steps of the algorithm follows.

Calculation of a guide tree

To enable the progressive alignment, a rooted phylogenetic tree is estimated from the input sequences using Muscle [45]. Muscle (–treeonly) calculates the number of shared k-mers between sequences to build a distance matrix for a UMPGA tree building procedure [137]. Leaves in this guide tree correspond to genomes, which may be represented by multiple sequences, in case of draft genomes.

Progressive alignment

The phylogenetic guide tree is then used by Para-Mugsy to determine the order in which sequences and intermediate alignments represented by profiles are calculated and combined. The algorithm vists nodes of the tree from leaves to root, visiting children before parents. At each non-leaf node, two sets of alignments are computed

i) Pairwise sequence alignments. The alignment between all pairs of descendent genomes and their sequences using Nucmer [68].

ii) Profile-profile alignments. Alignments are computed using MugsyWGA producing locally collinear blocks (LCBs) . A profile representation of each LCB is saved at the tree node.

Independent subtrees are visited in parallel allowing for concurrent computation of alignments across multiple CPUs (Figure 5.1). A perfectly balanced guide tree provides for maximum speedup, although no effort is currently made to ensure the guide tree is balanced. Such an optimization is left as future work.

Pairwise alignment

Para-Mugsy uses Nucmer [68] for pairwise alignments between genomes. A total of $N*(N-1)$ pairwise comparisons for a set of N genomes is performed. Each pair of genome comparisons can be computed independently and in parallel. Draft genomes with multiple contigs are represented in multi-FASTA files and aligned to other genomes at once.

Pairwise alignments are needed during progressive alignment for descendent sequences. Rather than pre-compute all of the pairwise similarities, alignments are calculated during progressive alignment for pairs of descendent genomes. This allows for concurrent calculation of pairwise sequence alignment and independent profile-profile alignments across a cluster. A future possible optimization can short-circuit some of the later pairwise alignments during the guide tree progression by considering the composition of these intermediate alignments. Pairwise alignments between descendent sequences within a subtree are computed once and saved for subsequent access during tree traversal.

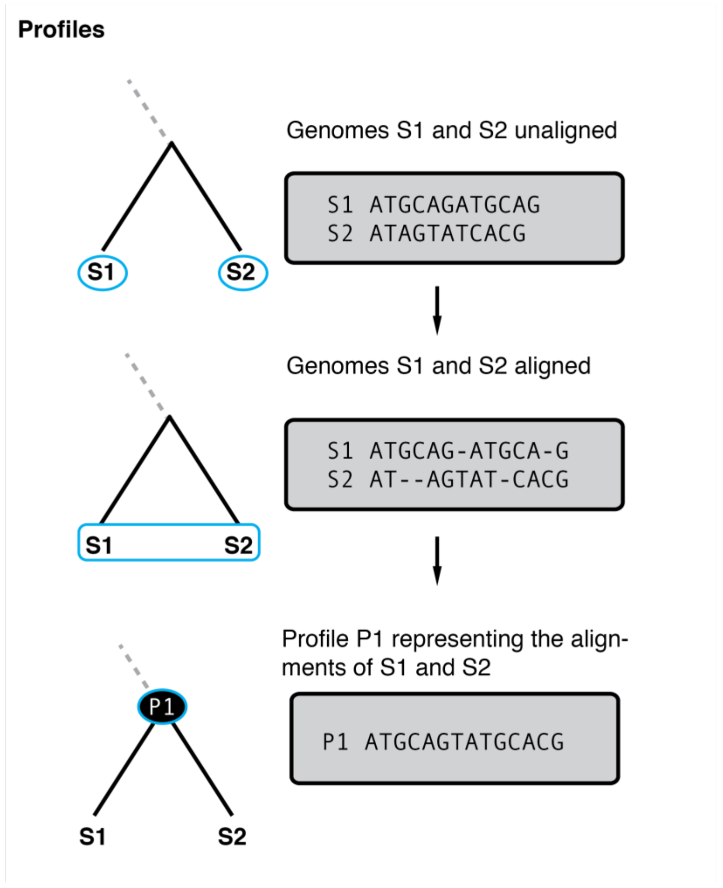

Profiles
Genomes S1 and S2 unaligned
S1 ATGCAGATGCAG
S2 ATAGTATCACG
S1
S2
Genomes S1 and S2 aligned
S1 ATGCAG-ATGCA-G
S2 AT--AGTAT-CACG
S1
S2
Profile P1 representing the alignments of S1 and S2
P1 ATGCAGTATGCACG
P1
S1
S2

Figure 5.2 – Generation of profiles

A profile sequence is the combination of all of these sequences where gaps are filled in a consensus character or an "N" the matching nucleotide in the other sequence. Any mismatches are represented with an "N" character. All of the profile sequences are put in a multi-FASTA file.

Profile-profile alignments

Pairwise alignments are combined into locally collinear blocks (LCBs) and multiple alignments using Mugsy. A profile representation of each LCB is generated and assigned to the internal node (LCB profiles in Figure 5.2). These profiles are used as an input sequences for subsequent invocations of Mugsy along the guide tree. LCB profiles are aligned to each other during progressive alignment to build new LCBs at each internal node of the tree using Mugsy. Each invocation of Mugsy requires two inputs:

1) FASTA sequences of the input genomes. For profile-profile alignments, these inputs are LCB profile sequences.

2) Library of pairwise homology matches between sequences.

A mapping between the profile sequence positions and underlying pairwise alignments on the constituent sequences allows for construction of a pairwise library between any two profiles. This mapping is stored on disk as a list of interval pairs between each profile segment and a contiguous range on the original sequences.

Since profiles are sequence strings, this strategy allows for use of the original Mugsy executables described in Chapter 4 without modification for profile-profile alignments. No comparison of the actual profile strings is required. Rather, pairwise alignments generated by Nucmer on the original sequences are transformed to the corresponding coordinates on the profile sequences (Transform in Figure 5.1). This allows use of profile sequence in place of multiple sequences throughout the algorithm while representing all available homology information.

Because profiles represent multiple sequences, redundant matches are expected in the pairwise alignment library for profile-profile alignments. Mugsy utilizes a segment graph for determining locally collinear blocks (LCBs) that can be aligned. This segment graph is built from a set of pairwise alignments using the combination and refinement procedure such that each segment is non-overlapping and all pairwise matching information is preserved [87].

The calculations of intermediate alignments are independent for siblings in the guide tree and can be aligned in parallel (Figure 5.1). To avoid construction of numerous, relatively small intermediate alignments and profiles in very deep trees, intermediate alignments are not calculated for internal nodes until a threshold of the sum of the descendent sequence lengths is reached. The default threshold is 20MB equaling roughly 5-10 bacterial chromosomes. An initial set of profile LCBs are calculated for these genomes and are then combined in pairwise fashion during tree traversal.

Progressive alignment of profiles is continued along the guide tree until the root of the tree is reached at which point the profiles represent LCBs across all input genomes. A translation step converts these final set of profiles into a multiple alignment on the original sequences. The result is a single file in MAF format that contains multiple alignments for each LCB.

Table 5-1 Para-Mugsy performance

	CPUs	**Runtime (min)**	Max RAM	No. LCBs	Total aln (bp)	Core aln (bp)
Original Mugsy 57 *E. coli*	1	**1,155**	~12GB	11,881	289,057,323 (97%)	2,946,752
Para-Mugsy 57 *E. coli*	8	**482**	1.4GB	12,409	261,012,599 (87%)	2,938,034
Para-Mugsy 145 *E. coli*	8	**2,282**	6.0GB	24,944	627,955,536 (84%)	2,441,541

The genomes are 57 genomes of *Escherichia coli* are from Supplemental Table 1 in [1]. The dataset of 145 genomes adds 89 unpublished draft genomes.

<u>Results</u>

We generated whole genome multiple alignments of up to 145 *E. coli* genomes using Para-Mugsy. The 145-way alignments were generated in ~38 hours using 8 CPUs and no more than 6GB of RAM of a c1.2xlarge virtual machine instance at Amazon EC2. Comparable tools that run on a single CPU are unable to complete an alignment of a subset of 57 genomes in less than 23 hours of processing and 8GB RAM [1,83].

To assess quality, we compared a 57-way alignment produced by Mugsy with that calculated by Para-Mugsy (Table 5-1). Para-Mugsy calculated the aligned core that includes all input genomes calculated comprising 2,938,034 nucleotides compared to 2,946,752 nucleotides calculated by original Mugsy algorithm. The number of aligned core nucleotides calculated by Para-Mugsy decreased to 2,441,541 in the 145-way comparison. In non-core regions, Para-Mugsy demonstrates decreased sensitivity compared to the original Mugsy algorithm; 87% of all input nucleotides covered in Para-Mugsy versus 97% in Mugsy. The 145-way alignment covered 84% of the total base input pairs.

To further assess quality, ten *E. coli* genomes were selected and aligned with both the original Mugsy algorithm and the distributed Para-Mugsy tool. Using Mugsy as the hypothetical truth, Para-Mugsy displayed a specificity of 97.7 and a sensitivity of 99.2 on this dataset.

Discussion

With the increasing availability of multi-core computers and cloud computing, distributed computing solutions are readily available to improve processing throughput. By using distributed computation, Para-Mugsy increases the number of genomes that are feasible for whole genome comparison. Para-Mugsy enabled alignment of 145 *E. coli* genomes. An alignment of 57 *E. coli* genomes with the original serial Mugsy algorithm requires almost one day to compute on single CPU and at least twice as long (> 2 days) with Mauve.

In addition to computational time, memory consumption is a limiting factor in whole genome multiple alignment as large memory requirements can exclude computation entirely. Memory is also relatively expensive currently compared to other component costs of a computer. The peak memory consumption of Para-Mugsy is significantly less than peak memory consumption of Mugsy on the same input data (~1.4GB vs. 12GB for a 57-way alignment of *E. coli*).

The graph based alignment strategy utilized by Mugsy is inefficient in comparisons of diverse sequences. A key data structure utilized in Mugsy and Seqan::TCoffee is an alignment graph, which stores a vertex for each ungapped aligned segment of a genome. The number of vertices the alignment graph is a function of both the number of genomes and number of gaps in the pairwise alignments, while the number edges is proportional to the square of the number of vertices. As a result, the size of the graph grows quickly, requiring large amounts of RAM, in comparisons of many genomes that are very diverse; many gaps, and genomes lead to many vertices in the graph.

Para-Mugsy utilizes profile sequences to represent intermediate alignments of each LCB calculated along the guide tree. The use of profiles by Para-Mugsy avoids this exploding memory requirement by reducing the number of genomes under comparison since the profile-profile alignment only appears as two genomes in the segment graph, regardless of how many genomes are represented by the profile. This reduction in memory comes at the expense of additional computation, as alignments are recomputed during guide tree traversal. This increased computation is compensated for by use of multiple CPUs.

The current implementation of the algorithm performs at reduced sensitivity for non-core regions that are conserved in subsets of inputs. Para-Mugsy also produces more LCBs and a more fragmented view of genome conservation than either Mauve or Mugsy. This fragmentation is due to the use of profiles, which appear to the algorithm as separate sequences during the progressive alignment. As a result, LCBs calculated at each intermediate step can only be combined with existing boundaries or split further, but are never extended. Additional work is needed to address this limitation, such as additional chaining steps during the progression that merge collinear profile sequences. The use of a guide tree helps alleviate the data fragmentation since the most similar sequences are combined first so that LCBs start in the progression with the longest spans. The work of Para-Mugsy enables computation of a multiple alignment of the conserved genetic core across more genomes than is currently feasible using other methods.

Chapter 6 - Rapid Comparison and Annotation of Pan-genomes

Sequencing of whole genomes from populations (pan-genomes) is becoming commonplace to study functional capabilities of a species [5,6,29] and identify targeted therapeutics [138,139]. To aid with analysis of populations of genomes, efficient methods are needed for comparison and annotation of genes across a pan-genome.

Exploration of cellular functions and metabolic pathways is often reliant on comparison of genes and gene families between and within species to those that a share a common ancestry with well-studied and experimentally verified homologs. As the number of sequenced genomes within a single species continues to grow, tools are needed for efficiently identifying genes that share a common ancestry allowing for rapid interrogation and annotation of genes across a population.

To accommodate high-throughput genome sequencing, exclusively automated methods are used for gene prediction and functional annotations [99]. The resulting genome annotations can contain inconsistencies and errors that hinder comparative analysis even within a single species [54].

In this Chapter, we introduce a novel method that 1) identifies orthologs, i.e.genes descended from the same ancestral sequence, and 2) evaluates annotation quality across a pan-genome using whole genome multiple alignment. The methodology, implemented as the tool Mugsy-Annotator, uses conserved genomic position, i.e. synteny, to aid in identifying orthologs. This provides the foundation for comparing gene structures to identify annotation anomalies, including inconsistently annotated translation initiation sites (TIS), missing annotations, and disrupted genes due to sequencing and assembly errors, or pseudogenes, including frameshifted genes. Finally, Mugsy-Annotator identifies alternative annotations that resolve anomalies and improve annotation consistency.

We evaluated the tool on annotations across a number of bacterial pan-genomes demonstrating that annotation anomalies are common, especially at translation initiation sites. We also utilized the tool for improving annotations on a set of twenty genomes of the environmentally prevalent and occasionally pathogenic bacterium *Neisseria meningitidis* [140]. The tool is freely available at http://mugsy.sf.net.

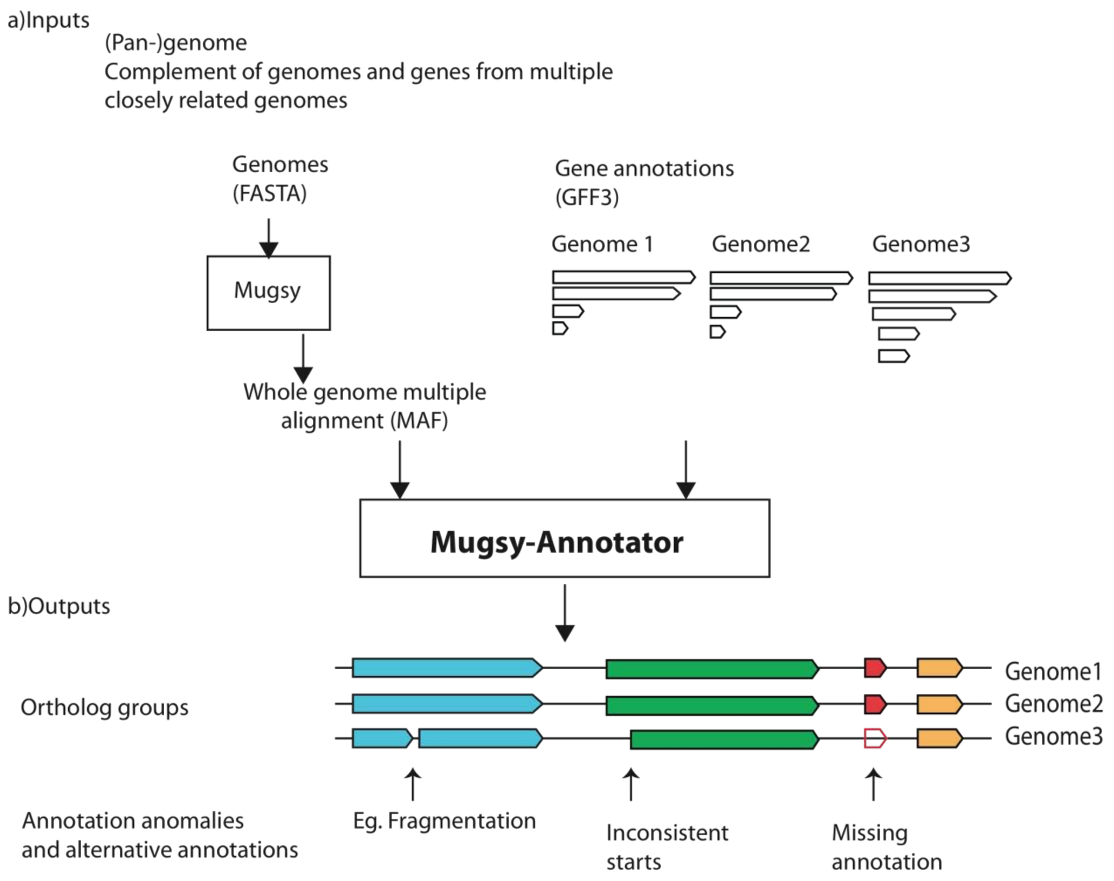


Figure 6.1 - Identifying orthologs and comparing gene structures in a pan-genome using whole genome multiple alignments

The input is provided as a set of genomic sequences (FASTA format) and gene annotations (GFF3 format). Whole genome multiple alignments (top left) are first calculated using Mugsy (Angiuoli and Salzberg 2011). Mugsy-Annotator then builds groups of orthologous gene structures that are conserved in sequence and genomic context according to the alignment. The alignment also indicates the location of each predicted translation initiation start and stop across the genomes, allowing for identification of annotation anomalies or missing annotations.

Methods

The method consists of three primary steps, (1) aligning multiple whole genomes, (2) mapping orthologs among the genomes, and (3) identifying annotation anomalies (Figure 6.1). Two types of input files are required: genome sequences in FASTA format, and annotated gene structures in Genbank or GFF3 format. It is expected that a gene prediction algorithm has been run on all input genomes. For step 1, we generate reference-independent whole genome multiple alignments using Mugsy [1]. The alignments generated by Mugsy are restricted to a single region per genome and used by Mugsy-Annotator to define orthologous relationships between sequences. Mugsy outputs alignments in Multiple Alignment Format (MAF) that are passed to Mugsy-Annotator along with the genome annotations needed to complete steps 2 and 3. The genomic coordinates and alignment string of each aligned interval are extracted from the MAF files and stored in an interval tree [141] to provide fast querying of genomic intervals. The start and end coordinates of each gene are also extracted as intervals from the annotation files and stored in the interval tree. The interval tree is then queried by Mugsy-Annotator to build groups of orthologs and identify anomalies in gene boundaries. Although we utilize Mugsy for whole genome multiple alignment, Mugsy-Annotator accepts MAF files as input and other whole genome alignments tools can be used instead of Mugsy as long as the input is properly formatted.

Identification of orthologs

Sets of orthologs are determined by retrieving genes whose intervals are aligned via whole genome alignment (WGA). First, the input genes are sorted on length. The longest gene remaining in the input set, termed the query gene, is removed from the input and used to define a new ortholog group. Genes from other species that align to the query gene in the WGA are added to the ortholog group and removed from the input set. This ensures genes are placed in exactly one group. A configurable coverage cutoff can limit consideration to alignments that span a minimum percentage of the query gene and other matching genes. In this study, we set these length cutoffs to 50%. The procedure continues in a greedy fashion using the longest remaining gene to seed new groups (or clusters) until no genes are remaining. Query genes with no overlapping genes above the cutoffs are reported as singleton groups. Using this method, the query gene in each ortholog group is at least as long as any other gene in the cluster and may span multiple adjacent genes in other genomes. This allows our method to identify apparent fragmented genes within a single region.

To generate OrthoMCL clusters for comparison [142], we performed an all-against-all BLAST searches of conceptual translations of the gene predictions. BLAST alignments with e-value < 10^{-5} were used as input to OrthoMCL v1.4 to predict groups of orthologs.

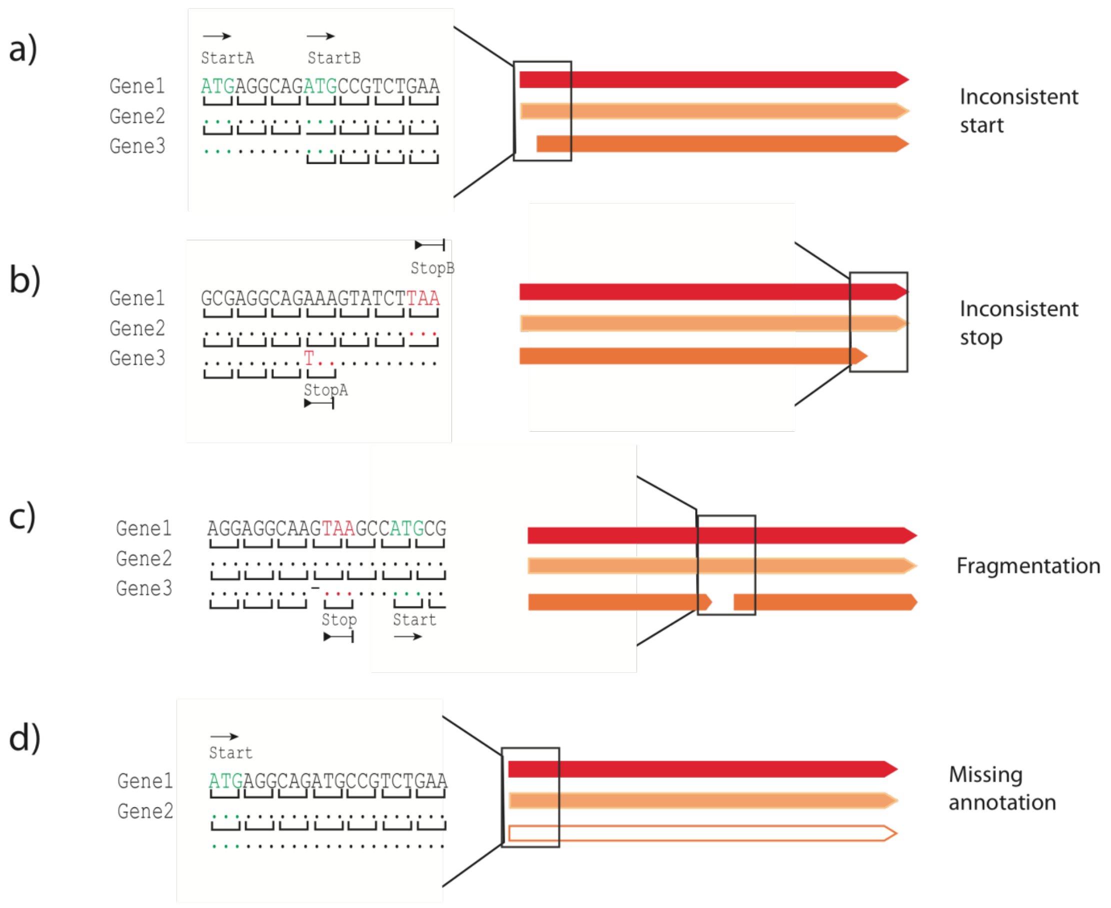


Figure 6.2 – Annotation anomalies identified by Mugsy-Annotator
Four classes of anomalies are shown (a-d). On the right, examples of aligned genes are drawn with the boxed region indicating the location of the anomaly. On the left, a multiple alignment is depicted across the highlighted region with sequence identity indicated by dots. In (c), a gap indicated by a dash introduces a shift in reading frame that results in use of a termination codon that is inconsistent with the annotations in the other genomes. Translation initiation sites are marked as "start" and termination codons are marked as "stop" with an arrow indicating the direction of translation.

Identification of annotation inconsistencies

Mugsy-Annotator produces a report of the annotation consistency for each ortholog set. To classify annotation consistency for each ortholog set, we examine the location of the annotated start and stop codons for each gene in the multiple alignment. If all annotated start and stop codons are in the same location, the ortholog set is consistently annotated and we identify no inconsistencies. Otherwise, we classify the ortholog set into one or more classes: inconsistent starts, inconsistent stops, and multiple gene fragments. If the stop codon locations are the same for all annotated genes but the translation initiation sites (TIS) differ, we classify the set as inconsistent starts (Figure 6.2a). If the start codon locations are the same for all genes but the stop codons locations differ, we classify the set as inconsistent stops (Figure 6.2b). If both start and end locations differ for some members of the group, we classify the group as a combination class. This class will include genes that overlap in the alignment but in different reading frames or strands. Aligned gene sets with multiple annotated genes in the same genome are classified as multiple gene fragments (Figure 6.2c).

Alternative annotations

Mugsy-Annotator suggests edits that can resolve anomalies and improve the consistency of each aligned gene set. To determine the possible edits, start and stop codons pairs from each aligned set are checked against the WGA to determine if the aligned positions correspond to ORFs with a valid translation start and stop site (NCBI translation table 11) in each of the other aligned genomes. In cases where the region already contains gene predictions, only alternatives that are greater than a specified percentage (50% by default) of the annotated length are considered.

The procedure will also identify aligned gene sets with multiple gene fragments that can be merged into a single spanning gene by introducing a point mutation or frameshift into the annotation. If the aligned regions contain gaps, Mugsy-Annotator attempts to introduce a frameshift to create a valid ORF joining the start and stop codon pair. Start and stop codon pairs are then displayed ordered by the number of valid ORFs and their length, although this sort order is configurable. This procedure will also identify possible missing genes in regions of the genome that are aligned to other annotated genes (Figure 6.2d). To be considered a missing annotation, there must be no overlapping gene predictions in the aligned interval.

