Scoring-and-Unfolding Trimmed Tree Assembler: Algorithms for Assembling Genome Sequences Accurately and Efficiently

by

Giuseppe Narzisi

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

May 2011

Bud Mishra — Advisor

To Valentina

&

My family

Acknowledgements

The work presented in this dissertation would not have been possible without the contribution of many great people.

My advisor, Professor Bud Mishra, has definitely played the most important role during my studies and research at New York University. It has been a great privilege to work with him for so many years, since I first entered the Ph.D. program at NYU. He has been a scientific mentor, a human guide and an enormous source of ideas thanks to his extraordinary and deep proficiency in many different fields. In a time when sequence assembly was assumed to be an almost solved puzzle, he encouraged me to rethink again about the problem without being biased by the currently developed models and helped me to overcome the many obstacles that I faced as I was making progress in my studies. He has certainly made the major contribution to shape the scientific research presented in this thesis.

I was fortunate to have a very diverse thesis committee which has provided many important insights during the preparation of the dissertation. In particular I am grateful to Professor Ernest Davis for taking the time to read and review my thesis with so much care. By raising many fundamental questions, he has al-

lowed me to clarify several important aspects of the thesis for the general reader. I thank Professor Michael Schatz for making sure that the theoretical framework presented in the dissertation was correctly presented and related to the relevant prior art extensively. I also express my thanks to Professor Alan Siegel for improving the presentation and stile of the thesis. He has been not just a scientific mentor but also a tutoring figure in virtue of his high commitment to the value of education. Finally, I would like to thank Professor Raul Rabadan for suggesting many areas of application of the tools developed in this thesis.

One of the contributions of the dissertation (TotalReCaller) is the result of the joint work with Fabian Menges. I am particularly grateful to him for such collaboration as well as for all the valuable and energetic discussions that we had on many of the topics presented in this thesis. I also would like to thank all the members of the NYU Bioinformatics Group for creating a unique and exciting research environment during my work and study at NYU. In particular I would like to thank: Salvatore Paxia, Fabian Menges, Matthias Heymann, Andreas Witzel, Andrew Sundstrom, Samantha Kleinberg, Antonina Mitrofanova, Iuliana Ionita, Venkatesh P. Mysore, Marco Antoniotti, Bing Sun, Ofer Gill, Josh Mincer, Seongho Ryu, Shaila Musharoff, Fang Cheng, Raoul-Sam Daruwala, Yi (Joey) Zhou, Peyman Faratin, Pierre Franquin, Athena Shilin, and Thomas S. Anantharaman.

Five years of studies have been a long journey with many ups and downs but, even during the most difficult times, there was always a constant figure in my life: my love and wife Valentina. She has been an extraordinary source of love and

stability in my life, thanks to her continuous support and trust. She has clearly played an important role for the successful completion of this thesis.

Finally, I thank my whole family, my parent and my younger brother, for being always very supportive even when I decided to embrace the American dream. Despite the big ocean dividing our lives, they have always been caring as I have never left them.

Abstract

The recent advances in DNA sequencing technology and their many potential applications to Biology and Medicine have rekindled enormous interest in several classical algorithmic problems at the core of Genomics and Computational Biology: primarily, the *whole-genome sequence assembly problem* (WGSA). Two decades back, in the context of the Human Genome Project, the problem had received unprecedented scientific prominence: its computational complexity and intractability were thought to have been well understood; various competitive heuristics, thoroughly explored and the necessary software, properly implemented and validated. However, several recent studies, focusing on the experimental validation of de novo assemblies, have highlighted several limitations of the current assemblers.

Intrigued by these negative results, this dissertation reinvestigates the algorithmic techniques required to correctly and efficiently assemble genomes. Mired by its connection to a well-known $\mathcal{NP}$-complete combinatorial optimization problem, historically, WGSA has been assumed to be amenable only to greedy and heuristic methods. By placing efficiency as their priority, these methods opted to rely on local searches, and are thus inherently approximate, ambiguous or

error-prone. This dissertation presents a novel sequence assembler, SUTTA, that dispenses with the idea of limiting the solutions to just the approximated ones, and instead favors an approach that could potentially lead to an exhaustive (exponential-time) search of all possible layouts but tames the complexity through constrained search (Branch-and-Bound) and quick identification and pruning of implausible solutions.