Data sets

The *Nmen* dataset of 20 genomes was the same as used in [140]. Two versions of the annotation were available, *Nmen* verA and *Nmen* verB. *Nmen* verA contained 13 genomes that had been annotated using one of two automated pipelines prior to any manual review. Unless noted, the annotation anomalies identified in this study used the *Nmen* verB annotations, which had undergone limited manual review. The remaining species pan-genomes used in this study were downloaded from the Refseq database [129]. MUMi [82] distance measurements were calculated for each pair of sequences with a named species.

Results

Mugsy-Annotator for finding orthologs

Mugsy-Annotator uses whole genome alignment (WGA) calculated by Mugsy [1] to identify conserved genes in a set of genomes (Figure 4.1). In cases where the alignment represents orthologous regions, these aligned genes correspond to orthologs; i.e., genes descended from the same ancestral sequence. WGA aids in distinguishing orthologs from paralogs by identifying regions that are syntenic and conserved in both sequence and chromosomal position. By aligning genomic DNA, WGA can also identify erroneous gene predictions in a reading frame that produce a nonsense translation and escapes detection by similarity methods that rely on conceptual translations, such as BLASTP. On the other hand, by relying on DNA alignment, Mugsy-Annotator might miss sequence conservation between genes that is only detectable at the protein level.

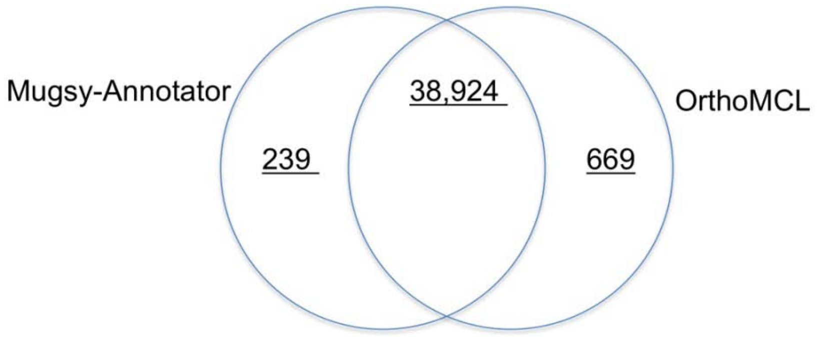


Figure 6.3 – Comparison of ortholog groups between Mugsy-Annotator and OrthoMCL
The intersection between Mugsy-Annotator and OrthoMCL reports the number of genes reported in ortholog groups by both methods. The remainder for Mugsy-Annotator and OrthoMCL reports the number of genes classified in ortholog groups by one of the methods only.

As a case study to evaluate WGA for ortholog identification, we compared the groups of orthologs reported by Mugsy-Annotator and OrthoMCL for 20 *Neisseria meningitidis (Nmen)* genomes [140]. OrthoMCL performs a clustering of Reciprocal Best BLAST (RBB) matches between conceptual translations of genes to identify orthologs. In *Nmen*, Mugsy-Annotator identified 2,440 ortholog groups compared to 2,320 reported by OrthoMCL. The Mugsy-Annotator groups include nearly all the genes included in RBB matches used by OrthoMCL (38,905 of 39,593 total, 98%).

Both methods reported genes missing from groups reported by the other method, totaling 239 and 669 genes reported by Mugsy-Annotator and OrthoMCL exclusively (Figure 6.3). Many of the genes reported exclusively by one method appear to be paralogs based on intra-genome BLAST matches (40% and 66% reported exclusively by Mugsy-

Annotator and OrthoMCL methods respectively) or have functional names that indicate transposases (33% and 23% for WGA and RBB respectively) or hypothetical proteins (34% and 31% for Mugsy-Annotator and OrthoMCL respectively).

Clustering of RBB matches can collapse orthologs and paralogs into a single group. Identifying orthologs separately is needed for phylogenetic analysis of gene families that rely on orthologs , comparison of upstream regulatory regions, and examination of segmental duplications, where each duplicated copy has a distinct genomic context. As described in Methods, our tool, by utilizing WGA, incorporates genome context and synteny in determining matches and builds groups that are restricted to a single gene copy per genome, thus avoiding the grouping of orthologs and paralogs together. In the case of segmental duplications, Mugsy-Annotator will report separate groups for each copy. In the *Nmen* comparison, OrthoMCL reports 310 groups with multiple genes per genome that align to each other via BLAST, indicating paralogs in a single group. Mugsy-Annotator will sometimes report groups with more than one gene per genome (Figure 6.2c, "Fragmentation"), but rather than paralogs, these groups represent fragmented genes due to draft genome sequencing (gaps or sequencing errors) or potential pseudogenes.

For genes grouped exclusively by Mugsy-Annotator, 23 have no reported intra-species BLAST matches to other genes in *Nmen*, and include annotations that appear to be in an incorrect open reading frame (Table S2). Although we found this class of anomaly to be rare in our evaluation, Mugsy-Annotator, by using WGA, is able to identify orthologs to such regions that lack BLAST matches within the dataset and may have a nonsense conceptual translation. An additional 68 genes (28%) reported exclusively by Mugsy-

Annotator are adjacent to contig boundaries and may be truncated gene predictions that escape detection by BLAST.

Our WGA method is computationally efficient and has a significant runtime performance advantage over BLAST. The comparison of 20 *Nmen* genomes runs on a single CPU in ~4 h (~2 h for WGA with Mugsy and ~2 h for comparing annotations with Mugsy-Annotator). By comparison, the exhaustive all-against-all BLAST of predicted proteins needed for OrthoMCL consumed ~32 CPU hours and was run on a compute cluster to obtain a faster runtime. In addition, BLAST-based methods that rely on searches of conceptual translation may require additional search of the genomic DNA, such as with BLASTx, to confirm gene presence and avoid mis-prediction of paralogs as orthologs.

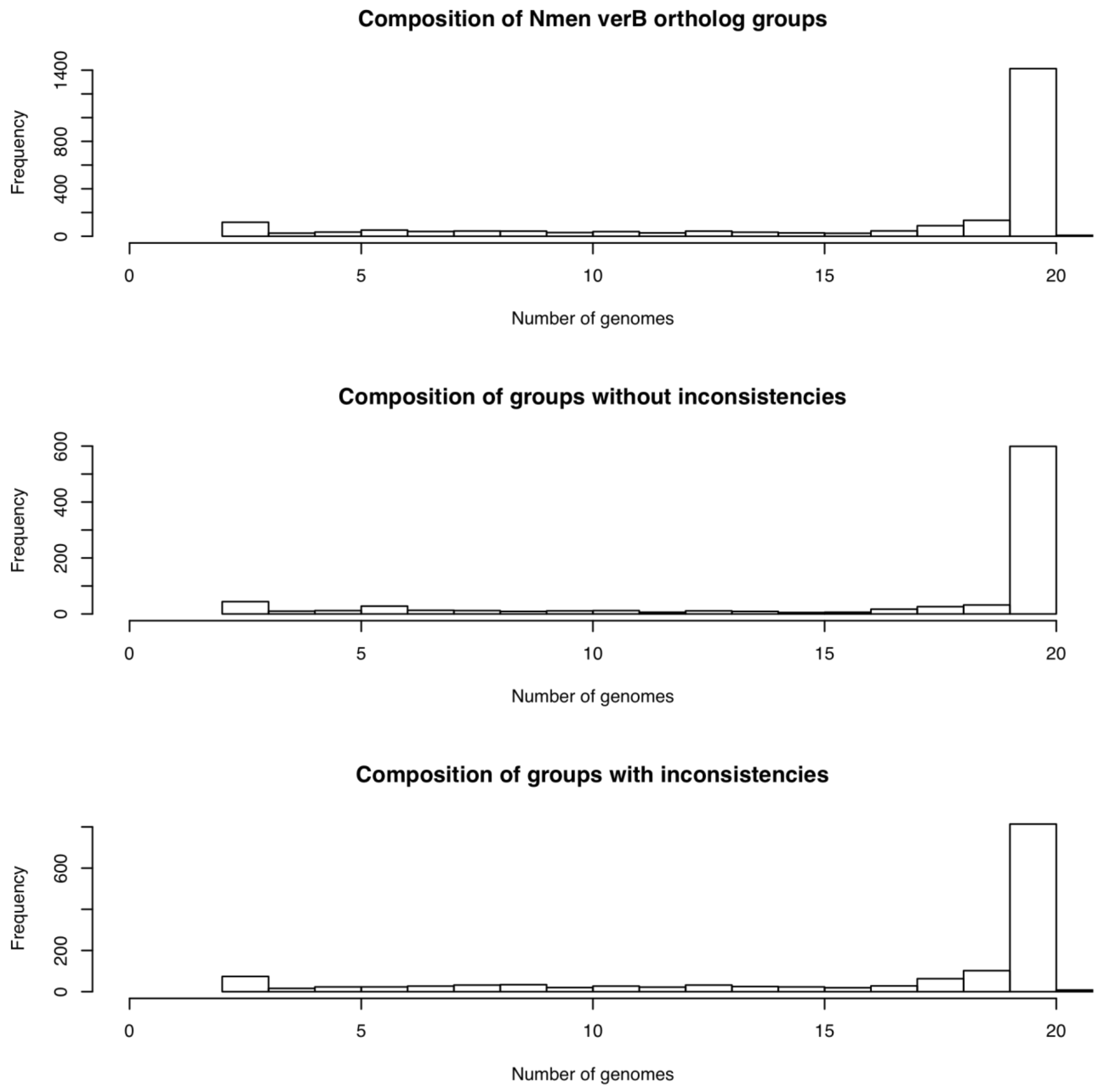


Figure 6.4 – Distribution of the number of genomes in ortholog groups

The number of genomes per orthologs are provided for all orthology groups (top), consistently annotated groups only (middle), and exclusively groups with annotation inconsistencies (bottom) as identified by Mugsy-Annoator for 20 *Nmen* genomes..

Missing annotations

Mugsy-Annotator can be used to identify missing annotations and putative genes by looking for regions of the alignment with a prediction in some genomes but not others (Figure 6.2d, “Missing annotation”). These missing annotations can arise from use of varying gene prediction tools and uncertainty in gene calling procedures, especially for short genes [143]. In our study of 20 *Nmen* strains, a majority of the aligned gene sets contain one annotated region from each of the genomes (Figure 6.4) and missing gene predictions were rare, totaling only 50 genes missing in alignments containing 18 or more genomes (Table S3).

Mugsy-Annotator identifies missing annotations if DNA corresponding to a putative gene is an open reading frame that is conserved across genomes. However, it does not provide additional evidence to determine if a gene prediction is missing in some genomes (false negative) or there is an overcall in the other aligned genomes (false positive). Our methodology relies on sequence conservation between the input genomes, which by itself is insufficient to distinguish between these due to the short phylogenetic distance and high similarity of the genomes. Examination of additional evidence (eg. HMM or BLAST searches) or experimental validation is required to differentiate between these cases.

Identifying and resolving annotation anomalies

To aid in re-annotation efforts, Mugsy-Annotator identifies likely annotation problem areas and suggests alternative genes based on the whole genome multiple alignment. To find such problem areas, Mugsy-Annotator first examines each of the aligned gene sets for inconsistencies in annotated gene boundaries amongst members of the set (Figure 6.2). The reported anomalies include inconsistently located TIS, disrupted genes, or alternative open reading frames. Mugsy-Annotator then generates a report for each aligned gene set that describes the inconsistency and possible resolutions. A HTML report of the annotations overlaid on the whole genome multiple alignment is also provided.

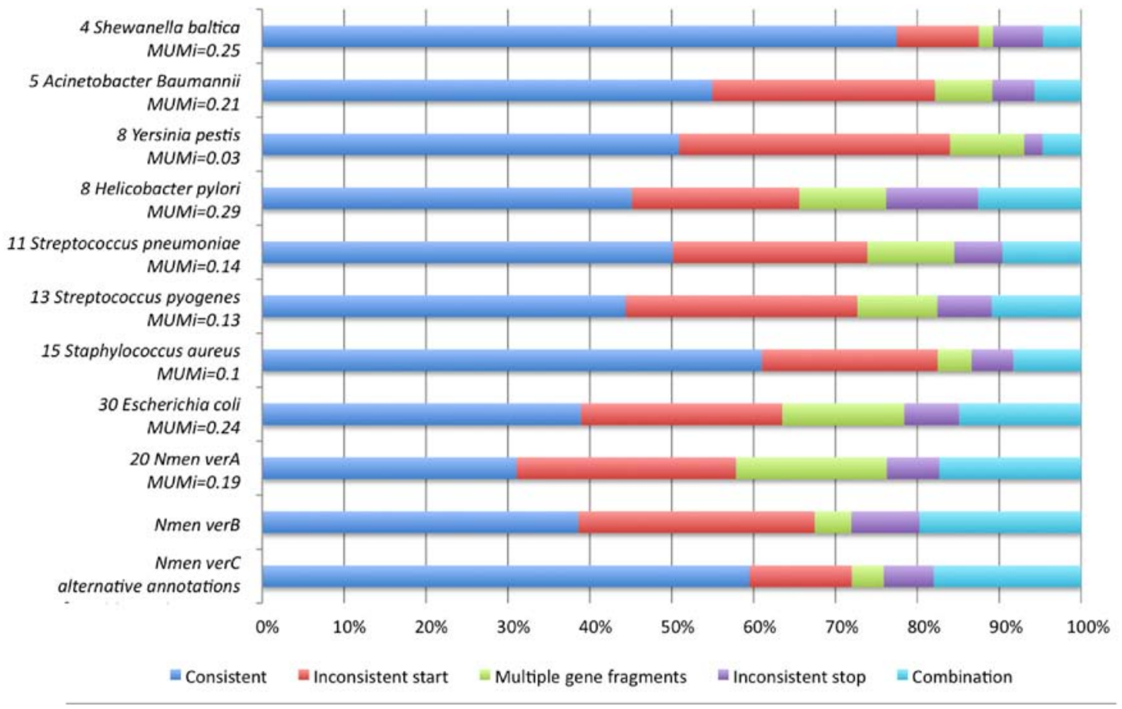


Figure 6.5 – Consistency of annotated gene structures in several species pan-genomes

Each row provides the fraction of aligned gene sets in each class of anomaly and groups with no identified inconsistencies (blue) as identified by Mugsy-Annotator. The number of genomes compared and their average MUMi similarity (Deloger et al. 2009) distance is also provided, ranging from zero for most similar to 1, least similar. The bottom three rows describe three versions of annotations from the case study of *Neisseria meningitidis (Nmen).* The last version (*Nmen* verC) demonstrates improvements in consistency using alternative annotations suggested by Mugsy-Annotator.

To demonstrate the tool, we ran Mugsy-Annotator on nine bacterial species, all of which have multiple strains with complete genomes available (Figure 6.5). The output indicates many inconsistencies in annotated gene structures, with inconsistent TIS locations as the most commonly identified anomaly. While the inconsistencies may indicate errors in the annotated gene structures in one or more of the genomes, the results are not surprising as the sequencing coverage, date of annotation, and annotation protocols vary. The presence of annotation errors in public repositories has been widely recognized [144,145,146] leading to a number of re-annotation efforts for genomes in a single species [147,148].

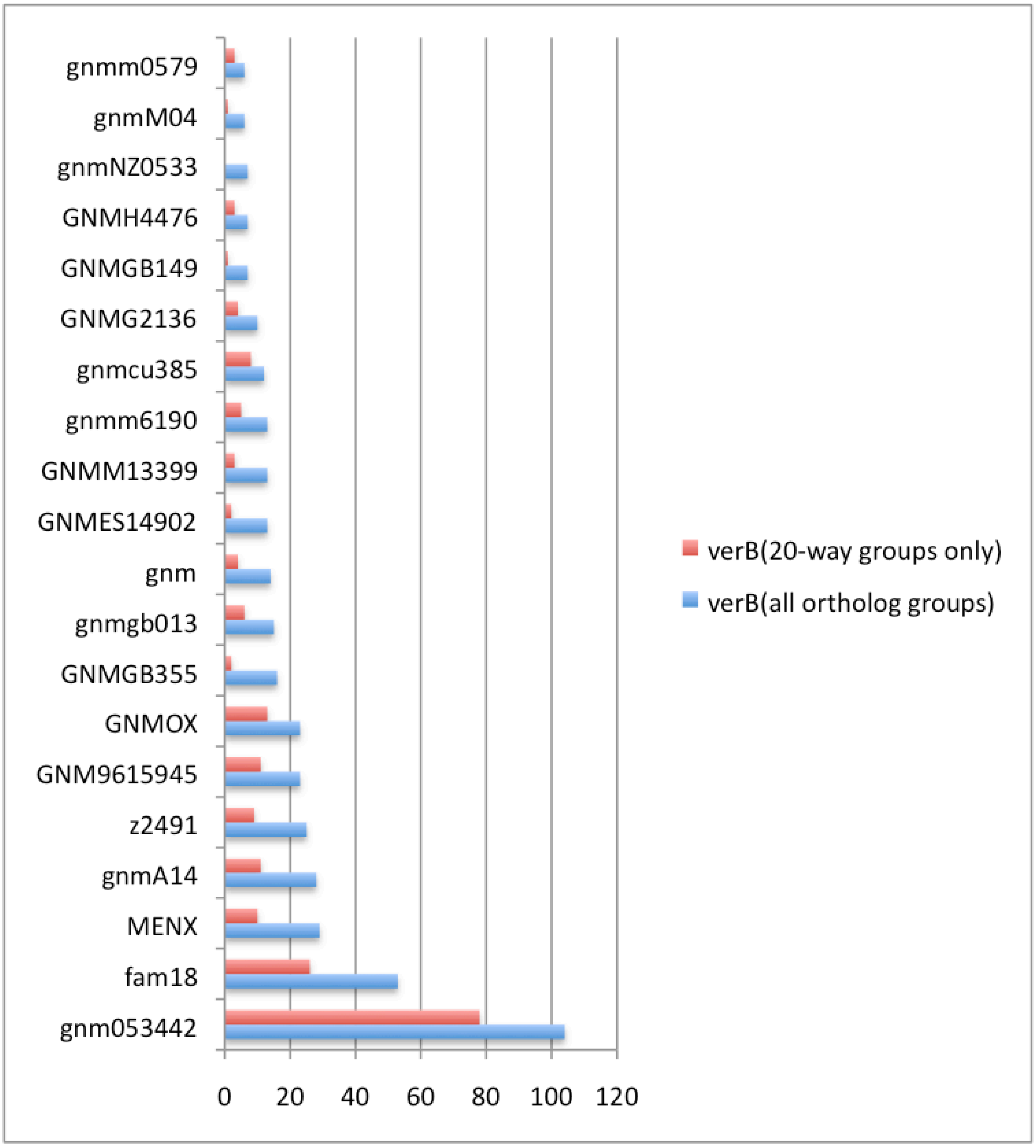


Figure 6.6 – Annotation anomalies caused by single genomes
Number of instances where an annotated translation initiation site in a single genome in *Nmen* verB did not match any of the remaining annotated gene structures the aligned ortholog groups.

As a case study, we evaluated the Mugsy-Annotator report for the dataset of 20 *Nmen* genomes. Inconsistent TIS are the most commonly detected anomaly in *Nmen* with 30% of aligned gene sets containing more than one annotated TIS. Due to lack of precision in TIS prediction, we expect the number of TIS inconsistencies to increase as the number of genomes increases, especially since our method marks a group as inconsistent even if the annotation error is limited to a single genome. To see how overall consistency is affected by any single genome, Mugsy-Annotator reports the number of times a single genome is inconsistent in comparison to the set. An examination of the *Nmen* genomes shows that certain subsets of genomes have better internal consistency. In 27% of groups with TIS inconsistencies, an alternative annotation in a single genome will resolve the inconsistencies for the group (Figure 6.6). Although some of the *Nmen* genomes contributed to more annotation inconsistencies than others, all of the genomes contributed to inconsistencies in at least one group.

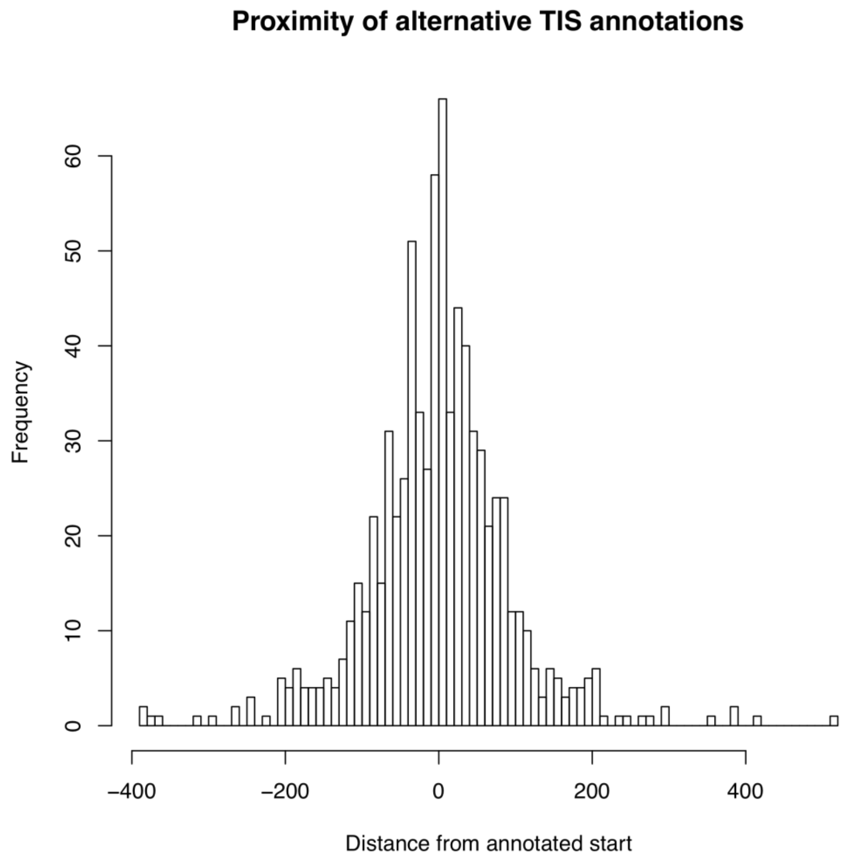


Figure 6.7 – Distance of alternative TIS from the annotated site
Distance between the annotated translation initiation site and the most consistent translation initiation site reported by Mugsy-Annotator.

Mugsy-Annotator suggests alternative gene structures that improve annotation consistency. In *Nmen* core gene groups containing all genomes, 55% (400/725) of groups with inconsistent TIS can be resolved by an alternative annotation that is conserved across all the aligned genomes. In 50% of these cases, the alternative start site is upstream of the existing annotation, resulting in longer annotations. In the remaining cases, the most consistent TIS location results in a shorter gene in at least one genome. A majority of the alternative TIS locations are in the same coding frame and within 42 bp of the annotated TIS (Figure 6.7), indicating that annotation protocols have chosen inconsistently between sites that are nearby along the genome. Adjusting the TIS can result in an overlap with an adjacent gene. To help avoid mis-annotation of overlapping genes [149], Mugsy-Annotator flags edits that would result in an overlap with an adjacent gene. In alternative annotations of *Nmen* groups, 15% (63/400) introduce overlap with adjacent annotations indicating further evaluation is needed to determine the correct annotation.

When a single gene in one genome is aligned to multiple genes in other genomes, Mugsy-Annotator calls this an anomaly (Figure 6.2c, "Fragmentation"). These apparent gene fragments can arise from sequencing and assembly errors; from interesting novel gene fusions; or from pseudogenes, in which frameshifts or in-frame stop codons can split an open reading frame into multiple gene-like fragments. In our case study of *Nmen*, draft genome sequencing appears to contribute to a vast majority of occurrences of this anomaly (Table S4), although the tool has also aided in the identification of several novel gene fusions that are not fixed in the population. To aid in classifying this anomaly further, Mugsy-Annotator reports whether or not frameshifts can extend the interrupted gene fragments to match a longer annotated gene. Amongst the aligned gene sets containing all 20 *Nmen* genomes, Mugsy-Annotator found 48 cases where a single previously un-annotated frameshift would resolve the anomaly and result in a consistently annotated set (Table S5). In many other cases, some of the genes can be extended with a frameshift but other anomalies remain in the group. Additional review would be needed to further classify these anomalies.

In the *Nmen* study, Mugsy-Annotator suggests alternative annotations that can improve consistency in up to 57% of ortholog groups. Although the alternatives improve consistency, in most cases an examination of additional evidence is required to ensure that edits improve quality. In this case study, the variability of the annotation is partly due to the multitude of sources and sequencing strategies. The *Nmen* genomes are in varying stages of completeness genomes include 9 draft and 11 complete genomes and the annotation evaluated came from a total of 5 laboratories using varying gene prediction protocols and levels of manual curation. To better accommodate draft genomes, the gene prediction procedure used in some of the *Nmen* genomes allows for partial open reading frames that terminate or initiate outside of a contig boundary. Mugsy-Annotator flags anomalies that are caused by these partial genes adjacent to contig boundaries. In *Nmen,* such cases contributed to ~9% of start and stop site inconsistencies and at least 67% of all of the multiple gene fragment anomalies (Table S4). Annotation anomalies due to draft genome assemblies will continue to be an issue in multi-genome analysis as current generation sequencing technologies have prompted an explosion in the number of draft genomes.

To demonstrate annotation improvements using Mugsy-Annotator, we scored annotation consistency in three versions of annotation for *Nmen.* An initial version of the *Nmen* annotation (*Nmen* verA), contained predominantly automated annotation in 13 newly sequenced genomes and curated annotation for 7 complete genomes. *Nmen* verA and showed a large number of inconsistencies, encompassing 72% of orthology groups (Figure 6.5). As part of the study in [140], limited manual curation was performed and resulted in annotation of frameshifts and removal of many short, unsupported hypothetical gene predictions and resulted in the annotations in *Nmen* verB. Although this manual effort was aided by the Mugsy-Annotator report, the curation effort was not meant to be exhaustive and not all reported inconsistencies were examined during the review. Subsequent to this manual effort, Mugsy-Annotator was run again and generated a new set of alternative annotations (*Nmen* verC*)* suggesting additional improvements were possible. This resulted in consistent annotations in 59% of groups in *Nmen* verC, which was an increase from 28% in *Nmen* verA The improvement in annotation consistency between versions highlights the need for re-annotation and manual review subsequent to automated annotation.

Discussion

With numerous individual genomes for many bacterial species, there is an increasing need for tools that compare the genomes. Mugsy-Annotator by using whole genome multiple alignment can be used to efficiently identify orthologs and annotation problem areas in a bacterial pan-genome.

Mugsy-Annotator implements method that is independent of a reference genome. For draft genome projects, Mugsy-Annotator identifies anomalies that are due to draft genome sequencing, such as inconsistently located translation initiation sites and disrupted genes. For re-annotation efforts, Mugsy-Annotator can be used to direct curators to likely errors and highlight alternative gene structures that are consistent across a population, enabling re-annotation across many genomes simultaneously, rather than one genome at a time. Mugsy-Annotator is currently limited to comparisons of genes that do not contain spliced gene structures.

As our comparisons within a species demonstrated more variation between annotations than is actually present in the genome, researchers should be careful when relying on gene structures for downstream applications. Mugsy-Annotator looks for inconsistencies in gene structures to identify likely errors but it is also possible that consistency results from the propagation of error, especially since it is common to use reference genomes when annotating new genomes. In some cases, the annotated gene structures may be consistent but incorrect and Mugsy-Annotator will not identify any anomaly. On the other hand, due to the short evolutionary distance between the genomes under evaluation in our case study, inharmonious gene boundaries in orthologs are expected to indicate an improper gene boundary assignment in at least one genome.

One area of future work is extending Mugsy-Annotator to build a fully automated pan-genome annotation system. Such a system would utilize the comparative genomics data for identifying genes and gene structures, regulatory elements, and prediction and assignment of gene functions, consistently across a species or higher-level clade. One option for an implementation of this would include integration of sequence conservation into an existing *de-novo* gene finder with whole genome alignment providing additional evidence supporting the annotation, especially if a well-chosen out-group sequence is provided. Comparative gene finders have been used extensively in eukaryotic annotation [150,151]. Similarly, a mapping approach for mapping between two genomes [98,152], could be extended across multiple genomes and used to augment existing gene predictions and transfer names and functional annotations across the new genomes. Since gene prediction runs quickly on bacteria (usually minutes), we expect the speed of such an approach would be limited by the time required to calculate a whole genome multiple alignment.

Importantly, additional evidence besides the whole genome alignment will often be needed to determine the correctness of the annotations including, but not limited to, gene boundaries of more distantly related orthologs, third position compositional bias, predicted ribosomal binding sites, and predicted signal peptides. As such, our tool stops short of determining the correctness of any gene calls, as this is best left to follow-up analysis or experimentation in the laboratory. Yet, our tool is ideally suited to direct the annotation curator towards the regions in most need of attention, and where Mugsy-Annotator suggestions will greatly facilitate rapid improvement of annotation consistency. Such tools are urgently needed in light of the explosion of genomes currently happening as researchers are sequencing hundreds of genomes for many individual species.

Chapter 7 - CloVR: A portable system for automated and distributed analysis in bioinformatics

High-throughput sequencing technologies have decentralized sequence acquisition, increasing the number of users performing sequence analysis all over the world. Technical challenges in use of bioinformatics software [17,18] and difficulties in utilization of available computational resources [12,19] impedes analysis, interpretation and full exploration of sequence data. In the following two chapters, we introduce a software package called Cloud Virtual Resource (CloVR) for automated and portable sequence analysis. CloVR is free open source software available at http://clovr.org.

In building the CloVR software, we relied on two enabling technologies, virtual machines (VM) and cloud computing platforms [60], to address software and hardware requirements for bioinformatics analysis. The CloVR software is a single virtual machine (CloVR VM) containing pre-configured and automated sequence analysis pipelines, suitable for easy installation on a personal computer and with cloud support for increased analysis throughput. In this chapter, the technical architecture of the CloVR VM is described and evaluated. Then in Chapter 8, we present a case study that evaluates the costs and required resources of microbial sequence analysis using protocols bundled in CloVR.

While, cloud computing platforms provide computing resources for anyone to access and use over the Internet on-demand, utilization of bioinformatics tools and pipelines on such distributed systems requires technical expertise to achieve robust operation and intended performance gains [12]. Also despite emergence of tools and methods designed for cloud-ready frameworks, there is greater availability of bioinformatics tools, analysis pipelines, and standardized methods that are designed for distributed computation on static compute clusters [101,102]. Challenges in data storage and transfer over the network add to the complexity of using cloud computing systems [119].

In building the CloVR VM, we have addressed the following technical challenges in using the cloud:

i) Elasticity and ease-of-use, clouds can be difficult to adopt and use requiring operating system configuration and monitoring; many existing tools and pipelines are not designed for dynamic environments and require re-implementation to utilize cloud-ready frameworks such as Hadoop;

ii) Limited network bandwidth, Internet data transfers and relatively slow peer-to-peer networking bandwidth in some cloud configurations can incur performance and scalability problems; and

iii) Portability, reliance on proprietary cloud features, including special storage systems can hinder pipeline portability; also, virtual machines, while portable and able to encapsulate complex software pipelines, are themselves difficult to build, configure, and maintain across cloud platforms.

The architecture of CloVR addresses these challenges by

i) simplifying use of cloud computing platforms by automatically provisioning resources during pipeline execution;

ii) using local disk for storage and avoiding reliance on network file systems;

iii) providing a portable machine image that executes on both a personal computer and multiple cloud computing platforms;

In this chapter, we describe the technical architecture of CloVR and evaluate some of the features, particularly portability, scalability, use of local storage, and automated access of cloud resources.

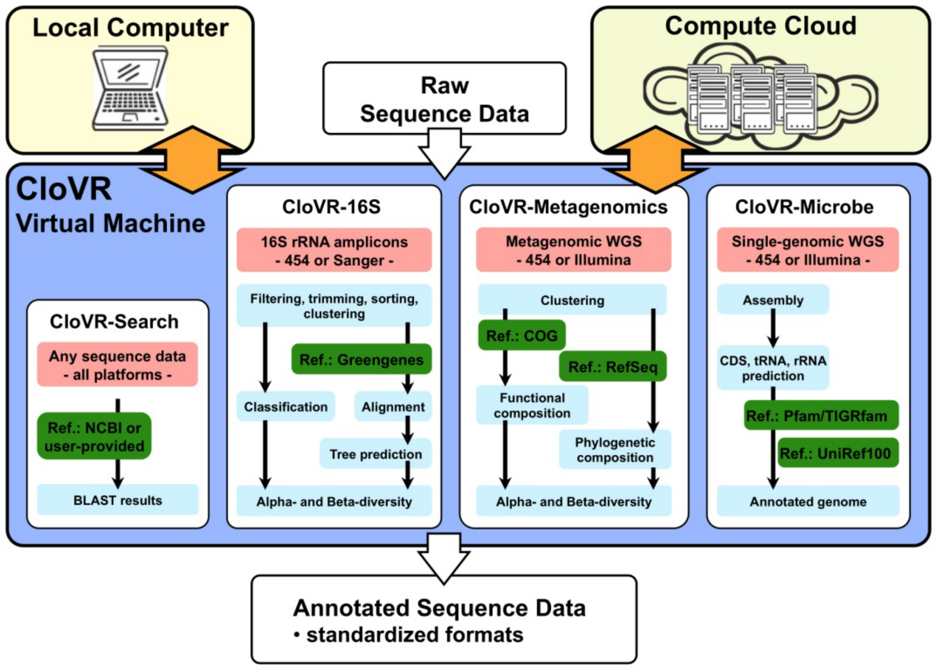


Figure 7.1 - Schematic of the automated pipelines provided in the CloVR virtual machine
The CloVR virtual machine includes pre-packaged automated pipelines for analyzing raw sequence data on both a local computer and cloud computing platform.

Implementation

Architecture overview

CloVR is a virtual machine (VM) that executes on a desktop (or laptop) computer, providing the ability to run analysis pipelines on local resources (Figure 7.1). CloVR is invoked using one of two supported VM players, VMware [55] and VirtualBox [56]; at least one of which is freely available on all major desktop platforms: Windows, Unix/Linux, and Mac OS. On a local computer, CloVR utilizes local disk storage and compute resources, as supported by the VM player, including multi-core CPUs if available. To access data stored on the local computer, users can copy files into a “shared folder” that is accessible on both the VM and the local desktop. Once inside the shared folder, CloVR can read this data for processing. Similarly, CloVR writes output data to this shared folder, making pipelines outputs available on the desktop. This shared folder feature is supported by both popular desktop virtual machines players, VMware and VirtualBox.

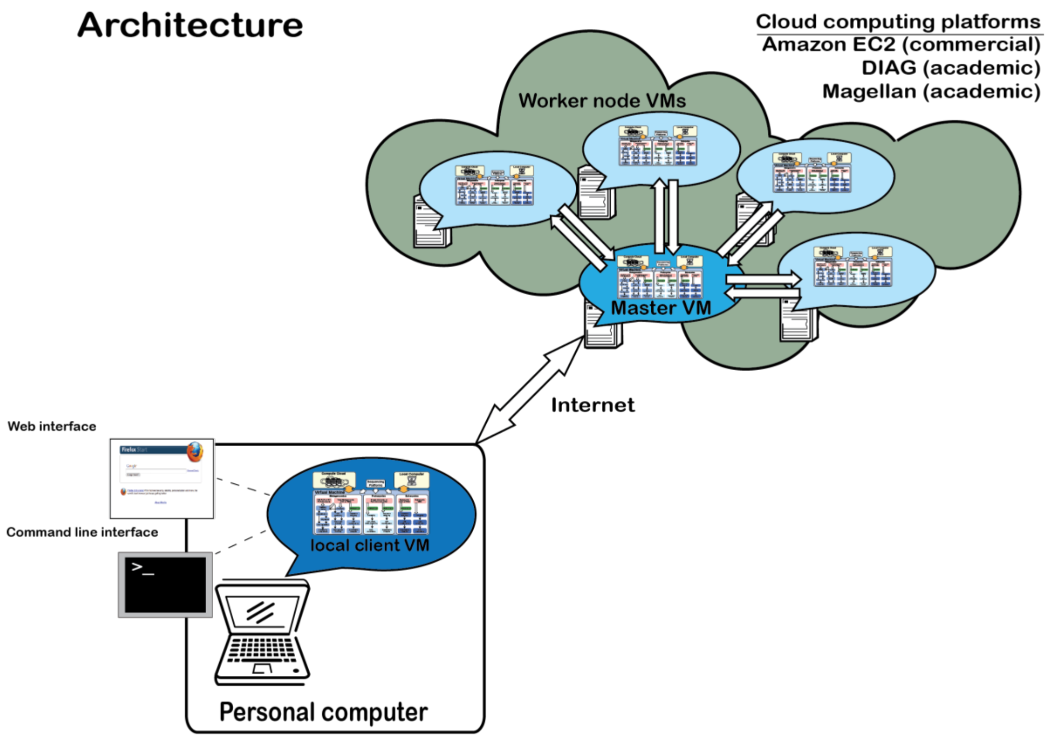


Figure 7.2 - Architecture of the CloVR application
CloVR provides a virtual machine (VM) that is run on user's local desktop or laptop computer. The user interacts with the local VM via a command line or web interface to execute pipelines. Optionally, clusters of additional VM instances are provisioned on supported cloud platforms for increased throughput. Each cluster has a master VM instance that provides services for GridEngine [153] and Hadoop [154]. Input data and output data is transferred between the local VM and a master VM instance in the cloud over the Internet.

Optionally, the CloVR VM can be configured to automatically access a cloud computing provider for additional resources. Supported clouds include the commercial Amazon

Elastic Compute Cloud [61] and the academic platforms DIAG [63] and Magellan [62]. In utilizing the cloud, multiple copies of the CloVR VM execute concurrently and interact as a cluster for parallel processing of data (Figure 7.2).

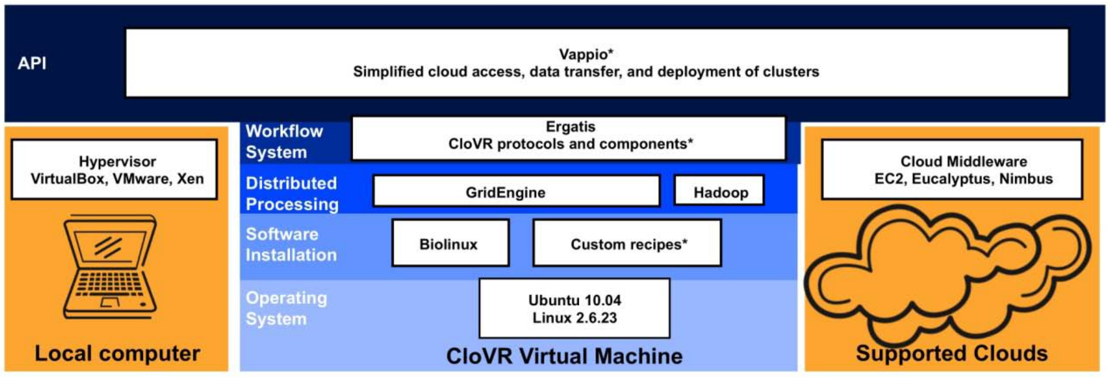


Figure 7.3 - Components of the CloVR virtual machine
The CloVR virtual machine includes pre-installed and pre-configured software dependencies (blue) on an Ubuntu operating system. Key software that is bundled with the VM is shown. The (*) indicates software that was developed as part of the CloVR project

Components of the CloVR VM

To address technical challenges associated with software installations and pipeline configurations, the CloVR VM comes bundled with all required software pre-installed and pre-configured, (Figure 7.3). The bundled software includes a base operating system (Ubuntu 10.04 [155] + BioLinux [17]), job schedulers (Grid Engine [153], Hadoop [154]), and a workflow system (Ergatis [102]). In addition, numerous open source bioinformatics tools are pre-installed and bundled into automated pipelines for pre-defined analysis protocols (Table 8-1).

Building the virtual machine images

An automated build and configuration process is used to generate the virtual machine images in formats compatible with both virtual machine players and cloud computing platforms. A specially configured VM (CloVR buildbox, http://clovr.org/developers/) running the Hudson continuous integration server [156] is used to schedule and automate the builds. The build starts with a skeleton Ubuntu 10.04 disk image [157]. During the build process, a series of recipes are applied to the skeleton image to install all the necessary software, resulting in a fully installed disk image. Finally, this disk image is converted into formats for VMWare (.vmdk files) and VirtualBox (.vdi files). The raw disk image is also uploaded to Amazon EC2 as an AMI and DIAG as a Xen compatible image for Nimbus [158].

Components of a CloVR automated pipeline

The CloVR VM (version 0.6) currently includes four pre-packaged and automated analysis protocols (Figure 7.1): (i) a simple parallelized BLAST [159] search protocol (CloVR-Search ver 1.0 [160]); (ii) a comparative 16S rRNA sequence analysis pipeline (CloVR-16S ver 1.0 [161]); (iii) a comparative metagenomic sequence analysis pipeline (CloVR-Metagenomics ver 1.0 [162]); and (iv) a single microbial genome assembly and annotation pipeline (CloVR-Microbe ver 1.0 [163]). For each protocol, a limited set of configuration options and pre-defined input files are supported, such as SFF, FASTA, and FASTQ. Output files are generated in standardized formats, such as FASTA and Genbank flat files.

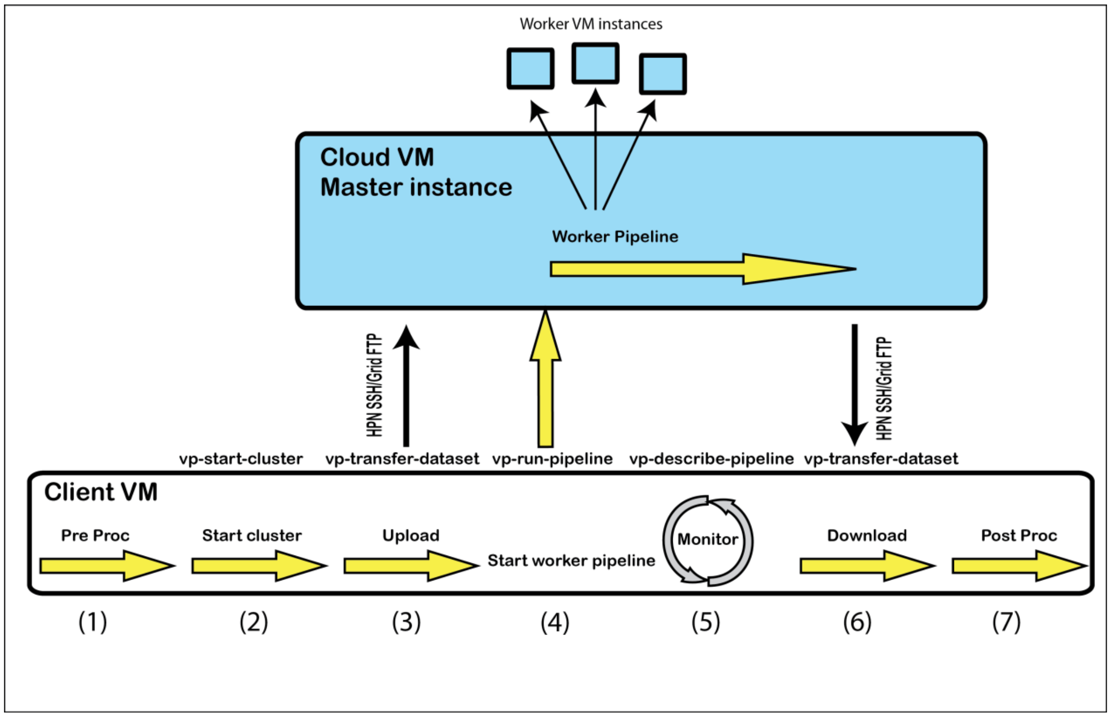


Figure 7.4 - Steps of an automated pipeline in CloVR

Each CloVR protocol is implemented as two discrete pipelines: 1) a worker pipeline and 2) a wrapper pipeline. CloVR uses the Ergatis workflow engine [102] to describe and execute each of these pipelines. The worker pipeline implements and performs the particular analysis protocol, while the wrapper pipeline manages automated use of the cloud from the desktop using the local VM client (Figure 7.4). Each wrapper pipeline is composed of seven primary phases: (1) pre-processing, including quality and integrity checks of input data; (2) starting a remote cluster for distributed processing; (3) data upload to the cloud; (4) execution of the worker pipeline; (5) monitoring of the worker pipeline; (6) data download from the cloud and (7) post-processing. Steps (2), (3), (6) are only executed when utilizing a remote cloud platform.

To implement each of these steps in the wrapper pipeline, we built a set of utilities and a web services API in a software package called Vappio (http://vappio.sf.net). Vappio is built on top of the Amazon EC2 API [164] for managing images, instances, and authentication key pairs. Vappio provides functions for managing (i) clusters, (ii) datasets, and (iii) protocols and pipelines. A summary of the Vappio functions and web services follow:

(i) Clusters. On the cloud, clusters of CloVR VM instances are configured for parallel processing. CloVR utilizes these clusters as temporary resources during pipeline processing, provisioning a new cluster for each pipeline, and terminating the cluster upon pipeline completion. Each clusters runs an instance of both Grid Engine [153] and Hadoop [154] for job scheduling. Clusters are composed of a single master node and one or more worker nodes (Figure 7.2). The first VM that is started in a cluster is designated as the master node. Subsequent VMs are designated as worker nodes and automatically registered with the master node and added to the cluster upon boot of the image. The user-data environment on the cloud platforms is used to configure each node type and associate master and worker instances during image boot. Worker nodes are configured in Grid Engine queues for receiving a number of work units based on the number of CPUs that are available on the instance. The client CloVR VM running on the user's desktop is also considered a cluster, named 'local' that is both a master and worker type.

To manage the cluster on the cloud, Vappio provides web services to dynamically start (*vp-add-cluster*), resize (*vp-add-instances*) and terminate (*vp-terminate-cluster*) clusters of VM instances. These web services in turn utilize EC2 API calls [164], including *ec2-run-instances, ec2-terminate-instances,* and *ec2-describe-instances*. In addition to executing the EC2 API calls, the Vappio web services manage the configuration of Grid

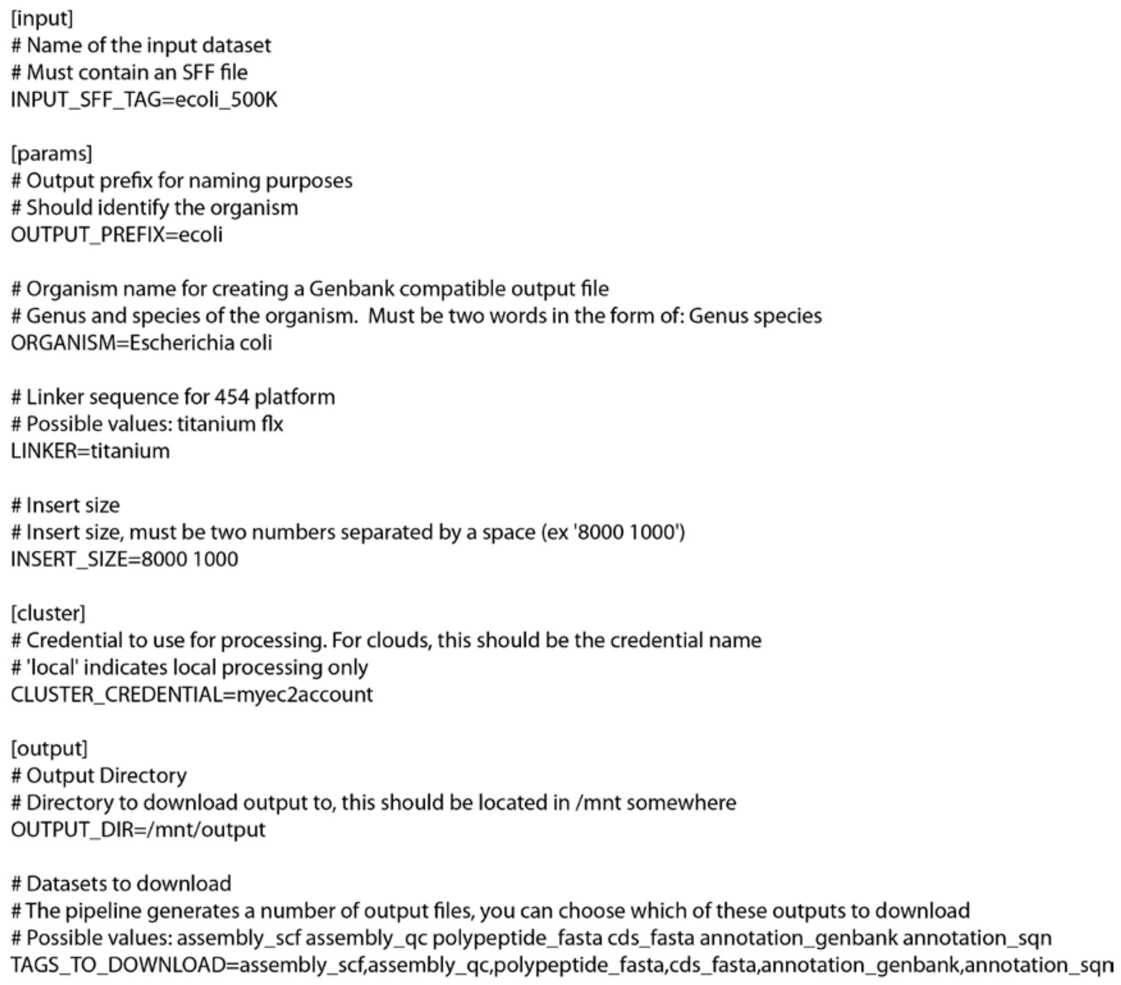
Engine and Hadoop on each instance as the instance is started and added to the cluster, or terminated and removed from the cluster.

In order to access the cloud, user account and authentication information is required and provided by the cloud provider. To simplify access to the cloud during pipeline execution and without jeopardizing security, Vappio provides a unique identifier, called a credential name, for each cloud account. During an initial configuration, the credential name is configured and associated with the cloud account and authentication keys using the Vappio web service, *vp-add-credentials*. This credential name is then used to refer to the account during subsequent Vappio web service calls during pipeline execution.

All communication and data transfer between a user's desktop and the cloud is managed by the client CloVR VM running on a local computer. The local client VM communicates with the master CloVR VM on the cloud to transfer data, invoke worker pipelines, and monitor pipeline state (Figure 7.4**)**. To provide security and help ensure data privacy, each remote cluster of CloVR VMs uses a unique authentication key. This key is used to enable secure data transfer between instances with Secure Shell (SSH) both within the cloud and over the Internet and between the local client VM and master cloud CloVR VMs.

(ii) Datasets. In Vappio, datasets are described as lists of files or Uniform Resource Locators (URLs) that are accessible by a cluster or the local client CloVR VM. User provided sequence data, reference data, and outputs generated by the CloVR analysis pipelines are all managed as datasets. Datasets are moved between a local desktop and disk storage on the remote cluster as needed for processing (Figure 7.4**,** Steps 3 and 6). Vappio provides utilities for 1) registering new data sets with the cluster (*vp-add-*

dataset), 2) transferring datasets between clusters (*vp-transfer-dataset*), and 3) describing information about a data set (*vp-describe-dataset*).



```
[input]
# Name of the input dataset
# Must contain an SFF file
INPUT_SFF_TAG=ecoli_500K

[params]
# Output prefix for naming purposes
# Should identify the organism
OUTPUT_PREFIX=ecoli

# Organism name for creating a Genbank compatible output file
# Genus and species of the organism.  Must be two words in the form of: Genus species
ORGANISM=Escherichia coli

# Linker sequence for 454 platform
# Possible values: titanium flx
LINKER=titanium

# Insert size
# Insert size, must be two numbers separated by a space (ex '8000 1000')
INSERT_SIZE=8000 1000

[cluster]
# Credential to use for processing. For clouds, this should be the credential name
# 'local' indicates local processing only
CLUSTER_CREDENTIAL=myec2account

[output]
# Output Directory
# Directory to download output to, this should be located in /mnt somewhere
OUTPUT_DIR=/mnt/output

# Datasets to download
# The pipeline generates a number of output files, you can choose which of these outputs to download
# Possible values: assembly_scf assembly_qc polypeptide_fasta cds_fasta annotation_genbank annotation_sqn
TAGS_TO_DOWNLOAD=assembly_scf,assembly_qc,polypeptide_fasta,cds_fasta,annotation_genbank,annotation_sqn
```

Figure 7.5 - Example of specification files used for running pipelines

(iii) Protocols. Pre-defined analysis protocols are invoked for data analysis using a single configuration file (Figure 7.4**,** Step 4). Vappio provides utilities for configuration and invocation of analysis protocols with the services *vp-describe-protocol* and *vp-run-pipeline*. An example of the specification file for CloVR-Microbe454 produced by *vp-describe-protocol* is shown in Figure 7.5. The specification file includes references to input data sets, configurable analysis parameters, and, optionally, references to cloud account credentials for accessing the cloud. Protocols are executed with *vp-run-pipeline*, which accepts the specification file as input. Once executed, we refer to the running instance of the protocol as a pipeline. The status of pipelines is monitored with the service *vp-describe-pipeline*.

To ensure transparency of the CloVR-supported analysis, each CloVR protocol is described by two documents: (1) An abstract workflow XML file that is used by the Ergatis workflow engine to execute the protocol and (2) a human readable standard operating procedure (SOP) document that describes the protocol in detail. The abstract workflow XML is an exact description of the executions used to perform the analysis. The SOPs describe each step of the pipeline, including software tools, software versions, and parameters used. The protocol SOPs are published online with references stored on the VM and in pipeline configuration files, allowing for association between an analysis result and protocol description.

To ensure reproducibility of individual analysis results, CloVR uses the following additional principles: 1) All pipelines are executed using the Ergatis workflow system that tracks process flow and exact parameters invoked at each step in an XML file. 2) As part of the CloVR software installation process, versioning is applied to each analysis protocol, reference data set, and to the CloVR VM image itself. All results generated during CloVR pipeline runs have references to these versions.

Data storage and transfers

Local disk storage on the personal computer is used to store all input data and results generated during an analysis. Input data is copied to the cloud if needed. To improve network transfer performance, CloVR uses high performance Secure Shell (HPN-SSH) [165] to transfer files. Rsync [166] is also used in conjunction with HPN-SSH to avoid redundant data transfers. Since all network transfers between a local desktop and the cloud are managed by CloVR VMs, data transfer with these tools is automatic, invisible to the user and does not require further software installations or configurations by the user.

The pipelines in CloVR are configured to avoid unnecessary data transfers for both local and cloud-based execution modes. For example, several of the supported protocols rely on publically available reference data sets that are either permanently hosted in the cloud or at an Internet accessible URL. When executing CloVR pipelines in the cloud, pipelines will utilize reference datasets hosted on the cloud whenever possible. For local execution, the reference datasets must first be downloaded to the local VM over the Internet. CloVR ensures such local transfers happen only once, the first time the data is accessed, and the reference data is then saved locally for subsequent access.

For data storage on the cloud, CloVR utilizes local disks and does not require any access to a shared file system, such as a NFS server. Instead, all intermediate results or temporary files are stored on the local ephemeral disk storage provided to each VM instance. Under this model, worker nodes must receive copies of input data from the cluster master node before beginning work. This is implemented using the job prolog feature of Grid Engine to copy input data prior to job execution. Similarly, output data is copied back to the master node using the job epilog feature of Grid Engine. To provide robustness and scalability, all data transfers to and from the head node are also scheduled as jobs in Grid Engine queues named staging.q and harvesting.q. The number of slots in these queues allow for control over how many simultaneous transfers a master node will process. HPN-SSH and rsync are used to perform the transfer between instances in the cloud.

In some cases, pipelines use reference data sets or intermediate outputs that need to be accessed on every instance in a cluster. A single directory (the staging directory, eg. /mnt/staging/) is used to mirror such data to all instances in the cluster. Rather than rely exclusively on the master node to provide the mirror, worker nodes can share copies of the staging directory in a peer-to-peer fashion to provide additional bandwidth and improve throughput. Upon receiving a complete copy of the staging directory, worker nodes are added to a Grid Engine queue (named stagingsub.q) indicating that they can mirror copies to peers. Grid Engine queues are also used to limit the number of transfers between each worker node.

Upon pipeline completion, final outputs are transferred from the master node instance outside the cloud to the local VM. After output has been transferred back to the local CloVR VM, the cluster and all associated local storage is no longer needed and instances are terminated.

Automatic resource provisioning in the cloud

Cloud resources are automatically provisioned during execution of CloVR pipelines. To accomplish this, steps are added to the pipelines allocate additional cloud resources if necessary. Pipelines that are configured to run exclusively on the personal computer skip these resource allocation steps. To determine the number of compute instances needed for these protocols, custom scripts consider one or more of these factors: 1) hard-coded assumptions about expected resource utilization and 2) instance limits from the cloud provider or user 3) estimation of runtime from input data.

To support genome assembly of Illumina data using Velvet, CloVR pipelines are hard-coded to start a single high memory instance type (m2.xlarge) on Amazon EC2 prior to running assembly that provides 17.1GB of RAM, which in our testing is sufficient for assembly of single bacterial genomes.

For three of the pre-packaged protocols in CloVR (Microbe, Metagenomics, and Search), BLAST searches are the primary processing bottleneck. An estimation of total BLAST runtime can serve as a good approximation to predict the overall pipeline runtime. A default minimum of 5 c1.xlarge instances providing a total of 40 CPUs is started to support BLAST steps in these pipelines. For the CloVR-Search and CloVR-Metagenomics protocols, a prediction of total CPU runtime is estimated based on the input data using Cunningham [167] to determine how many instances to start prior to search.

For a particular search database, BLAST runtimes can vary depending on the length and composition of query sequences. Cunningham, which was implemented as part of the CloVR project, rapidly estimates BLAST runtime by comparing *kmer* profiles (*k*=3 for protein, *k*=11 for DNA including reverse complemented sequence) for a reference database and the input query sequence. The number of matching *kmers* between the query subsample and the reference database are saved. The *kmer* profile of the reference database is pre-calculated and saved so that only the *kmer* profile of the query sequence needs to be calculated during pipeline execution. Then, a different linear model for BLASTN, BLASTP, and BLASTX, is built in the common form:

$$T \approx \beta_1 M + \beta_2 L$$

where the runtime T depends on the number of shared seed pairs M, and the average query sequence length L. Calibration runs of each BLASTN, BLASTP, and BLASTX using randomly selected shotgun metagenomic datasets lised in {White, 2011 #527} on c1.xlarge Amazon EC2 machine types are used to obtain parameters β_1 and β_2. Default BLAST parameters were used with the exception of '-b 1 -e 1e- 5 -F F'.

Also impacting runtimes is number and size of partitions that are used for parallel processing. In CloVR, BLAST searches are run in parallel by dividing the input query multi-FASTA files into partitions and executing a search of each partition concurrently against the reference database. Over-partitioning of the data in which jobs finish in a very short in duration can lead to inefficient use of resources and increased runtimes, since there is overhead in scheduling and invocation of each job. Provided a runtime estimate, the partition size P for each BLAST query is obtained by

$$P = \frac{N_q}{T/R}$$

where N_q is the total number of query sequences, T is the estimated CPU runtime from Cunningham or some other estimation procedure, and R is a configurable parameter for the preferred execution time for a single data partition (default: 2 hours). The support for runtime estimates is provided as a configurable module that reads the pipeline configuration and produces an estimate. This allows for custom modules for runtime prediction in the future using some other logic.

The cloud provider may impose a limit on the maximum number of instances that can be started by a user (eg. Amazon EC2 imposes a default of 20 instances per account, which can be raised on request). CloVR also has an instance limit option in each pipeline specification file. CloVR prevents attempts to start more than this number of instances for a particular pipeline.

Results

We evaluated four features of the CloVR architecture: portability across computing platforms, support for elastic provisioning of resources, scalability of clusters of instances, and use of local data storage on the cloud.

Table 1 – Portability of the CloVR VM

	Local PC (Intel Xeon 5130) Max No CPUs : 4	**DIAG** (medium instance) Max No. instances: 5 Max. No CPUs : 20	**Amazon EC2** (c1.xlarge instance) Max No. instances: 18 Max No. CPUs: 80
	Runtime	Runtime	Runtime
Assembly	29 min	25 min	28 min
Annotation	2 days 6 hrs 26 min	9 hrs 30 min	7 hrs 2 min
Total	2 days 7 hr 5 min	9 hrs 55 min	7 hrs 30 min

CloVR runs on the desktop and can utilize resources at multiple cloud providers

To demonstrate the portability of CloVR, we executed a single analysis protocol (CloVR-Microbe) analysis on a personal computer and two cloud computing platforms (Table 2). The input data was comprised of 250,000 454 FLX Titanium sequencing reads of the bacterium *Acinetobacter baylyi* totaling ~89Mbp is and expected to cover the ~3.5Mbp genome at 25-fold coverage. Identical output, comprised of 38 contigs (N50=262Kbp) and 3417 predicted protein coding genes was obtained on all three platforms. For local analysis, a 4-CPU VM with 8GB of RAM was used. When using the cloud platforms, the local client VM can be executed in as little as 2GB of RAM. The DIAG and EC2 platforms allowed for execution of steps of the protocol in parallel offering 4-CPUs per “medium” instance type on DIAG (8GB RAM) and 8-CPUs per “c1.xlarge” instance type on EC2 (7.5GB RAM).

Our evaluation of the CloVR-Microbe protocol demonstrates the ability to run a genome assembly and annotation protocol both locally on the cloud for increased throughput. A single configuration setting is changed to invoke the pipeline on either the personal computer or the supported clouds.

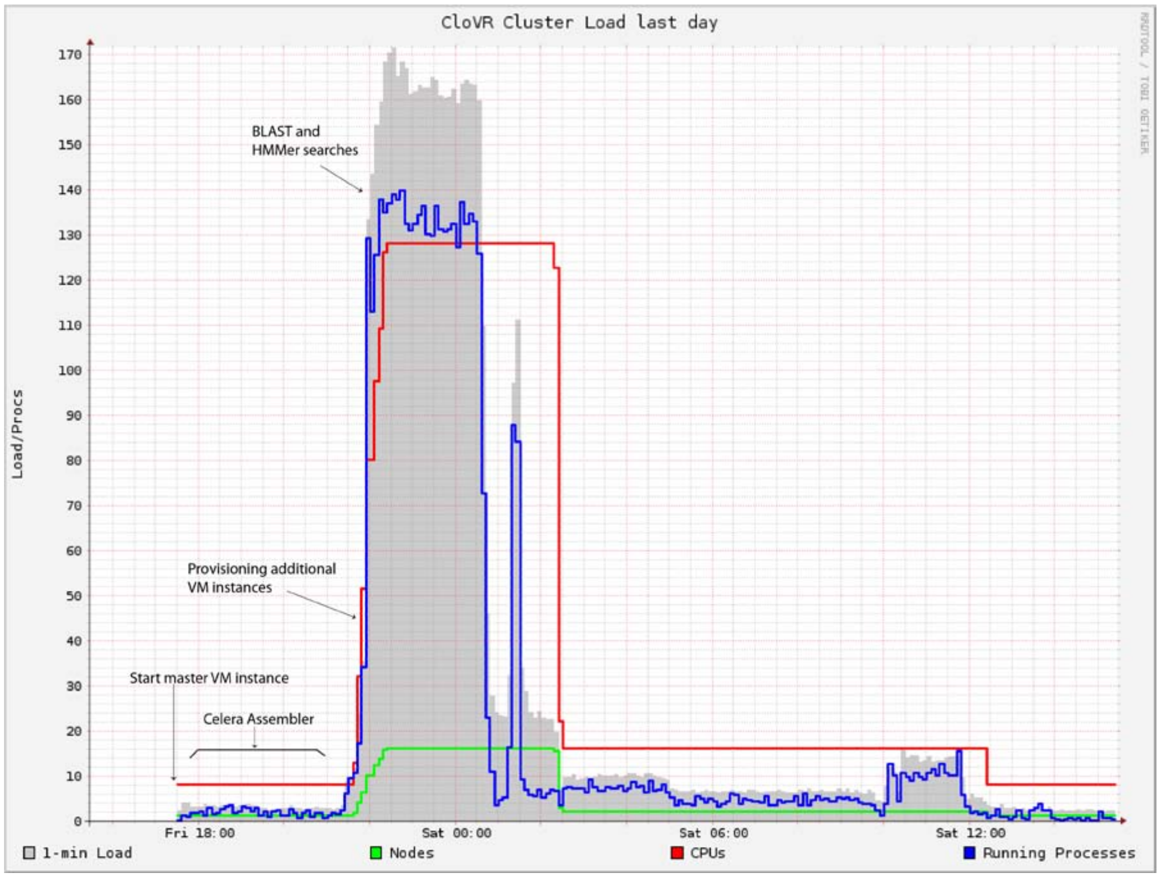


Figure 7.6 -Execution profile of an analysis with CloVR-Microbe
The example shows the number of CPUs and their workload that are part of an Amazon EC2 cluster that is used to run CloVR-Microbe. The BLAST and HMMER searches are amenable to parallelization and are executed across a cluster of CPUs, while genome assembly processing is run on a single CPU.

CloVR provides automated resource provisioning in the cloud

Elasticity, i.e. dynamic provisioning of resources, is a primary feature of the cloud and allows for the addition of computational resources on-demand. An example of this dynamic capability is provided for the microbial genome assembly and annotation steps of the CloVR-Microbe pipeline (Figure 6). In this example, the CloVR-Microbe protocol was used to perform whole genome assembly and annotation on 500,000 sequencing reads from the 454 Titanium FLX platform. The local VM client first starts a remote (master) VM instance on the cloud. The input sequencing reads (676 MB compressed

SFF file) were copied to this instance and genome assembly is completed on a single c1.xlarge VM instance, using no more than two virtual CPUs. Then, prior to genome annotation, 15 additional CloVR VM instances were allocated to improve processing throughput. A configurable parameter limits the number of instances that are added. Idle instances are subsequently terminated automatically upon job completion on an hourly timer. Importantly, this provisioning and termination of resources is automatic and does not require the user to interact with the remote cluster on the cloud.

This protocol was migrated to the CloVR VM from a pipeline used in the online service IGS Annotation Engine [109], and was not optimized for performance on a cluster. Many steps of later steps in this protocol are amenable to parallel computation but have not been implemented to run across a cluster, currently requiring hours to run on a few CPUs. Improved support for parallel computation and other optimizations are left as future work.

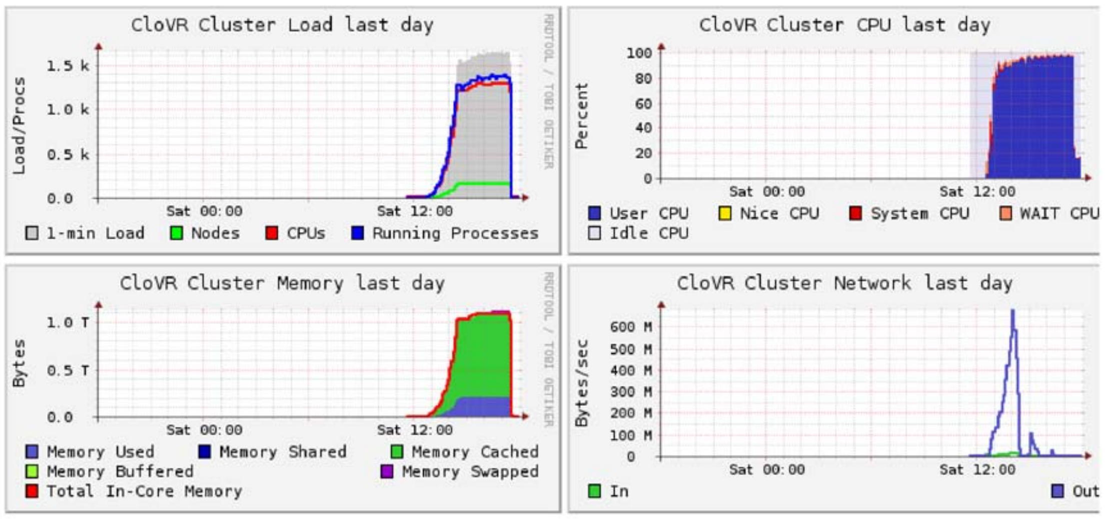


Figure 7.7 - Dynamically allocated cluster of CloVR VM instances running BLAST
A cluster of CloVR VMs is deployed on-the-fly and scaled to 160 c1.xlarge Amazon EC2 instance types (totaling 1280 virtualized CPUs).

To assess the scalability of the CloVR architecture, we executed BLASTX searches using CloVR-Search on clusters composed of up to 160 c1.xlarge instances, comprising 1280 CPUs of a random sample of ~100M nucleotides from a oral microbiome sample sequenced with a 454 Titanium FLX machine (unpublished) against NCBI non-redundant protein database (Figure 7.7). This BLASTX search ran at a throughput of ~36.9Mbp per instance hour for a c1.xlarge Amazon EC2 instance type, at an estimated cost of ~$108 per hour for 160 instances. A subset of the data was used for the evaluation; the estimated runtime for the complete sample of 561Mbp is ~15 hours (19,940 c1.xlarge CPU hours) at a total cost of ~$1641.

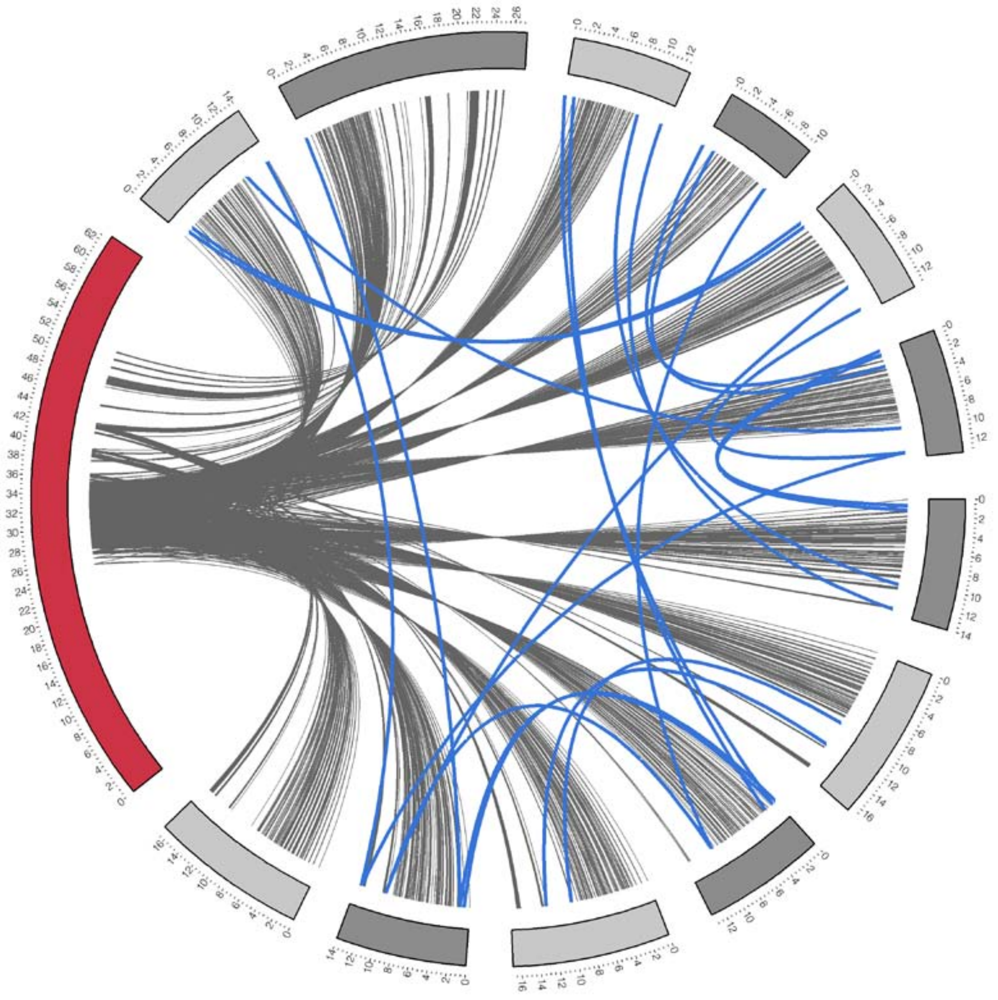


Figure 7.8 - Visualization of data transfers between instances over time in a cluster of CloVR VMs. Each arc represents the lifetime of a CloVR VM instance with the time labeled relative to bootup of the instance. The red arc is a master node CloVR VM and grey arcs are worker VM instances. Data transfers between master and worker instances are shown as grey lines. Transfers between worker instances are shown as blue lines.

CloVR uses local disk and does not rely on network file systems

Bioinformatics tools typically operate on files and expect a file system for reading and writing data. Bottlenecks in reading or writing data on a shared, network-based file system, such as NFS [168], can cause performance problems during processing, especially when many concurrent processes are executing against the shared resource.

CloVR does not rely on network file systems for processing and, instead, uses local disk to avoid introducing data transfer bottlenecks during computation. To achieve this, input files must be transferred to compute hosts (worker instances in Figure 7.2). These file transfers between master and worker node types are made prior to computation for inputs and subsequent to job completions for outputs. A depiction of these data transfers during a run of CloVR-Microbe is shown in (Figure 7.8). Some of the data transfers are required to support parallel execution of BLAST, where the input query FASTA files is split into fixed size partitions and each partition is searched independently and in parallel. In CloVR, the input FASTA data and output BLAST report for each partition is copied between the master VM and a worker VM before and after processing.

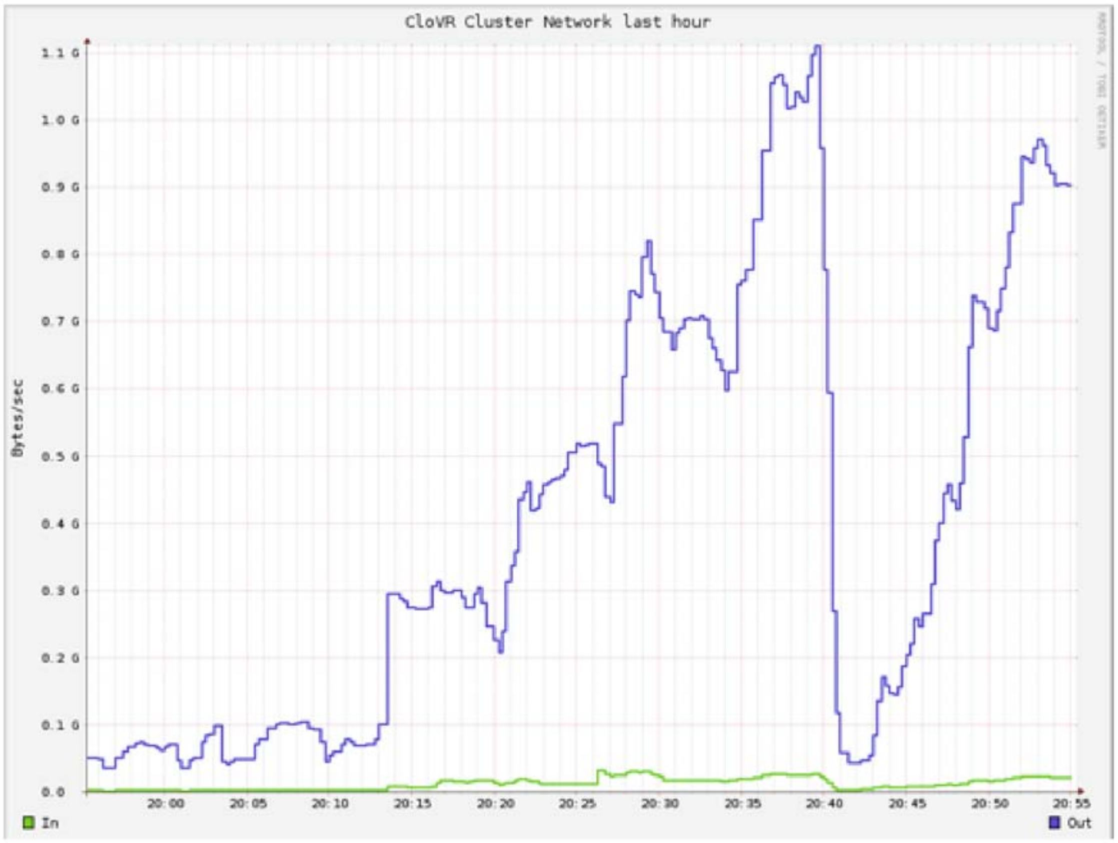


Figure 7.9 - Network throughput on a cluster of CloVR VMs

The aggregate network throughput as measured by Ganglia [169] during a peer-to-peer data transfer on a cluster of 160 c1.xlarge instances on Amazon EC2.

Reference data sets and some intermediate outputs need to be accessed by all VM instances in a cluster. To improve distribution of these data sets, a peer-to-peer data transfer scheme is used for sharing intermediate results and reference data sets (blue lines in Figure 7.8). To evaluate the performance of these data transfers, we tested the throughput for providing 3.1GB of compressed reference data for BLAST to 100 c1.xlarge VM instances (Figure 7.9). During this execution, instances came online in a staggered fashion and received copies of the reference data upon boot of the instance. Aggregate throughput exceeded 1.1 GB/second. By comparison, network transfer speeds between a pair of c1.xlarge instance type on Amazon EC2 network were found to be typically fall below ~40MB/second.

Discussion

CloVR reduces bottlenecks in sequence analysis by using two related technologies: virtual machines and cloud computing. By using virtualization technology, CloVR simplifies deployment of complex bioinformatics workflows by providing a single executable (the virtual machine) that can execute on a personal computer. In addition, by supporting Amazon EC2 and other cloud computing platforms, CloVR provides access to large distributed computing resources, providing a potential alternative to building and maintaining in-house infrastructure for computational analysis.

CloVR is implemented as a software-as-a-service solution for sequence analysis, although, in contrast with Internet based services, CloVR runs directly on a personal computer rather than a central server. Web accessible workflow systems, such as Galaxy [101] or Taverna [104], provide a centralized analysis resource for a lab or community that is accessible over the network and typically executes on dedicated resources where users upload data for centralized processing. This centralization of services is in contrast to the current decentralization in genome sequence generation. CloVR provides an alternative decentralized model, where each user runs an instance of the CloVR VM on their personal computer that is independent from others in a multi-user environment. By running on the desktop, CloVR can utilize local compute and storage resources and avoid transfer of user generated sequencing data over the Internet in some cases. The CloVR architecture also avoiding contention for centralized web servers for processing, while still supporting the shared cloud computing resource for increased throughput.

As the number of computing cores available in a personal computer is expected to increase in the coming years, the desktop support in CloVR provides an opportunity to utilize substantial computing power on a local machine; potentially avoiding need of the cloud entirely.

CloVR does not provide all of the features of a genomic workbench, in particular it does not provide a web interface for running and configuring individual analysis tools. While genomic workbench systems have focused on making individual tools easy to run and integrate into pipelines, many projects rely on static, standardized analysis pipelines. In contrast to genomic workbenches that provide extensive choices of tools, CloVR provides pre-defined standard pipelines that integrate tools for particular analysis objectives so that no configuration or expertise with individual tools is required. This level of automated processing is particularly useful for users that find choice of bioinformatics tools overwhelming and instead seek recommendations for best practices. While our work has focused on providing automated pre-defined pipelines, by providing a VM, all the bioinformatics tools included in CloVR can be run from a command line terminal within the VM. This mode of access may also be of interest for experienced users. While, at the time of this study, CloVR does not provide a web interface, work on a web-based user interface is in progress. Similar to a local interaction with the VM, the web interface will run locally as a service on the VM, running on a user's desktop.

CloVR provides utilities for building private clusters of VM instances on-demand. A few other systems, Nimbus one-click clusters [170], Galaxy CloudMan [171] and StarCluster [172], are also designed to deploy clusters of instances in the cloud. In contrast to these systems, CloVR users are not expected to start, manage, or resize clusters of VM instances in the cloud. Instead, pipelines include steps to provision these resources automatically. This ability enables cost savings in the case of commercial clouds, by allocating resources only as they are needed ("just-in-time").

To help ensure compatibility with multiple cloud providers, CloVR avoids reliance on proprietary features of individual cloud providers, instead utilizing only three EC2 API calls during pipeline execution (ec2-run-instances, ec2-terminate-instances, and ec2-describe-instances). Such core functions of the EC2 API are supported by all the clouds evaluated and are becoming a standard in middleware that provides cloud services. CloVR is ready to support an emerging cloud computing platform that provides this baseline interface.

The architecture of CloVR, which utilizes utilize Grid Engine [153] for job scheduling and uses local disks for storage, allows for migration of tools and pipelines to the cloud without reimplementation. All of the analysis protocols provided on the CloVR VM were migrated from a non-cloud version that previously executed on a static local compute cluster. This approach is in contrast to cloud-ready frameworks like Hadoop, which are designed algorithms that follow MapReduce [115], often requiring new methods or reimplementation of existing tools to utilize the framework. As more tools are becoming available utilizing MapReduce [116,117,118], Hadoop is included on the VM for future integrations of new tools that take advantage of this framework.

Compute clusters often rely on centralized, shared storage systems or file servers to simplify access to data for users and pipelines. As part of the design to be both portable and scalable on cloud computing networks, CloVR does not rely on a shared, network file system, such as NFS, for storage. Instead, CloVR relies on local storage on either the users' desktop to store pipeline input and output, or temporary disk storage available the cloud VM instances during pipeline execution. Other distributed storage systems, such as Amazon S3 [173] or HDFS [174] require use of specialized utilities to read and write data. Rather than retool software to use these systems, all tools integrated into CloVR pipelines operate on files and local file systems without any required modification of the included analysis tools. Also, by using local disk for storage rather than the network, CloVR can be expected to run on commodity cloud systems with relatively slow networking and without reliance on the specialized storage features of cloud providers, such as Amazon Elastic Block Storage [175].

The CloVR architecture saves all pipeline inputs and output on the personal computer, enabling additional control on maintaining data privacy. In contrast, online bioinformatics resources require the user to relinquish some control over data, since sequences and metadata are uploaded and saved on a remote server for processing. Although CloVR transfers data to cloud servers for processing, CloVR uses the cloud as a temporary resource for processing and does not require that either inputs or results are stored on the cloud.

With the increasing volume of next-generation sequencing data, data transfer over the Internet can be an impediment for utilizing the cloud. In CloVR, the transfer of user inputs and sequence data to and from the cloud occurs over the Internet and can be slow. To address this bottleneck, CloVR currently utilizes HPN-SSH [165] for all data transfers. In addition, the CloVR VM includes the GridFTP fast file transfer protocol [176]. We may be able to utilize this protocol or others in the future to provide further speedup. Since all data transfers occur between local and remote CloVR VM as part of pipeline execution, use of new data transfer protocols can be implemented without user installation or configuration of either a server daemon or client utility.

A strategy for moving analysis to data, rather than transferring data over the network, has been raised as a potential solution to dealing with data transfer bottlenecks [19]. The portability of the VM provides such flexibility. The CloVR VM is 1.4GB compressed and can be easily transferred to computational resources that are co-located with large data sets. The CloVR VM already supports a similar model in the utilization of reference databases, such Uniref100 [177], which we hosted at the cloud to support the CloVR-Microbe genome annotation protocol.

CloVR promotes transparency of methods, by providing published and accessioned protocols for each pipeline, and enables reproducible research, by executing all pipelines in VM environment. For complex pipelines, reproducibility becomes increasingly difficult and virtualization and clouds have been recognized as ideal platforms to promote pipeline reproducibility [114]. CloVR realizes this potential by executing all steps on a portable VM that encapsulates the entire runtime environment, included versioned protocols and analysis results.

The CloVR pipelines are composed of multiple steps, only some of which are computationally demanding or support parallelization on multiple CPUs. To match pipeline needs with available resources, each CloVR pipeline includes steps to automatically provision cloud resources as needed. One strategy for efficient allocation of resources is to estimate runtimes for steps that execute in parallel so only as many resources are provisioned as can be used. As an example, CloVR-Search and CloVR-Metagenomics currently use a utility that we've built, named Cunningham [167], to estimate the runtime of BLAST during pipeline execution. This strategy is only meant to provide a rough estimate to avoid starting to many instances for small searches or too few instances for larger searches. In addition, a rough estimate of runtime can help avoid over-partitioning of the input query sequence data resulting in very short search time for each data partition, introducing overhead that degrades overall performance. Our use of Cunningham is meant to be illustrative and by making runtime estimates and cluster provisioning discrete steps in the pipeline, we can incorporate other runtime prediction methods in the future. While not all pipelines will consume a predictable amount of resources, the ability to predict runtimes can also be used to provide an a priori estimation to the user of how much an analysis will cost or whether a particular analysis is even feasible. We plan to explore providing such estimates as future work and anticipate this will be of much interest to users of the software.

While current protocols in CloVR focus on applications in microbial genomics, the platform is generic and useful for other genomics applications. Ongoing and future work will implement analysis pipelines for viral and eukaryotic genomics and transcriptomic projects. Also, while the whole genome and metagenomics protocols largely rely on BLAST for identifying sequence similarities, future work can add protocols that utilize more efficient tools and methods allowing for processing larger datasets. To enable further comparisons across protocols, more work is needed especially in standardization of analysis outcomes and data formats [94,178]. Analysis competitions and bake-offs are a good driver for these developments [179].

The CloVR VM can serve as a platform for the integration of additional or alternative tools and pipelines developed by other members of the research community. The recipe driven build process used in CloVR to build a single VM image for both the desktop and cloud computing platforms can also be used to build other custom VMs. A first step in this direction has been made by the use of the CloVR to create a VM for the QIIME package [180].

Chapter 8 - Resources and costs for microbial sequence analysis evaluated using virtual machines and cloud computing

Costs are commonly evaluated for sequencing technologies and continue to decrease [181], while costs for computational analysis have proven more elusive to quantify [120]. The availability of cloud computing platforms with transparent pricing has enabled attaching real dollar costs to bioinformatics workflows. Such costs provide both a measure of analysis efficiency and have practical value for project planning and budgeting. There is some debate of the economical feasibility of using commercial cloud computing platforms [19,119]. The evaluation of cost is complicated by poorly defined analysis tasks and difficulties in comparing analysis protocols across computational platforms. For example, researchers producing sequence data are confronted with the following questions:

(i) What are the available methods for sequence analysis in order to generate publishable results in standards-conforming formats?

(ii) What are the computational requirements for analysis?

(iii) Given a particular application, does it make sense to use Platform as a Service (IaaS) models, such as the Amazon EC2 cloud, or to invest in a local grid network?

(iv) What are the real dollar costs of analysis?

In this Chapter, we address these questions and provide cost and resource benchmarks for microbial sequence analysis using the CloVR platform (Chapter 7). These benchmarks are of interest to researchers, service providers, and funding agencies that invest in microbial genomics projects.

Table 8-1 Overview of CloVR analysis protocols

Protocol	Process	Tool	Input	Output
CloVR-Search [160]	Database search	BLAST [37]	nt or pep FASTA	BLAST output
CloVR-Microbe [163]	Assembly	Celera assembler [182] Velvet [183]	Raw sequence data (sff, nt.FASTA, nt.FASTQ)	nt.FASTA, BNK
	Gene prediction	Glimmer3 [48]		pep.FASTA
	tRNA prediction	tRNA-scan [184]		GBK, SQN
	rRNA prediction	RNAmmer [185]		GBK, SQN
	Functional annotation	BLASTX (+Extend Repraze [186]) against UniRef100 [177], COG [187] db HMMER [188] search against Pfam [189], TIGRfam [190]		Annotated GBK, SQN
CloVR-16S [161]	Quality checking	Mothur [110] Qiime [111]	nt.FASTA	nt.FASTA
	Taxonomic classification	RDP classifier [191]		raw output, summary reports
	Multiple sequence alignment	Mothur Qiime (PyNAST)		nt.FASTA alignments
	OTU clustering	Mothur (distance matrix) Qiime (uclust [192])		OTU list/table
	α-diversity analysis	Mothur (collectors, rarefactions, estimators)		summary reports/ diversity curves
	β-diversity/ comparative analysis	Metastats [193], custom R scripts, Qiime		summary reports/figures
CloVR-Metagenomics [162]	Clustering and artificial replicate removal	UCLUST	nt.FASTA	nt.FASTA
	Functional classification	BLASTX against COG db		raw output, summary reports
	Taxonomic classification	BLASTN against RefSeq db [129]		raw output, summary reports
	Comparative analysis	Metastats, custom R scripts		summary reports/figures

Abbreviations: nt, nucleotide; pep, peptide; BNK, Bank format; GBK, GenBank.; db, database; SQN, Sequin; Key bioinformatics tools utilized in each protocol are listed. For input, only the required inputs from the user for each analysis track are listed. For outputs, only the data saved from each step is listed.

Methods

Analysis protocols

In this study, we utilize a commercial cloud computing platform (Amazon EC2) and CloVR (Chapter 8) as a model for addressing questions of resource requirements and costs for microbial genomics applications utilizing high-throughput sequencing platforms. CloVR supports analysis for a broad variety of small to large-scale genomics applications. Four analysis protocols for microbial genome analysis (Table 8-1) were utilized in the study:

(i) a simple parallelized BLAST [159] search protocol (CloVR-Search 1.0 [160]);

(ii) a single microbial genome assembly and annotation pipeline (CloVR-Microbe 1.0 [163]).

(iii) a 16S rRNA sequence analysis pipeline (CloVR-16S 1.0 [161]);

(iv) a metagenomic sequence analysis pipeline (CloVR-Metagenomics 1.0 [162]);

These protocols were intentionally derived from existing methods for microbial sequence analysis, including the IGS Annotation Engine [109] for the protocol CloVR-Microbe, Mothur [110] and Qiime [111] for CloVR-16S, and BLAST for CloVR-Metagenomics. The reference for each CloVR pipeline provides a schematic diagram as well as a detailed document describing the standard operating procedure (SOP).

The 16S rRNA protocol allows for intra- and inter-group comparative analysis, and is based on methods from Mothur [110], Qiime [111], the RDP Bayesian classifier [194], and Metastats [193]. CloVR-16S calculates the number of non-redundant sequences within the total data set and uses a threshold of 50,000 above which the computationally expensive distance matrix calculation, which is part of the Mothur component of the pipeline, is not performed. The metagenomics protocol performs clustering of redundant sequences, a BLAST-based taxonomic and functional assignment against the NCBI microbial genome Reference Sequence collection (RefSeq) [195] and clusters of orthologous genes (COGs) [187] databases, respectively, and further allows for comparative analysis between subjects of interest. We also include an alternative metagenomics protocol that calls full and partial ORFs on shotgun fragments using Metagene [196], followed by functional annotation of predicted peptides using BLASTP against the NCBI COG database. The single microbial genome analysis protocol is based on the IGS Annotation Engine [109], with the addition that sequence assembly is performed using Celera Assembler [197] for 454 and Sanger platforms and Velvet [183] for Illumina platforms. This protocol performs a comprehensive annotation including gene prediction with Glimmer3 [48], ribosomal RNA (rRNA) gene identification with RNAmmer [185], transfer RNA (tRNA) genes identification with tRNAscan-SE [184], and two types of homology searches using BLASTX against UniRef100 and HMMER [198] against Pfam [199] and TIGRFAM [200].

Computational resources

All analyses were performed using the CloVR version beta-0.5 (build clovr-standard-2011-12-04-22-00-04). The local computer used for evaluation was a 64-bit quad core (Intel Xeon E5520 2.27 GHz CPU) with 6 gigabytes of RAM. For local execution, CloVR was run using VMware Player v. 2.0.5 build-109488 [55] configured to use a single CPU core and 2012 MB of memory. Amazon EC2 provides numerous instance types with varying CPU speeds, available RAM and storage [61]. Previous work in [121] showed the choice of c1.xlarge to be most cost efficient amongst the choices for applications such as BLAST. The c1.xlarge instances provide 8 virtual CPU cores, 8GB RAM per instance, and 400GB of local temporary disk storage. In this study, each pipeline was run on a separate cluster of instances within the cloud consisting of one master node and zero or more worker nodes. All worker node instances utilized c1.xlarge instances, which at the time of preparing this manuscript were priced at $0.68 per CPU hour (CPU hr). All master nodes utilized c1.xlarge instances except for Illumina assembly with CloVR-Microbe. Assembly of Illumina sequence data required nodes with RAM in excess of the c1.xlarge instance capacity. For single-end Illumina runs, a m2.2xlarge instance ($1.00/CPU hr) was used providing 17.1GB RAM, while for the paired-end Illumina run an m2.xlarge master node ($0.50/CPU hr) was used providing 34.2GB RAM. Associated pipeline costs on Amazon EC2 were calculated using cluster performance charts, visualized with the Ganglia tool (http://ganglia.sourceforge.net/), which describe the number of instances utilized in each cluster over time. Pipeline runtimes were obtained from the Ergatis workflow system.

Spot market bid-price simulations

To simulate runtime distributions within the Amazon EC2 spot market, we first collected corresponding hourly spot prices for the c1.xlarge instance type from October 20, 2010 to January 24, 2011. Assuming a hypothetical pipeline CPU hour requirement of 120 hours a range of bid prices ($0.27/CPU hr to $0.80/CPU hr), we simulated the actual (wall-clock) runtime of a pipeline from random starting points in the collected spot market price data. Given a bid price β and a CPU hr requirement γ, 500 random starting points were picked between 10-20-2010 to 01-24-2011, and the runtime was calculated assuming no processes were running whenever the spot price was above the bid price β. For example, if the bid price was constantly greater than or equal to the spot price for a particular pipeline, then the actual runtime would be γ, because the requested price was always met. Alternatively, if the bid price fell below the spot price for a single hour, then no work was done in that hour and the total actual runtime would be $\gamma+1$. In these simulations, if a simulated pipeline extended beyond 01-24-2011, it immediately continued from the beginning of the time-series. Runtime distributions were visualized in R (http://www.r-project.org/).

Table 8-2 Datasets used for CloVR protocol benchmarking

Dataset	Data type	Sequencing platform	Library type[1]	Total reads	Units[2]	Avg. read length [bp]	Size [MB]	Samples
a) CloVR-Search								
Infant gut WGS	WGS	454 Titanium	SE	595816	0.6 plates	244	145.3	12
Metahit 500K	WGS	Illumina GAII	-	500000[3]	1/80 channels	75	37.5	1
b) CloVR-16S								
Humanized mice	Amplicon	454 FLX	SE	530030	1.1 plates	232	122.5	215
Infant gut 16S	Amplicon	454 FLX	SE	399127	0.8 plates	179	95.1	63
Human vagina	Amplicon	454 FLX	SE	901264	1.8 plates	223	200.6	392
c) CloVR-Metagenomics								
Obese twins	WGS	454 FLX	SE	999990	2 plates	219	218.9	18
Infant gut WGS	WGS	454 Titanium	SE	595816	0.6 plates	244	145.3	12
Nine biomes	WGS	454 FLX	SE	5785371	11.6 plates	109	631.2	45
d) CloVR-Microbe								
Escherichia coli 250K	WGS	454 Titanium	PE (3 kbp)	250000[3]	0.25 plates	279	69.7	1
Escherichia coli 500K	WGS	454 Titanium	PE (8 kbp	500000[3]	0.5 plates	367	183.9	1
Escherichia coli 8M SE	WGS	Illumina GAII	SE	8000000[3]	0.2 channels	36	288	1
Escherichia coli 8M PE	WGS	Illumina GAII	PE (3 kbp)	8000000[3]	0.2 channels	49	392	1
Acinetobacter baylyi 250K	WGS	454 Titanium	PE (8 kbp)	250000[3]	0.25 plates	338	84.7	1

[1] Abbreviations: bp, basepairs; SE, single-end; PE, paired-end (in parentheses: insert size); WGS, whole-genome shotgun
[2] References for unit sizes: Roche/454 GS FLX, 500K reads per plate (two half plates); Roche/454 GS FLX Titanium, 1M reads per plate (two half plates); Illumina GAII, 40M reads per channel (eight channels per flowcell).
[3] Trimmed datasets.

Results

Computational requirements of microbial genomics applications

Representative datasets from two next-generation sequencing platforms, the Roche/454 GS (FLX and FLX Titanium) and Illumina GAIIx (Table 8-2), were processed with several pipelines (CloVR-16S, -Microbe, -Metagenomics, and -Search) to determine processing requirements for typical microbial genome projects (Table 8-2). The datasets evaluated include typical outputs of single or multiple sequencing reactions of the

Roche/454 and Illumina platforms or fractions thereof (unpublished sequence data from the Institute for Genome Sciences), as well as published data from sequencing projects that received wide recognition in the microbial genomics field [201,202,203,204,205,206,207].

CloVR-16S was always run on a single-CPU, on both a local desktop and one CPU of a c1.xlarge Amazon EC2 instance, and finished in less than 14 hours (see Supplementary Table S1 for a comparison of local and EC2-based CloVR-16S runs). Processed datasets included up to ~900K Roche/454 GS FLX reads from ~400 samples as well as up to ~40K Sanger reads from ~120 samples. The 530K humanized mouse gut sequences from 215 different samples [206], for example, which contain a total of 14,363 operational taxonomic units (OTUs), were processed in about the same time as the 901K human vaginal sequences from 392 samples [205], which only contain 4,967 OTUs.

Table 8-3 Cost and runtime parameters of CloVR pipeline runs on example datasets

Dataset	**Upload time**	**Pipeline runtime**	**Down-load time**	**Total cost[1]**	**Max. VMs[2]**	**Max. CPUs**	**QC**		
a) CloVR-Search, BLASTN against RefSeq							**RefSeq matches**		
Infant gut WGS	3 min	1 hr 26 min	20 min	\$11	8	64	34.3		
Metahit 500K	11 min	10 hr 42 min	17 min	\$151	20	160	3.2 %		
b) CloVR-16S							**OTUs**		
Humanized mice	42 min	1 hr 30 min	12 min	\$3	1	8	14363		
Infant gut 16S	3 min	42 min	10 min	\$1	1	8	3447		
Human vagina	1 hr 17 min	1 hr 51 min	14 min	\$3	1	8	4967		
c) CloVR-Metagenomics[3]							**nr reads**	**RefSeq matches**	**COG matches**
Obese twins	8 min	2 hr 25 min	24 min	\$30	20	160	93.6%	33.3 %	29.6 %
Infant gut WGS	7 min	2 hr 17 min	29 min	\$24	15	120	98.2%	35.2 %[4]	33.5 %
Nine biomes	15 min	5 hr 35 min	39 min	\$56	20	160	89.9%	9.3 %	5.6 %
d) CloVR-Microbe							**Scaffold/ Contigs**	**N50**	**CDS[5]**
Escherichia coli 250K	24 min	16 hr 21 min	52 min	\$55	14	112	8 / 414	25 kbp	6313
Escherichia coli 500K	20 min	20 hr 23 min	50 min	\$60	15	120	37 / 141	183 kbp	5827
Escherichia coli 8M SE	12 min	15 hr 44 min	37 min	\$62	15	120	553 / 553	17 kbp	4803
Escherichia coli 8M PE	16 min	15 hr 2 min	44 min	\$44	15	120	481 / 481	18 kbp	4464
Acinetobacter baylyi 250K	20 min	9 hr 46 min	37 min	\$39	15	120	4 / 38	262 kbp	3417

[1] Rounded to the next full dollar.

[2] VM instances are linked together as a cluster for parallel processing on the cloud. The number of instances in a cluster can change during pipeline execution. The maximum utilized is reported

[3] CDS, coding sequences

CloVR-Microbe and CloVR-Metagenomics analyses of all datasets were performed exclusively on Amazon EC2 where all runs finished in under 24 hours (Table 8-3). Dataset sizes for CloVR-Metagenomics ranged from ~600K reads (454 FLX Titanium), corresponding to 1.2 full sequencing plates, to 5.8M reads (454 FLX), corresponding to

11.6 full sequencing plates, all of which were processed in less than six hours on Amazon EC2. Additional time due to upload of input and download of output was consistently less than one hour. Input data sizes for CloVR-Microbe were representative of typical microbial genome project work loads and included sequence read numbers corresponding to a quarter (250K) or a half (500K) plate of 454 FLX Titanium as well a 1/5 (8M reads) of an Illumina GAIIx lane (single read and paired-end read libraries). Pipeline outputs were found to be in agreement with results from previously analysis on these projects in terms of number of detected OTUs, relative OTU compositions, principal coordinate analysis plots of OTU assignments (CloVR-16S), number of functionally and assigned reads (CloVR-Metagenomics) and number and lengths of contigs, number and functional annotation of genes (CloVR-Microbe). Cluster sizes on Amazon EC2 were configured automatically based on the pipeline requirements as estimated using input data sizes. The estimates for our evaluation ranged from 14 to 20 machine instances, comprising up to 160 virtual CPUs (Table 8-3).

BLASTN searches of metagenomic WGS sequence data against the NCBI RefSeq collection were performed on Amazon EC2 using CloVR-Search. Using the multi-CPU support of Amazon EC2, ~600K reads of 454 FLX Titanium, corresponding to 0.6 full plates could be processed in less than two hours (64 CPUs maximum usage). In comparison, the BLASTN search of a similar number (500K) of shorter (75bp) Illumina GAIIx reads against RefSeq, which produced about the same percentage of matches (3.2% vs 3.4%) took about 10 times longer to complete (~11 hours), using 2.5 times the amount of CPUs (160 CPUs maximum usage). For the Illumina GAIIx platform, 500K reads correspond to only 1/6 of the average sequencing output of a single channel (eight lanes per flow cell).

Real dollar values of microbial sequence analysis applications

Real dollar costs were calculated for all microbial sequence analyses performed with the CloVR pipelines (Table 8-3), in order to provide guidelines for costs associated with microbial genomics projects. The costs include overhead introduced by the CloVR VM to make use of the cloud environment, including time for data upload and download and to prepare input and output data. Table 8-3 also provides example network transfer times for upload to and download from the cloud, although such times can vary substantially based on the network environment. Several large datasets that are used as reference data for the CloVR pipelines, e.g. 3.4GB of compressed reference data for CloVR-Microbe comprising the UniRef100 protein database, were hosted permanently on the Amazon Simple Storage Service [173], which provides data storage inside the cloud network and reduced the need for data transfer over the Internet when executing in the cloud. During the pipeline execution, the free ephemeral instance storage was used as temporary storage and all output data was compressed and downloaded to the local desktop upon pipeline completion. All CloVR VMs on the cloud are shut down automatically upon pipeline completion, in order to avoid charges for idle instances and persisting storage at Amazon EC2.

Based on the CloVR runs on EC2, the cost of all 16S rRNA community analyses was less than $10. For the sequence data generated with the short amplicon 454 sequencing protocol, costs ranged from less than $1 to $2.72. Since all pipelines finished in less than two hours, the costs associated with Amazon EC2 charges for instances being active during upload and download times constitute a significant fraction of the total cost (Table 8-3), but are nominally small at < 1 c1.xlarge instance hour ($0.68).

All CloVR-Metagenomics and CloVR-Microbe runs were completed at costs of less than $100. Sequence analyses with the CloVR-Metagenomics pipeline had an associated cost of between ~$23 and ~$56; CloVR-Microbe runs had costs of between ~$39 and ~$62.

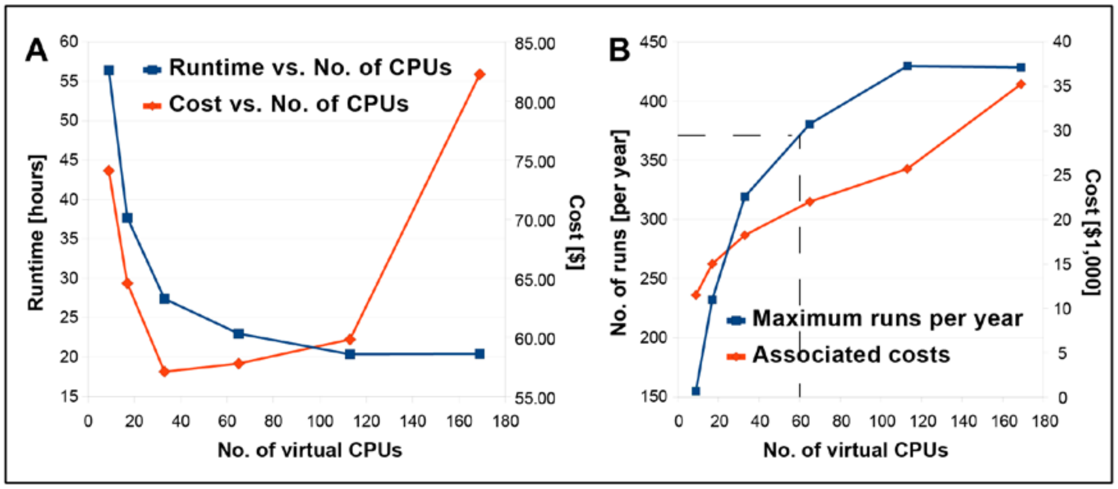


Figure 8.1 - Cost and performance of CloVR-Microbe on varying size compute clusters
A) Steps of the CloVR-Microbe pipeline can be executed in parallel to improve performance as shown by plotting pipeline runtimes (blue) and associated costs (red) against the number of CPUs used to perform the analysis on Amazon EC2. B) Using this data, the theoretical maximum throughput per year (blue) as well as associated costs (red) of analysis using CloVR-Microbe can be extrapolated. As an example, the output of a single 454 FLX Titanium machine, run every other day with two single microbial genomes per sequencing plate (365 total runs), can be processed on EC2 using 60 CPUs (or eight c1.xlarge instances) for less than $25,000, as indicated by the dashed line.

Capacity and optimization of processing pipelines

The multi-CPU capabilities of the cloud allow for decreased runtime for pipelines involving analysis steps than can be parallelized, e.g. the BLASTX sequence comparisons of the CloVR-Microbe pipeline. At the same time, partitioning of data into multiple parallel processes using the CloVR VM architecture, involves additional copying of reference data, increases the amount of data transfer between machines and incurs additional processing overhead. Also, implementation of the protocol may prevent full utilization of a cluster or limit the partitioning of data for parallel processing. To determine differences in the CloVR-Microbe runtimes and associated costs depending on the number of CPUs used, the same dataset of 500K 454 FLX Titanium reads, corresponding to one full plate of 8kbp paired-end sequences, was run with different cluster sizes on Amazon EC2 (Figure 8.1, Supplementary Table). Based on this example, the lowest runtimes and costs achieved fell between 72 CPUs (23 hours, $58) and 120 CPUs (20 hours, $60). These numbers represent a runtime and cost improvement of up to 36 hours and $16 compared to the run with the smallest cluster size (16 CPUs: 56 hours, $74). A further increase of the cluster size to 172 CPUs did not result in a runtime improvement but resulted in increased cost ($82) due to under-utilized instances. A local run on a single-CPU machine was canceled after 14 days.

Table 8-4 Variations in cost and runtime parameters of different CloVR pipeline runs on the same metagenomics WGS dataset (Infant gut).

Protocol	Upload time	Pipeline runtime	Down-load time	Total cost[1]	Max. VMs[2]	Max. CPUs	QC		
							Contigs	**N50**	**CDS**
CloVR-Microbe	20 min	23 hr 45 min	1 hr 23 min	$48	8	64	2056	2524	7983
							RefSeq matches	**COG matches**	
CloVR-Metagenomics, BLASTX	7 min	2 hr 17 min	29 min	$24	15	120	35.2 %	33.5 %	
							RefSeq matches	**COG matches**	
CloVR-Metagenomics, Metagene/ BLASTP	7 min	2 hr 19 min	21 min	$31	15	120	35.2 %	28.7%	
							RefSeq matches		
CloVR-Search, BLASTN (RefSeq)	3 min	1 hr 26 min	20 min	$10.88	8	64	34.3 %		

[1] Rounded to the next full dollar.

[2] VM instances are linked together as a cluster for parallel processing on the cloud. The number of instances in a cluster can change during pipeline execution. The maximum utilized is reported.

Three different analysis protocols (CloVR-Microbe, CloVR-Metagenomics and CloVR-Search) were evaluated for their impact on runtime and cost for metagenomics WGS analysis (Table 8-4). All analyses were run on the same Infant Gut Microbiome WGS input dataset [203], corresponding to 0.6 full plates of single-end 454 FLX Titanium sequences. CloVR-Microbe pipeline was included to provide genome assembly of metagenomics data in the comparison. We note that the Glimmer gene finding tool [48], which is part of the CloVR-Microbe protocol, was optimized for large contiguous assembled sequence data and is known to perform less optimally on short sequence fragments that contain a large number of truncated coding sequences [196]. Two variations were used of the CloVR-Metagenomics protocol: i) The BLASTX protocol searches each nucleotide sequence read against the COG database [187] by translating all six frames into protein sequences, whereas ii) the Metagene/BLASTP protocol first runs a gene prediction with Metagene [196], before translating the identified genes into protein sequences and running a BLASTP search. A BLASTN comparison of each read against NCBI's RefSeq database performed with CloVR-Search was used as the most basic analysis protocol.

Compared to the CloVR-Microbe protocol, both CloVR-Metagenomics protocols executed about ten times faster (~2.5 hours compared to 24 hours) at about 50% of the cost ($23 / $31 compared to $48) (Table 8-4). Although the BLASTX-based and Metagene-based CloVR-Metagenomics protocols finished in about the same time, the BLASTX search against the COG database identified more matches (33% compared to 29%) and the total cost of the pipeline run was lower. The BLASTN search against RefSeq alone finished in about 1.5 hours for $11.

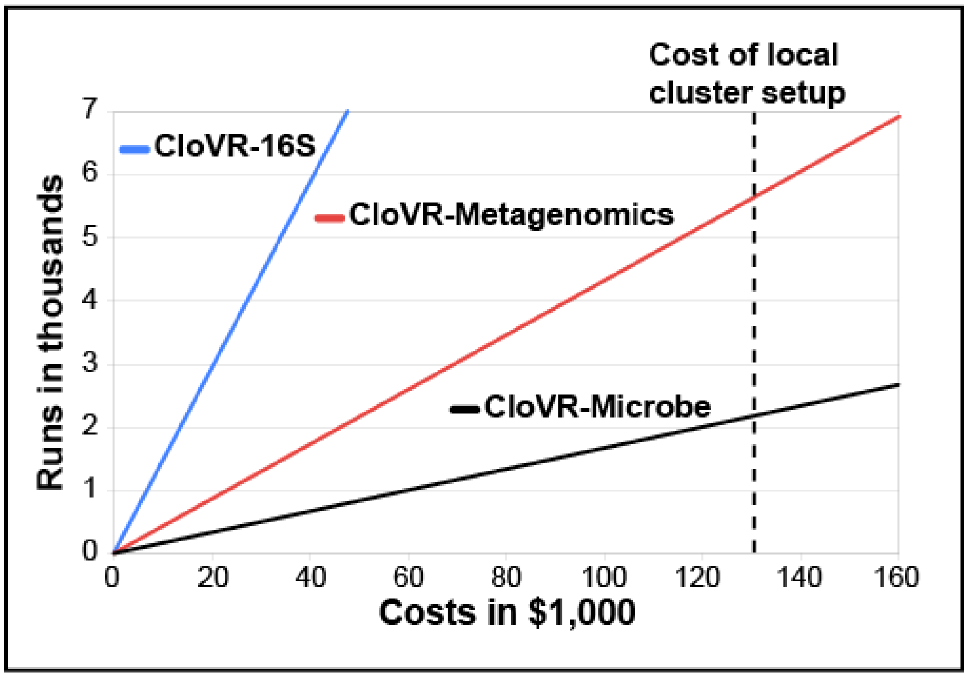


Figure 8.2 – Costs and throughput of CloVR protocols.
Costs for single CloVR-16S (blue), CloVR-Metagenomics (red) and CloVR-Microbe (black) runs of comparable datasets (~500K 454 FLX or FLX Titanium reads) on Amazon EC2 were extrapolated to calculate the number of runs that are obtainable for a given dollar value. The dashed line represent the average annual cost to set up and maintain a local cluster of 240 CPUs for a three years from Dudley et al. [120].

To estimate the amount of sequence analysis that is affordable for a given dollar value, the number of analysis runs using three different protocols (CloVR-16S, CloVR-Metagenomics and CloVR-Microbe) was plotted against the corresponding cost, using results from Table 8-3 (Figure 8.2). These costs were compared to the $130K estimated as average annual cost to set up and maintain a local cluster of 240 CPUs for three years as described in [120]. Using these estimates, 19,117 runs of CloVR-16S; 5,623 runs of CloVR-Metagenomics; and 2,172 runs of CloVR-Microbe can be processed each year on Amazon EC2, before the costs of a local cluster are more economical. For single whole-genome microbial analysis, with a theoretical annual output of 730 datasets per 454 FLX Titanium sequencer (one full plate per day, two single-genome datasets per plate), up to three sequencing machines can be supported using Amazon EC2 at current prices using our benchmark protocols before the estimated cost of a local cluster is reached.

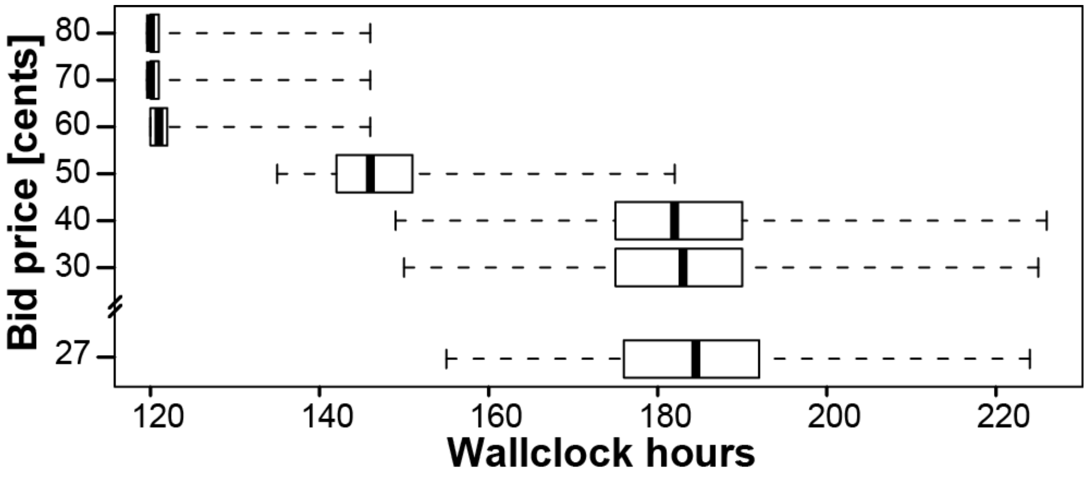


Figure 8.3 – Predicted runtimes for varying bid prices in the Amazon spot market for compute
An analysis requiring 120 CPU hours was used an example to estimate the expected completion time for different bid prices

Realizing cost savings using excess capacity in the Amazon EC2 spot market

The Amazon EC2 spot market allows customers to place bids on unused cloud resources and utilize instances for as long as the bid exceeds the current spot price (http://aws.amazon.com/ec2/spot-instances/). During periods of weak demand, the spot market provides the ability to utilize excess resources at a discounted price. Over the period of the past year, the spot market price for the c1.xlarge instance averaged $0.26 compared to an on-demand price of $0.68. This variable pricing is well-suited to processing needs that are not time critical since analysis will only proceed when the provided bid price is above the current market price for the resource. This market model also provides the ability to predict the expected completion time of a pipeline for a particular bid price using historical pricing data. The expected completion times were estimated for bids of $0.27 to $0.80 using a hypothetical analysis requiring 120 c1.xlarge instance hours (960 c1.xlarge CPU hours) for completion (Figure 8.3). The expected completion time was predicted for each bid price using the recorded pricing data for the past month. Based on this model, at a bid price of $0.68 the analysis was expected to execute in ~120 hours, while never taking longer than ~145 hours. By comparison, a $0.27 bid will not be fulfilled during times of peak demand when the market price rises above the bid, but during other times the user can realize a savings of 40%. A bid of $0.27 is estimated to take on ~185 hours on average, 50% slower on average than using the full on-demand price, but may complete in as little as ~155 hours (29% slower) or as many as ~225 hours (87% slower). These estimated runtimes are meant to be illustrative for bids on a single instance in the spot market. Since actual pipelines can run in parallel across multiple instances, the actual runtimes can be reduced compared to what is shown. CloVR provides the ability to use the spot market for allocating instances during pipeline execution. Bid prices are set in a configuration file.

Discussion

In this study, we explore the costs and resources required for microbial sequence analysis using pre-packaged protocols in CloVR (Chapter 7). The automated pipelines in CloVR were selected with the intention of packaging existing community-supported analysis protocols. The protocol, CloVR-Microbe, combines a a sequence assembly step with functional annotation from the IGS Annotation Engine [109]. With the support of a large local grid cluster, the IGS Annotation Engine was designed to be thorough for genome annotation but not optimized for speed or efficient CPU usage, and many alternative genome annotation protocols exist, e.g. RAST [106], DIYA [112]. To our knowledge, CloVR-Microbe represents the first automated pipeline that combines sequence assembly and annotation in a single automated pipeline.

The CloVR-16S pipeline was designed to combine components of several widely used 16S rRNA sequence analysis protocols, without making the entire workflow computationally too complex to process even large sequence datasets (>200 samples, >500K sequences). The current implementation of CloVR-16S supports a distance matrix-based operational taxonomic unit (OTU) assignment and α-diversity analysis with Mothur [110], direct taxonomic classifications of sequence reads with the RDP classifier tool [194] and microbial community analysis with the QIIME tool, which has a strong focus on phylogenetic distance-based β-diversity analysis [111]. A critical component of CloVR-16S in its current implementation is the threshold of 50,000 non-redundant sequences above which the Mothur component with its computationally expensive distance matrix calculation is not performed.

Metagenomics projects are usually designed to generate the most sequence data per invested dollar and, thus, often involve large-scale next-generation sequencing data. Due to the resulting dataset sizes, metagenomics analysis protocols therefore often rely on the direct classification of individual sequence reads by BLAST (e.g. MG-RAST [208]), instead of involving sequence assembly steps, which are even more computationally demanding and impractical for metagenomes. Similarly, the CloVR-Metagenomics pipeline was designed to examine and compare taxonomic and functional microbial community compositions within and between metagenomic samples using two BLAST searches against a bacterial genome database (BLASTN against NCBI's RefSeq) and against a functionally annotated protein database (BLASTP against NCBI's COG). The CloVR-Search pipeline was designed to provide support for large-scale BLAST comparisons using multiple computers offered by the cloud. As an alternative baseline metagenomics read classification, a direct BLASTN comparison of each sequence read against the NCBI RefSeq nucleotide database with CloVR-Search was shown to provide further runtime improvements compared to CloVR-Metagenomics pipeline although without producing the visual and statistical evaluations of the results that are generated by the CloVR-Metagenomics pipeline.

We decided to use the popular Amazon EC2 cloud as a model for evaluating analysis costs. Importantly for budgeting, the costs at Amazon EC2 are transparent and directly obtainable for any workload, allowing for attaching real dollar costs to computational analyses. Our results show that bioinformatics support for microbial genomics can be provided at a competitive price, provided analysis protocols are chosen carefully. In addition, as many analysis needs are not time-critical and can wait for off peak hours, so a bidding market for compute, such as the Amazon EC2 Spot Market, provides an intriguing model for further cost savings. Since these costs depend substantially on the choice of analysis protocol, the results in this study can also be used as benchmarks for comparing costs and resources of other analysis protocols.

The Amazon EC2 cloud can also serve as a model to evaluate the computational infrastructure needed to perform common microbial genomics applications. Our evaluation of the CloVR protocols shows that typical workloads of small to midsize sequencing facilities are most economically processed either locally, on a single desktop machine (CloVR-16S), or online using the Amazon EC2 cloud (CloVR-Metagenomics, -Microbe, -Search). The computational resources deployed on EC2 for the evaluation were modest, utilizing no more than 20 virtual machine instances, eight CPUs per instance and 160 CPUs at a maximum. As multi-core CPUs are increasingly becoming accessible on the desktop computer market, the ability to process larger data on desktops is likely to increase in the future.

While there has been tremendous interest in using clouds, there has also been concern over potentially high costs of clouds for big data applications [119], exceeding millions of dollars for searches of terabytes of short sequencing reads [121]. Yet, these high dollar values also demonstrate that a particular methodology is computationally demanding [12,19]. A large, inefficient computation is likely to be expensive to run on any computing resource, whether locally built or remotely hosted. As the largest commercial computing provider, Amazon EC2, realizes economies of scale enabling low capital and operating costs [209], which are likely to be difficult to match in smaller settings. Despite attempts at estimating comparable local cluster costs [120], modeling infrastructure costs in a small-scale research setting is fraught with problems as it is easy to miss certain operating costs while highlighting capital costs, precisely because continued operation of the resource usually does not rely on an accurate modeling of cost. Adding to the difficulty is that capital and operating costs are often shared between institutional and outside funding sources and these contributions are always itemized. Private clusters will prove more economical only in environments that can ensure high utilization rates.

This case study and the architecture described in Chapter 7 demonstrates the power of packaging pipelines into a single, portable automated framework. With the CloVR virtual machine, we can easily compare the performance or costs of protocols between platforms. CloVR is specifically designed to avoid vender-lockin and can immediately utilize new emerging cloud computing platforms, including those free for researchers

[62,63,158]. This portability provides users the flexibility to move their analysis to the resource where it is best suited based on the costs, reliability, availability, or size of the computing resource.

Chapter 9 - Discussion and conclusions

High-throughput sequencing has introduced analysis bottlenecks with many contributing factors, including computational complexity of methods, poor automation or usability of analysis software, and under-utilization of computational resources. The contributions of this dissertation improve analysis throughput by addressing specific components of this bioinformatics bottleneck. Mugsy, Para-Mugsy, Mugsy-Annotator improve computational efficiency for whole genome alignment and annotation and CloVR improves accessibility and throughput of analysis pipelines by supporting both multi-core personal computers and cloud computing resources for automated processing.

In this concluding chapter, I summarize and discuss each of the contributions of this dissertation, highlight recent applications of the work, and note areas for future development.

Novel methods and software solutions for the bioinformatics bottleneck

Efficient multiple alignment of closely related genomes

In Chapter 4, we describe Mugsy, a new tool and novel methodology for efficient, reference-independent multiple genome alignment. The primary advantage of Mugsy over similar tools is speed. Mugsy was the fastest tool evaluated for the alignment of four assembled human chromosomes, completing in less than one hour provided a library of pairwise alignments. Mugsy was one of only two tools that completed either alignments of four human chromosomes or 57 *E. coli* genomes in less than two days of processing time on a single CPU.

Importantly, Mugsy can align mixtures of complete and draft genomes making this tool particularly well-suited for use with high-throughput sequencing technologies, including 454 [8] and Illumina [9], where a majority of newly sequenced genomes are draft genomes represented by multiple contigs after assembly. Mugsy can identify sequence conservation and variation in any subset of input genomes.

As desktop computers are now commonly available with multiple CPUs, parallel processing using multiple CPUs enables faster runtimes and increases analysis throughput. In Chapter 5, we describe Para-Mugsy for distributed whole genome multiple alignment. This work enables faster, larger scale comparisons than what is currently feasible with other whole genome alignment tools.

A method for efficient comparison and improvement of pan-genome annotation

In Chapter 6, we introduced a new tool, Mugsy-Annotator, that implements a novel algorithm for identifying orthologs and evaluating annotation quality in a pan-genome. Mugsy-Annotator is computationally efficient compared to BLAST-based approaches for classification of orthologs. Also, by using whole genome alignment, Mugsy-Annotator incorporates synteny and genome context as additional evidence of orthology. Our method for identifying orthologs is also robust to certain types of annotation errors, such as missing annotations or incorrect reading frames. Since our method relies on accurate DNA alignment, it is most useful for closely related genomes that share a high percentage of identical DNA, such as isolates from the same or closely related species.

Structural inconsistencies, especially at translation initiation sites, are prevalent in publically available gene predictions, even for intra-species comparisons between nearly identical sequences. Our case study of *Neisseria meningiditis* indicates that a majority of the identified differences between annotations are a consequence of bioinformatics methods rather than true biological differences, especially in regions with poor sequencing coverage. Our results add to previous studies, which have noted errors in intra-species genome annotations [53,95] and caution against simple comparisons of genome annotations without exhaustive follow-up analysis [54,144]. Mugsy-Annotator provides an efficient tool for such an exhaustive comparative analysis of gene structures.

As genome sequencing throughput has increased, expert curation of genomes is no longer practical and has been replaced almost exclusively by high throughput, automated annotation methods. Yet existing automated methods do not match the accuracy of manual review by experts [99]. Mugsy-Annotator enables a semi-automated solution to this problem by highlighting likely annotation problem areas for focused review by human curators. Importantly, Mugsy-Annotator analyzes a pan-genome, allowing for re-annotation efforts to apply across a set of genomes simultaneously, rather than one genome at a time, significantly increasing analysis throughput.

A portable platform for automated and high-throughput sequence analysis

By providing a portable virtual machine, CloVR provides fully automated pipelines that are designed to alleviate the user from installation and configuration of bioinformatics tools or cloud computing client software. Automated systems not only save the time of the operators, they aid in reproducibly of results, which is a cornerstone of science but is notably lacking in genome analysis [18]. CloVR enables push-button reproducibility of results.

By avoiding network bottlenecks during processing, CloVR is also highly scalable supporting distributed computation on cloud computing platforms without any special storage or hardware requirements. The portability of CloVR enables comparison of analysis protocols within and across computing platforms. The ability to directly compare protocols on the basis of resource requirements, runtimes, and costs is of tremendous value for researchers utilizing genomics data.

We found that for whole genome sequencing of microbes, commercial cloud computing providers are a cost effective analysis platform, provided reasonable analysis protocols, such as those provided by CloVR. A processing throughput of ~2100 single bacterial genomes per year using the assembly and annotation protocol (CloVR-Microbe) is needed before use of a dedicated local compute cluster is more economical. A single execution of this assembly and annotation takes days to execute on a local computer with four processing cores. As next-generation sequencing has created smaller sequencing operations, many applications of sequencing will be relatively low throughput, occasionally generating data for a particular study. CloVR is perfectly suited for such applications, simplifying sequence analysis and making seemless use of cloud computing platforms to improve processing runtimes.

Highlighted applications

The contributions of this dissertation have wide applications in biological research, especially studies on the process of evolution and in the biomedical research domain in understanding the genetic basis of disease and pathogenesis. In Chapters 3-5, three novel algorithms (Mugsy, Para-Mugsy, and Mugsy-Annotator) are demonstrated to improve computational efficiency for whole genome comparison, allowing for more comparisons at higher throughput. In this section, we highlight two published studies that utilized these contributions and illustrate downstream applications and demonstrate impact.

Construction of high-resolution phylogenies using whole genome multiple alignment

Multiple locus sequence typing (MLST) utilizes 7-10 genes for alignment and typing species phylogenies in bacteria [210,211]. Yet, the use of so few sequences limits the resolution of such studies and ignores the whole genome data that is readily available with high-throughput sequencing. In [212], Mugsy was used to generate a high-resolution species tree providing new insights into the relatedness of pathotypes for the clinically relevant and enterotoxigenic isolates of *Escherichia coli*. Mugsy allowed for calculation of a conserved genetic core across the species using whole genome multiple alignment. This technique, which used whole genome multiple alignment in combination with automated alignment trimming [213] and efficient phylogenetic tree construction [214], provides a gene-independent view of species structure and evolution between closely related organisms.

As whole genome data for bacteria can be readily obtained with high-throughput sequencing methods, the ability to efficiently compare whole genome data provides a relatively fast platform for construction of high-resolution phylogenies. As sequence costs have also decreased, a typing approach based on genome alignment may soon rival the time and effort spent on PCR amplification and Sanger sequencing of 7 to 10 MLST gene fragments, which currently provides the standard for such phylogenetic studies. Our work on Mugsy enabled this high-resolution study and demonstrates a general framework for high-throughput, high-resolution strain identification and diagnostics.

High-throughput pan-genome annotation using whole genome multiple alignment

Genome annotation is critical for attaching biological meaning to genetic elements. While manual curation efforts were historically dedicated to annotation of single genomes, such an approach is no longer feasible as hundreds of new genomes are being generated for some species. Yet, fully automated annotation solutions can not currently produce the consistent quality of data that is generated by expert manual review. Alternative approaches are needed that incorporate all available pan-genomic data for a species while enabling expert review without loss of annotation quality.

In one such approach, Mugsy-Annotator, in conjunction with a comparative visualization tool Sybil [215], was used in to re-annotate isolates of the pathogenic bacterium *Neisseria meningitidis* (*Nmen*) as part of the study published as Budroni et al [140]. *Nmen* is prevalent in human populations [216], occasionally causing very severe meningococcal meningitis and septicemia. While possible virulence factors have been identified or proposed [217,218,219], the limited number of sequenced isolates had previously prevented a population study at the whole-genome. To enable a population-wide study, Mugsy-Annotator was used as part of the analysis and annotation of 15 previously unpublished and 5 publicly available Nmen genomes isolated from five continents. The identified ortholog families were used to revise and standardize the annotation across all the genomes, improving annotation consistency prior to examination of gene families. This re-annotation effort was critical to ensuring that the composition of gene families could be queried and searched for isolate and clade specific gene families. The study determined restriction modification systems are unique to distinct phylogenetic clades, resulting in a differential barrier to genetic transfer and accounting for the observed structuring of the population of *Nmen* genotypes. The contribution of Mugsy-Annotator accelerated this effort.

Discussion

As biologists continue to explore the rich genetic diversity of the biosphere, thousands of individual genomes will soon be available for some species and the ability to read genetic information is outpacing the speed at which we can analyze the data for meaningful relationships. In such an environment, the need to efficiently compare sequences is fundamental. In addition, comparative analysis needs to be performed in the context of populations, to avoid bias in selection of reference genomes. The contributions of Mugsy and Mugsy-Annotator accelerate comparative genomics studies by providing efficient methods for comparisons of pan-genomic data.

While current generation technologies boast of hundreds of gigabytes of data [60], much of this data is an artifact of the sequencing technology (massively redundant short reads) rather than representative of the physical specimen. It is hard to imagine these massively redundant sequencing technologies will prove efficient in the future. Rather, the ability to sequence individual DNA molecules [10] may provide a more complete view the sequence at decreased data volume per study, reducing the need for extensive pre-processing prior to analysis. After all, the genome of a bacterium is merely 2-5 megabytes on average and the human genome is ~3 gigabtyes per DNA molecule.

The ability to read whole DNA molecules will foster comparisons of multiple, nearly identical DNA molecules, with widespread applications for personalized genomics, including comparing DNA in families [220], phased chromosomes [221], somatic and germ lines or diseased tissues, such as cancers [222]. The methodologies used in Mugsy are perfectly suited to perform these comparisons on a whole genome level. By avoiding selection bias of a reference genome, Mugsy can multiply align populations of genomes and is well suited for the detection of rare variants, which have been considered as a possible major contributor to common diseases [223].

Cloud computing delivers on the promise of resources on-demand. But, a major impediment to use this remote resource is data transfer over a wide area network. As the Internet will likely remain slow and crowded, and specialty high-speed nationwide networks [224] available only to select institutions and applications, the data transfer problem is likely to remain in the forefront for genomics applications, especially now that sequence generation is decentralized. Next generation sequencing technologies that generate lower data volumes but otherwise more accurate and complete data will make it easier to distribute over computer networks for analysis and dissemination. Reference based compression allows for representing this data in even fewer bytes [225]. To aid with large data studies, Amazon EC2 currently offers the ability to send and receive hard drives of data (multiple terabytes in size) by the mail, although reliance on such schemes will impede rather than increase access to cloud computing resource. To achieve best possible throughput, CloVR utilizes open source transfer utilities designed for maximizing throughput on slow networks.

At the same time, in an environment where entire populations are sequenced, reference databases will continue to grow large, especially if expansive meta-data is captured [178]. Such reference sequence data are well-suited for co-location with computing resources avoiding network transfers [114]. CloVR as a portable VM is perfectly suited to an environment where analysis tools are brought to large data sets for computation.

Multi-core personal computers are readily available with the number of processing units expected to increase. The architecture of CloVR allows for utilization of both local computer and remote cloud resources directly from a users' desktop. We believe this is a preferred architecture for high-throughput sequence analysis and consistent with the decentralization trend of genome sequencing. The CloVR architecture can avoid network data transfers entirely and ensure data privacy by utilizing local computer resources, while maintaining support for large-scale analysis on distributed computing resources when necessary. The key to success of this architecture is automation. As cloud computing resources are low-level systems, automation is critical for improving analysis throughput and avoiding technical hurdles and troubleshooting that slows research projects and hinders discoveries. CloVR eliminates technical challenges in using the cloud, providing a highly automated system for high-throughput research.

To enable full use of larger data sets and exploration of populations of genomes, continued development of efficient algorithms and use of distributed computing resources are needed. The multiple alignment of several hundreds or more relatively small bacterial genomes remains a computational challenge and may limit the use of the growing amounts of whole genome data by biologists. Often, the only practical alternative is mapping data to a pre-defined reference genome. Distributed computing solutions are needed and Para-Mugsy provides an advance in this direction but more work is needed to improve the robustness of the software on even larger data sets. As

Para-Mugsy is implemented to run on the CloVR virtual machine, future work is ready to take advantage of local multi-core desktop computers and cloud computing platforms.

Concluding remarks

Biological insights and practical applications of genomics data are driven by the ability to compare sequences. The computational demands for these comparisons continue to grow as sequence databases increase in size and sequencing technologies improve resolution. As advancements in computer processing speeds are insufficient to address analysis bottlenecks, continued development of efficient algorithms and increased utilization of distributed computing solutions are needed for sequence analysis. The contributions in this dissertation address current bottlenecks, by improving efficiency and throughput in comparisons of populations of genomes (Mugsy, Para-Mugsy, Mugsy-Annotator) and providing an automated and portable platform for distributed computation (CloVR), simplifying sequence analysis for a growing pool of genomics users. The ability to quickly assimilate genetic information across populations stands to accelerate research and clinical applications of genomics.

Appendices

In addition to the contributions in this dissertation, the author also participated in a number of comparative genomics studies during the period of study, providing bioinformatics support and co-authoring findings, including [140,212,226,227,228,229,230,231,232,233,234,235,236,237]. Selected co-authored works with primary contributions from the dissertation author are referenced below.

i) Ergatis: a web interface and scalable software system for bioinformatics workflows [102].

   Author contribution: Formulated the idea, implemented the initial software, and led subsequent development efforts.

ii) Sybil: methods and software for multiple genome comparison and visualization [215].

   Author contribution: Formulated the idea, implemented the initial software, and led subsequent development efforts.

iii) Toward an online repository of Standard Operating Procedures (SOPs) for (meta)-genomic annotation [94].

   Author contribution: First author of the paper. This effort contributed to the launch of an open source journal for genomic standards; Standards in Genomic Sciences [238,239], which is now indexed by Pubmed. The dissertation author currently serves as a section editor.

Bibliography

1. Angiuoli SV, Salzberg SL (2011) Mugsy: fast multiple alignment of closely related whole genomes. Bioinformatics 27: 334-342.
2. Bailey RC (2010) Grand challenge commentary: Informative diagnostics for personalized medicine. Nat Chem Biol 6: 857-859.
3. Green ED, Guyer MS (2011) Charting a course for genomic medicine from base pairs to bedside. Nature 470: 204-213.
4. Guttmacher AE, McGuire AL, Ponder B, Stefansson K (2010) Personalized genomic information: preparing for the future of genetic medicine. Nat Rev Genet 11: 161-165.
5. Tettelin H, Riley D, Cattuto C, Medini D (2008) Comparative genomics: the bacterial pan-genome. Curr Opin Microbiol 11: 472-477.
6. Medini D, Donati C, Tettelin H, Masignani V, Rappuoli R (2005) The microbial pan-genome. Curr Opin Genet Dev 15: 589-594.
7. Li R, Li Y, Zheng H, Luo R, Zhu H, et al. (2010) Building the sequence map of the human pan-genome. Nat Biotechnol 28: 57-63.
8. Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, et al. (2005) Genome sequencing in microfabricated high-density picolitre reactors. Nature 437: 376-380.
9. Shendure J, Mitra RD, Varma C, Church GM (2004) Advanced sequencing technologies: methods and goals. Nat Rev Genet 5: 335-344.
10. Eid J, Fehr A, Gray J, Luong K, Lyle J, et al. (2009) Real-time DNA sequencing from single polymerase molecules. Science 323: 133-138.

11. DNA Sequencing Costs: Data from the NHGRI Large-Scale Genome Sequencing Program Available at: Available at: www.genome.gov/sequencingcosts. Accessed March 10,2011.

12. Schatz MC, Langmead B, Salzberg SL (2010) Cloud computing and the DNA data race. Nat Biotechnol 28: 691-693.

13. Elias I (2006) Settling the intractability of multiple alignment. J Comput Biol 13: 1323-1339.

14. Rusk N (2011) Torrents of sequence. Nature Methods 8: 44.

15. Next Generation Genomics: World Map of High-throughput Sequencers. Available at: http://pathogenomics.bham.ac.uk/hts/

16. Merali Z (2010) Computational science: ...Error. Nature 467: 775-777.

17. Field D, Tiwari B, Booth T, Houten S, Swan D, et al. (2006) Open software for biologists: from famine to feast. Nat Biotechnol 24: 801-803.

18. Mesirov JP (2010) Computer science. Accessible reproducible research. Science 327: 415-416.

19. Schadt EE, Linderman MD, Sorenson J, Lee L, Nolan GP (2010) Computational solutions to large-scale data management and analysis. Nat Rev Genet 11: 647-657.

20. Wu D, Hugenholtz P, Mavromatis K, Pukall R, Dalin E, et al. (2009) A phylogeny-driven genomic encyclopaedia of Bacteria and Archaea. Nature 462: 1056-1060.

21. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW (2011) GenBank. Nucleic Acids Res 39: D32-37.

22. GEBA. Available at: http://www.jgi.doe.gov/programs/GEBA/

23. Liolios K, Mavormatis K, Tavernarakis N, Kyrpides N (2008) The Genomes On Line Database (GOLD) in 2007: status of genomic and metagenomic projects and their associated metadata. Nucleic Acids Research 36: D475-479.

24. Collins FaB, A.D. (March 2007) Mapping the Cancer Genome. Scientific American.

25. Durbin RM, Abecasis GR, Altshuler DL, Auton A, Brooks LD, et al. (2010) A map of human genome variation from population-scale sequencing. Nature 467: 1061-1073.

26. Duncan DE (June 7, 2010) On a Mission to Sequence the Genomes of 100,000 People. New York Times.

27. Tettelin H, Masignani V, Cieslewicz MJ, Donati C, Medini D, et al. (2005) Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: Implications for the microbial "pan-genome". Proc Natl Acad Sci U S A 102: 13950-13955.

28. Rasko DA, Rosovitz MJ, Myers GS, Mongodin EF, Fricke WF, et al. (2008) The pangenome structure of Escherichia coli: comparative genomic analysis of E. coli commensal and pathogenic isolates. J Bacteriol 190: 6881-6893.

29. Mira A, Martin-Cuadrado AB, D'Auria G, Rodriguez-Valera F (2010) The bacterial pan-genome:a new paradigm in microbiology. Int Microbiol 13: 45-57.

30. Read TD, Ussery DW (2006) Opening the pan-genomics box - Editorial overview. Current Opinion in Microbiology 9: 496-498.

31. Medini D, Serruto D, Parkhill J, Relman DA, Donati C, et al. (2008) Microbiology in the post-genomic era. Nat Rev Microbiol 6: 419-430.

32. Edgar RC, Batzoglou S (2006) Multiple sequence alignment. Curr Opin Struct Biol 16: 368-373.

33. Batzoglou S (2005) The many faces of sequence alignment. Brief Bioinform 6: 6-22.

34. Maier D (1978) The Complexity of Some Problems on Subsequences and Supersequences. Journal of Association for Computing Machinery 25: 322-336.

35. Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48: 443-453.

36. Smith TF, Waterman MS (1981) Identification of common molecular subsequences. J Mol Biol 147: 195-197.

37. Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, et al. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 25: 3389-3402.

38. Zhang Z, Schwartz S, Wagner L, Miller W (2000) A greedy algorithm for aligning DNA sequences. J Comput Biol 7: 203-214.

39. Kent WJ (2002) BLAT--the BLAST-like alignment tool. Genome Res 12: 656-664.

40. Garey MRJ, D. S. (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman.

41. Edgar RC (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32: 1792-1797.

42. Bradley RK, Roberts A, Smoot M, Juvekar S, Do J, et al. (2009) Fast statistical alignment. PLoS Comput Biol 5: e1000392.

43. Thompson JD, Higgins DG, Gibson TJ (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res 22: 4673-4680.

44. Feng DF, Doolittle RF (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J Mol Evol 25: 351-360.

45. Edgar RC (2004) MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics 5: 113.

46. Stein L (2001) Genome annotation: from sequence to biology. Nat Rev Genet 2: 493-503.

47. Hyatt D, Chen GL, Locascio PF, Land ML, Larimer FW, et al. (2010) Prodigal: prokaryotic gene recognition and translation initiation site identification. BMC Bioinformatics 11: 119.

48. Delcher AL, Bratke KA, Powers EC, Salzberg SL (2007) Identifying bacterial genes and endosymbiont DNA with Glimmer. Bioinformatics 23: 673-679.

49. Lukashin AV, Borodovsky M (1998) GeneMark.hmm: new solutions for gene finding. Nucleic Acids Res 26: 1107-1115.

50. Hu GQ, Zheng X, Zhu HQ, She ZS (2009) Prediction of translation initiation site for microbial genomes with TriTISA. Bioinformatics 25: 123-125.

51. Nielsen P, Krogh A (2005) Large-scale prokaryotic gene prediction and comparison to genome annotation. Bioinformatics 21: 4322-4329.

52. Steen H, Mann M (2004) The ABC's (and XYZ's) of peptide sequencing. Nat Rev Mol Cell Biol 5: 699-711.

53. Overbeek R, Bartels D, Vonstein V, Meyer F (2007) Annotation of bacterial and archaeal genomes: improving accuracy and consistency. Chem Rev 107: 3431-3447.

54. Poptsova MS, Gogarten JP (2010) Using comparative genome analysis to identify problems in annotated microbial genomes. Microbiology 156: 1909-1917.

55. VMware. Available at: http://www.vmware.com/

56. VirtualBox. Available at: http://www.virtualbox.org/

57. Litzkow MaL, M and Mutka, M. {C}ondor - A Hunter of Idle Workstations; 1988.

58. Foster IaK, C (1996) Globus: A Metacomputing Infrastructure Toolkit International Journal of Supercomputer Applications

59. NIST Cloud Computing Definition. Available at: http://csrc.nist.gov/groups/SNS/cloud-computing/

60. Stein LD (2010) The case for cloud computing in genome informatics. Genome Biol 11: 207.

61. Amazon Elastic Compute Cloud. Available at: http://aws.amazon.com/ec2/

62. Magellan: Argonne's DOE Cloud Computing. Available at: http://magellan.alcf.anl.gov/

63. Data Intensive Academic Grid. Available at: http://diagcomputing.org/

64. Dewey CN, Pachter L (2006) Evolution at the nucleotide level: the problem of multiple whole-genome alignment. Hum Mol Genet 15 Spec No 1: R51-56.

65. Didelot X, Lawson D, Darling A, Falush D (2010) Inference of homologous recombination in bacteria using whole-genome sequences. Genetics 186: 1435-1449.

66. Bray N, Dubchak I, Pachter L (2003) AVID: A global alignment program. Genome Res 13: 97-102.

67. Schwartz S, Kent WJ, Smit A, Zhang Z, Baertsch R, et al. (2003) Human-mouse alignments with BLASTZ. Genome Res 13: 103-107.

68. Kurtz S, Phillippy A, Delcher AL, Smoot M, Shumway M, et al. (2004) Versatile and open software for comparing large genomes. Genome Biol 5: R12.

69. Paten B, Herrero J, Beal K, Fitzgerald S, Birney E (2008) Enredo and Pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. Genome Res 18: 1814-1828.

70. Blanchette M, Kent WJ, Riemer C, Elnitski L, Smit AF, et al. (2004) Aligning multiple genomic sequences with the threaded blockset aligner. Genome Res 14: 708-715.

71. Dubchak I, Poliakov A, Kislyuk A, Brudno M (2009) Multiple whole-genome alignments without a reference organism. Genome Res 19: 682-689.

72. Hohl M, Kurtz S, Ohlebusch E (2002) Efficient multiple genome alignment. Bioinformatics 18 Suppl 1: S312-320.

73. Darling AC, Mau B, Blattner FR, Perna NT (2004) Mauve: multiple alignment of conserved genomic sequence with rearrangements. Genome Res 14: 1394-1403.

74. Kumar S, Filipski A (2007) Multiple sequence alignment: in pursuit of homologous DNA positions. Genome Res 17: 127-135.

75. Lunter G (2007) Probabilistic whole-genome alignments reveal high indel rates in the human and mouse genomes. Bioinformatics 23: i289-296.

76. Prakash A, Tompa M (2007) Measuring the accuracy of genome-size multiple alignments. Genome Biol 8: R124.

77. Margulies EH, Cooper GM, Asimenos G, Thomas DJ, Dewey CN, et al. (2007) Analyses of deep mammalian sequence alignments and constraint predictions for 1% of the human genome. Genome Res 17: 760-774.

78. Chen X, Tompa M (2010) Comparative assessment of methods for aligning multiple genome sequences. Nat Biotechnol 28: 567-572.

79. Dewey CN (2007) Aligning multiple whole genomes with Mercator and MAVID. Methods Mol Biol 395: 221-236.

80. Kent WJ, Baertsch R, Hinrichs A, Miller W, Haussler D (2003) Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. Proc Natl Acad Sci U S A 100: 11484-11489.

81. Pham SK, Pevzner PA (2010) DRIMM-Synteny: decomposing genomes into evolutionary conserved segments. Bioinformatics 26: 2509-2516.

82. Deloger M, El Karoui M, Petit MA (2009) A genomic distance based on MUM indicates discontinuity between most bacterial species and genera. J Bacteriol 191: 91-99.

83. Darling AE, Mau B, Perna NT (2010) progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement. PLoS ONE 5: e11147.

84. Treangen TJ, Messeguer X (2006) M-GCAT: interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. BMC Bioinformatics 7: 433.

85. Raphael B, Zhi D, Tang H, Pevzner P (2004) A novel method for multiple alignment of sequences with repeated and shuffled elements. Genome Res 14: 2336-2346.

86. Zhang Y, Waterman MS (2005) An Eulerian path approach to local multiple alignment for DNA sequences. Proc Natl Acad Sci U S A 102: 1285-1290.

87. Rausch T, Emde AK, Weese D, Doring A, Notredame C, et al. (2008) Segment-based multiple sequence alignment. Bioinformatics 24: i187-192.

88. Paten B, Herrero J, Beal K, Birney E (2009) Sequence progressive alignment, a framework for practical large-scale probabilistic consistency alignment. Bioinformatics 25: 295-301.

89. Notredame C, Higgins DG, Heringa J (2000) T-Coffee: A novel method for fast and accurate multiple sequence alignment. J Mol Biol 302: 205-217.

90. Jacobson G, Vo K-P (1992) Heaviest Increasing/Common Subsequence Problems. Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching: Springer-Verlag. pp. 52-66.

91. Di Tommaso P, Orobitg M, Guirado F, Cores F, Espinosa T, et al. (2010) Cloud-Coffee: implementation of a parallel consistency-based multiple alignment algorithm in the T-Coffee package and its benchmarking on the Amazon Elastic-Cloud. Bioinformatics 26: 1903-1904.

92. Delcher AL, Harmon D, Kasif S, White O, Salzberg SL (1999) Improved microbial gene identification with GLIMMER. Nucleic Acids Res 27: 4636-4641.

93. Pati A, Ivanova NN, Mikhailova N, Ovchinnikova G, Hooper SD, et al. (2010) GenePRIMP: a gene prediction improvement pipeline for prokaryotic genomes. Nat Methods 7: 455-457.

94. Angiuoli SV, Gussman A, Klimke W, Cochrane G, Field D, et al. (2008) Toward an online repository of Standard Operating Procedures (SOPs) for (meta)genomic annotation. OMICS 12: 137-141.

95. Bakke P, Carney N, Deloache W, Gearing M, Ingvorsen K, et al. (2009) Evaluation of three automated genome annotations for Halorhabdus utahensis. PLoS ONE 4: e6291.

96. Siezen RJ, van Hijum SA (2010) Genome (re-)annotation and open-source annotation pipelines. Microb Biotechnol 3: 362-369.

97. Medigue C, Moszer I (2007) Annotation, comparison and databases for hundreds of bacterial genomes. Res Microbiol 158: 724-736.

98. Otto TD, Dillon GP, Degrave WS, Berriman M (2011) RATT: Rapid Annotation Transfer Tool. Nucleic Acids Res.

99. Petty NK (2010) Genome annotation: man versus machine. Nat Rev Microbiol 8: 762.

100. Schatz MC (2010) The missing graphical user interface for genomics. Genome Biol 11: 128.

101. Goecks J, Nekrutenko A, Taylor J (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol 11: R86.

102. Orvis J, Crabtree J, Galens K, Gussman A, Inman JM, et al. (2010) Ergatis: a web interface and scalable software system for bioinformatics workflows. Bioinformatics 26: 1488-1492.

103. Reich M, Liefeld T, Gould J, Lerner J, Tamayo P, et al. (2006) GenePattern 2.0. Nat Genet 38: 500-501.

104. Hull D, Wolstencroft K, Stevens R, Goble C, Pocock MR, et al. (2006) Taverna: a tool for building and running workflows of services. Nucleic Acids Res 34: W729-732.

105. Angiuoli S, Cochrane G, Field D, Garrity GM, Gussman A, et al. (2008) Towards a online repository of Standard Operating Procedures (SOPs) for (meta)genomic annotation. OMICS: A journal of integrative biology (in press).

106. Aziz RK, Bartels D, Best AA, DeJongh M, Disz T, et al. (2008) The RAST Server: rapid annotations using subsystems technology. BMC Genomics 9: 75.

107. Meyer F, Paarmann D, D'Souza M, Olson R, Glass EM, et al. (2008) The metagenomics RAST server - a public resource for the automatic phylogenetic and functional analysis of metagenomes. BMC Bioinformatics 9: 386.

108. Hemmerich C, Buechlein A, Podicheti R, Revanna KV, Dong Q (2010) An Ergatis-based prokaryotic genome annotation web server. Bioinformatics 26: 1122-1124.

109. Annotation Engine. Available at: http://ae.igs.umaryland.edu

110. Schloss PD, Westcott SL, Ryabin T, Hall JR, Hartmann M, et al. (2009) Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities. Appl Environ Microbiol 75: 7537-7541.

111. Caporaso JG, Kuczynski J, Stombaugh J, Bittinger K, Bushman FD, et al. (2010) QIIME allows analysis of high-throughput community sequencing data. Nat Methods 7: 335-336.

112. Stewart AC, Osborne B, Read TD (2009) DIYA: a bacterial annotation pipeline for any genomics lab. Bioinformatics 25: 962-963.

113. Bateman A, Wood M (2009) Cloud computing. Bioinformatics 25: 1475.

114. Dudley JT, Butte AJ (2010) In silico research in the era of cloud computing. Nat Biotechnol 28: 1181-1185.

115. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51: 107-113.

116. Schatz MC (2009) CloudBurst: highly sensitive read mapping with MapReduce. Bioinformatics 25: 1363-1369.

117. Langmead B, Schatz MC, Lin J, Pop M, Salzberg SL (2009) Searching for SNPs with cloud computing. Genome Biol 10: R134.

118. Langmead B, Hansen KD, Leek JT (2010) Cloud-scale RNA-sequencing differential expression analysis with Myrna. Genome Biol 11: R83.

119. Trelles O, Prins P, Snir M, Jansen RC (2011) Big data, but are we ready? Nat Rev Genet 12: 224.

120. Dudley JT, Pouliot Y, Chen R, Morgan AA, Butte AJ (2010) Translational bioinformatics in the cloud: an affordable alternative. Genome Med 2: 51.

121. Wilkening J, Wilke A, Desai N, Meyer DF (2009) Using Clouds for Metagenomics: A Case Study. IEEE Cluster 2009. New Orleans, LA: IEEE.

122. Doring A, Weese D, Rausch T, Reinert K (2008) SeqAn an efficient, generic C++ library for sequence analysis. BMC Bioinformatics 9: 11.

123. Gusfield D (1997) Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology: New York: Cambridge University Press.

124. Pevzner P, Tesler G (2003) Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. Genome Res 13: 37-45.

125. Bourque G, Pevzner PA, Tesler G (2004) Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse, and rat genomes. Genome Res 14: 507-516.

126. Ford FR, Fulkerson DR (1956) Maximal flow through a network. Canadian Journal of Mathematics 8: 399-404.

127. Corel E, Pitschi F, Morgenstern B (2010) A min-cut algorithm for the consistency problem in multiple sequence alignment. Bioinformatics 26: 1015-1021.

128. Edmonds J, Karp RM (1972) Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. J ACM 19: 248-264.

129. Wheeler DL, Barrett T, Benson DA, Bryant SH, Canese K, et al. (2008) Database resources of the National Center for Biotechnology Information. Nucleic Acids Res 36: D13-21.

130. Levy S, Sutton G, Ng PC, Feuk L, Halpern AL, et al. (2007) The diploid genome sequence of an individual human. PLoS Biol 5: e254.

131. Ahn SM, Kim TH, Lee S, Kim D, Ghang H, et al. (2009) The first Korean genome sequence and analysis: full genome sequencing for a socio-ethnic group. Genome Res 19: 1622-1629.

132. Wang J, Wang W, Li R, Li Y, Tian G, et al. (2008) The diploid genome sequence of an Asian individual. Nature 456: 60-65.

133. Li R, Zhu H, Ruan J, Qian W, Fang X, et al. (2010) De novo assembly of human genomes with massively parallel short read sequencing. Genome Res 20: 265-272.

134. Rosenbloom KR, Dreszer TR, Pheasant M, Barber GP, Meyer LR, et al. ENCODE whole-genome data in the UCSC Genome Browser. Nucleic Acids Res 38: D620-625.

135. Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, et al. (2001) dbSNP: the NCBI database of genetic variation. Nucleic Acids Res 29: 308-311.

136. IHGSC (2001) Initial sequencing and analysis of the human genome. Nature 409: 860-921.

137. Murtagh F (1984) Complexities of Hierarchic Clustering Algorithms: the state of the art. Computational Statistics Quarterly 1: 101-113.

138. Maione D, Margarit I, Rinaudo CD, Masignani V, Mora M, et al. (2005) Identification of a universal Group B streptococcus vaccine by multiple genome screen. Science 309: 148-150.

139. Rappuoli R, Covacci A (2003) Reverse vaccinology and genomics. Science 302: 602.

140. Budroni S, Siena E, Dunning Hotopp JC, Seib KL, Serruto D, et al. (2011) Neisseria meningitidis is structured in clades associated with restriction modification systems that modulate homologous recombination. Proc Natl Acad Sci U S A.

141. Samet H (2006) Foundations of Multidimensional and Metric Data Structures: Morgan Kaufmann.

142. Li L, Stoeckert CJ, Jr., Roos DS (2003) OrthoMCL: identification of ortholog groups for eukaryotic genomes. Genome Res 13: 2178-2189.

143. Warren AS, Archuleta J, Feng WC, Setubal JC (2010) Missing genes in the annotation of prokaryotic genomes. BMC Bioinformatics 11: 131.

144. Brenner SE (1999) Errors in genome annotation. Trends Genet 15: 132-133.

145. van den Berg BH, McCarthy FM, Lamont SJ, Burgess SC (2010) Re-annotation is an essential step in systems biology modeling of functional genomics data. PLoS ONE 5: e10642.

146. Devos D, Valencia A (2001) Intrinsic errors in genome annotation. Trends Genet 17: 429-431.

147. Rusniok C, Vallenet D, Floquet S, Ewles H, Mouze-Soulama C, et al. (2009) NeMeSys: a biological resource for narrowing the gap between sequence and function in the human pathogen Neisseria meningitidis. Genome Biol 10: R110.

148. Touchon M, Hoede C, Tenaillon O, Barbe V, Baeriswyl S, et al. (2009) Organised genome dynamics in the Escherichia coli species results in highly diverse adaptive paths. PLoS Genet 5: e1000344.

149. Palleja A, Harrington ED, Bork P (2008) Large gene overlaps in prokaryotic genomes: result of functional constraints or mispredictions? BMC Genomics 9: 335.

150. Meyer IM, Durbin R (2002) Comparative ab initio prediction of gene structures using pair HMMs. Bioinformatics 18: 1309-1318.

151. Meyer IM, Durbin R (2004) Gene structure conservation aids similarity based gene prediction. Nucleic Acids Res 32: 776-783.

152. Chatterji S, Pachter L (2006) Reference based annotation with GeneMapper. Genome Biol 7: R29.

153. GridEngine. Available at: http://gridengine.org

154. Apache Hadoop. Available at: http://hadoop.apache.org/

155. Ubuntu. Available at: http://www.ubuntu.com/

156. Hudson Continuous Integration. Available at: http://hudson-ci.org/

157. EC2 and Ubuntu Available at: http://alestic.com/

158. Science Clouds - Nimbus Open Source IaaS Cloud Computing Software. Available at: http://scienceclouds.org/

159. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215: 403-410.

160. CloVR-Search SOP v1.0. Available at: http://clovr.org/methods/clovr-search/

161. White J, Arze, Cesar, Matalka, Malcolm, Team, The CloVR, Angiuoli, Samuel, and Fricke, W. Florian. CloVR-16S: Phylogenetic microbial community composition analysis based on 16S ribosomal RNA amplicon sequencing – standard operating procedure, version1.0. Available from Nature Precedings <http://dx.doi.org/10.1038/npre.2011.5888.1> (2011)

162. White J, Arze, Cesar, Matalka, Malcolm, Team, The CloVR, Angiuoli, Samuel, and Fricke, W. Florian. CloVR-Metagenomics: Functional and taxonomic microbial community characterization from metagenomic whole-genome shotgun (WGS) sequences – standard operating procedure, version 1.0. Available from Nature Preceding <http://dx.doi.org/10.1038/npre.2011.5886.1> (2011)

163. Galens K, White, James, Arze, Cesar, Matalka, Malcolm, Gwinn Giglio, Michelle, Team, The CloVR, Angiuoli, Samuel, and Fricke, W. Florian. CloVR-Microbe: Assembly, gene finding and functional annotation of raw sequence data from single microbial genome projects – standard operating procedure, version 1.0. Available from Nature Preceding <http://dx.doi.org/10.1038/npre.2011.5887.1> (2011)

164. Amazon Elastic Compute Cloud API Reference Available at: http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/

165. Rapier C, Bennett B (2008) High speed bulk data transfer using the SSH protocol. Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities. Baton Rouge, Louisiana: ACM. pp. 1-7.

166. Tridgell A (1999) Efficient Algorithms for Sorting and Synchronization: The Australian National University.

167. White J, Matalka, Malcolm, Fricke, W. Florian, and Angiuoli, Samuel. Cunningham: a BLAST Runtime Estimator. Available from Nature Precedings <http://dx.doi.org/10.1038/npre.2011.5593.1> (2011)

168. Sandberg R, Goldberg D. , Kleiman S, Walsh D. , Lyon B. (1985) Design and Implementation or the Sun Network Filesystem.

169. Ganglia Monitoring System. Available at: http://ganglia.sourceforge.net/

170. Keahey K, Freeman T (2008) Contextualization: Providing One-Click Virtual Clusters. Proceedings of the 2008 Fourth IEEE International Conference on eScience: IEEE Computer Society. pp. 301-308.

171. Afgan E, Baker D, Coraor N, Chapman B, Nekrutenko A, et al. (2010) Galaxy CloudMan: delivering cloud compute clusters. BMC Bioinformatics 11 Suppl 12: S4.

172. STARDEV: Cluster. Available at: http://web.mit.edu/stardev/cluster/

173. Amazon Simple Storage Service. Available at: http://aws.amazon.com/s3/

174. Hadoop Distributed File System. Available at: http://hadoop.apache.org/hdfs/

175. Amazon Elastic Block Store. Available at: http://aws.amazon.com/ebs/

176. Allcock W (2003) GridFTP: Protocol Extensions to FTP for the Grid.

177. Suzek BE, Huang H, McGarvey P, Mazumder R, Wu CH (2007) UniRef: comprehensive and non-redundant UniProt reference clusters. Bioinformatics 23: 1282-1288.

178. Field D, Garrity G, Gray T, Morrison N, Selengut J, et al. (2008) The minimum information about a genome sequence (MIGS) specification. Nat Biotechnol 26: 541-547.

179. Callaway E (2010) Mutation-prediction software rewarded. Nature.

180. QIIME Virtual Box. Available at: http://qiime.sourceforge.net/install/virtual_box.html

181. Shendure J, Ji H (2008) Next-generation DNA sequencing. Nat Biotechnol 26: 1135-1145.

182. Miller JR, Delcher AL, Koren S, Venter E, Walenz BP, et al. (2008) Aggressive assembly of pyrosequencing reads with mates. Bioinformatics 24: 2818-2824.

183. Zerbino DR, Birney E (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res 18: 821-829.

184. Lowe TM, Eddy SR (1997) tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. Nucleic Acids Res 25: 955-964.

185. Lagesen K, Hallin P, Rodland EA, Staerfeldt HH, Rognes T, et al. (2007) RNAmmer: consistent and rapid annotation of ribosomal RNA genes. Nucleic Acids Res 35: 3100-3108.

186. BLAST-Extend-Repraze. Available at: http://ber.sourceforge.net/

187. Tatusov RL, Fedorova ND, Jackson JD, Jacobs AR, Kiryutin B, et al. (2003) The COG database: an updated version includes eukaryotes. BMC Bioinformatics 4: 41.

188. Eddy SR (1998) Profile hidden Markov models. Bioinformatics 14: 755-763.

189. Bateman A, Coin L, Durbin R, Finn RD, Hollich V, et al. (2004) The Pfam protein families database. Nucleic Acids Res 32: D138-141.

190. Selengut JD, Haft DH, Davidsen T, Ganapathy A, Gwinn-Giglio M, et al. (2007) TIGRFAMs and Genome Properties: tools for the assignment of molecular function and biological process in prokaryotic genomes. Nucleic Acids Res 35: D260-264.

191. Cole JR, Wang Q, Cardenas E, Fish J, Chai B, et al. (2009) The Ribosomal Database Project: improved alignments and new tools for rRNA analysis. Nucleic Acids Res 37: D141-145.

192. Edgar RC (2010) Search and clustering orders of magnitude faster than BLAST. Bioinformatics 26: 2460-2461.

193. White JR, Nagarajan N, Pop M (2009) Statistical methods for detecting differentially abundant features in clinical metagenomic samples. PLoS Comput Biol 5: e1000352.

194. Wang Q, Garrity GM, Tiedje JM, Cole JR (2007) Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. Appl Environ Microbiol 73: 5261-5267.

195. Pruitt KD, Tatusova T, Klimke W, Maglott DR (2009) NCBI Reference Sequences: current status, policy and new initiatives. Nucleic Acids Res 37: D32-36.

196. Noguchi H, Park J, Takagi T (2006) MetaGene: prokaryotic gene finding from environmental genome shotgun sequences. Nucleic Acids Res 34: 5623-5630.

197. Adams MD, Celniker SE, Holt RA, Evans CA, Gocayne JD, et al. (2000) The genome sequence of Drosophila melanogaster. Science 287: 2185-2195.

198. Eddy SR (2009) A new generation of homology search tools based on probabilistic inference. Genome Inform 23: 205-211.

199. Finn RD, Mistry J, Tate J, Coggill P, Heger A, et al. (2010) The Pfam protein families database. Nucleic Acids Res 38: D211-222.

200. Haft DH, Selengut JD, White O (2003) The TIGRFAMs database of protein families. Nucleic Acids Res 31: 371-373.

201. Dinsdale EA, Edwards RA, Hall D, Angly F, Breitbart M, et al. (2008) Functional metagenomic profiling of nine biomes. Nature 452: 629-632.

202. Grice EA, Snitkin ES, Yockey LJ, Bermudez DM, Liechty KW, et al. (2010) Longitudinal shift in diabetic wound microbiota correlates with prolonged skin defense response. Proc Natl Acad Sci U S A 107: 14799-14804.

203. Koenig JE, Spor A, Scalfone N, Fricker AD, Stombaugh J, et al. (2010) Microbes and Health Sackler Colloquium: Succession of microbial consortia in the developing infant gut microbiome. Proc Natl Acad Sci U S A.

204. Qin J, Li R, Raes J, Arumugam M, Burgdorf KS, et al. (2010) A human gut microbial gene catalogue established by metagenomic sequencing. Nature 464: 59-65.

205. Ravel J, Gajer P, Abdo Z, Schneider GM, Koenig SS, et al. (2010) Microbes and Health Sackler Colloquium: Vaginal microbiome of reproductive-age women. Proc Natl Acad Sci U S A.

206. Turnbaugh P, Ridaura V, Faith J, Rey FE, Knight R, et al. (2009) The Effect of Diet on the Human Gut Microbiome: A Metagenomic Analysis in Humanized Gnotobiotic Mice. Sci Transl Med 1: 6ra14.

207. Turnbaugh PJ, Hamady M, Yatsunenko T, Cantarel BL, Duncan A, et al. (2009) A core gut microbiome in obese and lean twins. Nature 457: 480-484.

208. Glass EM, Wilkening J, Wilke A, Antonopoulos D, Meyer F (2010) Using the metagenomics RAST server (MG-RAST) for analyzing shotgun metagenomes. Cold Spring Harb Protoc 2010: pdb prot5368.

209. Markoff JH, S. (14 June 2006) Hiding in plain sight, Google seeks more power. New York Times.

210. Reid SD, Herbelin CJ, Bumbaugh AC, Selander RK, Whittam TS (2000) Parallel evolution of virulence in pathogenic Escherichia coli. Nature 406: 64-67.

211. Wirth T, Falush D, Lan R, Colles F, Mensa P, et al. (2006) Sex and virulence in Escherichia coli: an evolutionary perspective. Mol Microbiol 60: 1136-1151.

212. Sahl JW, Steinsland H, Redman JC, Angiuoli SV, Nataro JP, et al. (2011) A comparative genomic analysis of diverse clonal types of enterotoxigenic Escherichia coli reveals pathovar-specific conservation. Infect Immun 79: 950-960.

213. Talavera G, Castresana J (2007) Improvement of phylogenies after removing divergent and ambiguously aligned blocks from protein sequence alignments. Syst Biol 56: 564-577.

214. Price MN, Dehal PS, Arkin AP (2009) FastTree: computing large minimum evolution trees with profiles instead of a distance matrix. Mol Biol Evol 26: 1641-1650.

215. Crabtree J, Angiuoli SV, Wortman JR, White OR (2007) Sybil: methods and software for multiple genome comparison and visualization. Methods Mol Biol 408: 93-108.

216. Stephens DS, Greenwood B, Brandtzaeg P (2007) Epidemic meningitis, meningococcaemia, and Neisseria meningitidis. Lancet 369: 2196-2210.

217. Snyder LA, Saunders NJ (2006) The majority of genes in the pathogenic Neisseria species are present in non-pathogenic Neisseria lactamica, including those designated as 'virulence genes'. BMC Genomics 7: 128.

218. Bille E, Ure R, Gray SJ, Kaczmarski EB, McCarthy ND, et al. (2008) Association of a bacteriophage with meningococcal disease in young adults. PLoS ONE 3: e3885.

219. Dunning Hotopp JC, Grifantini R, Kumar N, Tzeng YL, Fouts D, et al. (2006) Comparative genomics of Neisseria meningitidis: core genome, islands of horizontal transfer and pathogen-specific genes. Microbiology 152: 3733-3749.

220. Roach JC, Glusman G, Smit AF, Huff CD, Hubley R, et al. (2010) Analysis of genetic inheritance in a family quartet by whole-genome sequencing. Science 328: 636-639.

221. Tewhey R, Bansal V, Torkamani A, Topol EJ, Schork NJ (2011) The importance of phase information for human genomics. Nat Rev Genet 12: 215-223.

222. Leary RJ, Kinde I, Diehl F, Schmidt K, Clouser C, et al. (2010) Development of personalized tumor biomarkers using massively parallel sequencing. Sci Transl Med 2: 20ra14.

223. Shields R (2011) Common disease: are causative alleles common or rare? PLoS Biol 9: e1001009.

224. Internet2. Available at: http://www.internet2.edu

225. Hsi-Yang Fritz M, Leinonen R, Cochrane G, Birney E (2011) Efficient storage of high throughput sequencing data using reference-based compression. Genome Res.

226. Tsolis RM, Seshadri R, Santos RL, Sangari FJ, Lobo JM, et al. (2009) Genome degradation in Brucella ovis corresponds with narrowing of its host range and tissue tropism. PLoS ONE 4: e5519.

227. Fedorova ND, Khaldi N, Joardar VS, Maiti R, Amedeo P, et al. (2008) Genomic islands in the pathogenic filamentous fungus Aspergillus fumigatus. PLoS Genet 4: e1000046.

228. Carlton JM, Adams JH, Silva JC, Bidwell SL, Lorenzi H, et al. (2008) Comparative genomics of the neglected human malaria parasite Plasmodium vivax. Nature 455: 757-763.

229. Donati C, Hiller NL, Tettelin H, Muzzi A, Croucher NJ, et al. (2010) Structure and dynamics of the pan-genome of Streptococcus pneumoniae and closely related species. Genome Biol 11: R107.

230. Heinicke S, Livstone MS, Lu C, Oughtred R, Kang F, et al. (2007) The Princeton Protein Orthology Database (P-POD): a comparative genomics analysis tool for biologists. PLoS ONE 2: e766.

231. Ghedin E, Wang S, Spiro D, Caler E, Zhao Q, et al. (2007) Draft genome of the filarial nematode parasite Brugia malayi. Science 317: 1756-1760.

232. Hotopp JC, Lin M, Madupu R, Crabtree J, Angiuoli SV, et al. (2006) Comparative genomics of emerging human ehrlichiosis agents. PLoS Genet 2: e21.

233. Filliol I, Motiwala AS, Cavatore M, Qi W, Hazbon MH, et al. (2006) Global phylogeny of Mycobacterium tuberculosis based on single nucleotide polymorphism (SNP) analysis: insights into tuberculosis evolution, phylogenetic accuracy of other DNA fingerprinting systems, and recommendations for a minimal standard SNP set. J Bacteriol 188: 759-772.

234. Gill SR, Fouts DE, Archer GL, Mongodin EF, Deboy RT, et al. (2005) Insights on evolution of virulence and resistance from the complete genome analysis of an

early methicillin-resistant Staphylococcus aureus strain and a biofilm-producing methicillin-resistant Staphylococcus epidermidis strain. J Bacteriol 187: 2426-2438.

235. Gardner MJ, Bishop R, Shah T, de Villiers EP, Carlton JM, et al. (2005) Genome sequence of Theileria parva, a bovine pathogen that transforms lymphocytes. Science 309: 134-137.

236. El-Sayed NM, Myler PJ, Blandin G, Berriman M, Crabtree J, et al. (2005) Comparative genomics of trypanosomatid parasitic protozoa. Science 309: 404-409.

237. Tettelin H, Masignani V, Cieslewicz MJ, Donati C, Medini D, et al. (2005) Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: implications for the microbial "pan-genome". Proc Natl Acad Sci U S A 102: 13950-13955.

238. Garrity GM, Field D, Kyrpides N, Hirschman L, Sansone SA, et al. (2008) Toward a standards-compliant genomic and metagenomic publication record. OMICS 12: 157-160.

239. Standards in Genomic Sciences. Available at: http://standardsingenomics.org