Complementary to this problem is the task of validating the generated assemblies. Unfortunately, no commonly accepted method exists yet and widely used metrics to compare the assembled sequences emphasize only size, poorly capturing quality and accuracy. This dissertation also addresses these concerns by developing a more comprehensive metric, the Feature-Response Curve, that, using ideas from classical ROC (receiver-operating characteristic) curve, more faithfully captures the trade-off between contiguity and quality.

Finally, this dissertation demonstrates the advantages of a complete pipeline integrating base-calling (TotalReCaller) with assembly (SUTTA) in a Bayesian manner.

Contents

List of Figures

List of Tables

List of Appendices

Introduction

This dissertation addresses the problem of assembling the genome sequence of an organism using the available sequencing technologies and long-range information. Producing a complete and high-quality assembly of a genome has fundamental implications in Biology and Medicine; however such a task is particularly challenging for large, repeat-rich genomes such as those of mammals. For this reason new assembly tools, cautious experimental design, and novel metric for assembly validation must be developed if there is any hope to successfully and correctly assembly any genome. In this introductory chapter I will describe the major questions and grand challenges that must be faced in this field while explaining how the thesis contributes to address these problems.

Motivation

Probably there is no better way to describe my motivation for working on genome sequence assembly than considering how to answer the following general questions. The answers also highlight the reasons why this problem has received so much consideration by both the computer science and computational biology

community in the last 30 years.

Why is genome sequencing so important? Knowing the full sequence of a genome (in particular Human) has many important and direct applications to Biology and Medicine. Following is a (not exhaustive) list of such applications:

- Understand evolution: compare genomes of near-by species.
- Understand traits and diseases: compare genomes of wild-types vs. mutants, normals vs. patients, normals vs. tumor, etc.
- Find genes, regulatory regions, exon-intron-boundaries, splicing sites, etc. computationally (hence, more cheaply and quickly).
- Understand how the genome as a whole works: how genes work together to direct the growth, development and maintenance of an entire organism. Especially, the motifs and patterns in intergenic regions (containing so-called "junk" DNA).

If, by developing a successful technology for genome assembly, it will be possible to solve at least one of such challenges in Medicine and Biology, then we will have made an invaluable contribution to these fields.

Why is genome sequencing so difficult? In a more general framework, computer science researchers first formalized the shotgun sequence assembly problem as an "approximation" to finding the shortest common superstring of a set of sequences. Because of its natural connection to a well known $\mathcal{NP}$*-complete*

combinatorial optimization problem (Shortest Superstring Problem), in these solutions, accuracy is inevitably traded for computational efficiency with greedy and heuristic methods being the preferred choice. However these approaches have their foundations on the following argument: if the DNA was totally random then the overlap information between the sequences would be sufficient to reassemble the full sequence and greedy strategies would always have satisfactory performance. Unfortunately this assumption is not true for real biological problem. In fact genomes (especially Eukaryotic ones) are characterized by *non-random structures* (e.g., repeated regions, rearrangements, segmental duplications etc.) which complicate the assembly problem and make greedy strategies fail invariably. In addition, the sequence reads generated by the sequencing machines are not error-free and the error profiles change quickly over time in response to the changes in sequencing technology. This makes the assembly problem also *sequencing-technology dependent* and the algorithms must change to accommodate changes to read-length, errors profiles, and nature and availability of long range information.

How well have we done so far? The initial "draft" sequence of the human genome [38] has been revised several times, since its first publication, each revision eliminating various classes of errors through successive heuristic advances; nevertheless, genome sequencing continues to be viewed as an inexact craft and inadequate in controlling the number of errors, which in the draft genomes are estimated to be up to hundreds or even thousands [86]. So ten years after the publication of the first draft of the human genome (2001), there is still need for

error-free and efficient methods to further improve the quality of the assemblies.

What is missing? Despite extensive usage of these methods and documentation in the literature for almost 20 years, there still exists no rigorous characterization of these methods, i.e., (*i*) no complete analysis of their performance in the context of varying genomic structures, (*ii*) no comparative study on the accuracy and efficiency of each of their heuristics, (*iii*) no guiding principles on how to choose and adapt the heuristics to changing DNA sequencing technologies, and finally, (*iv*) no flexibility in exploiting the long range information to decipher structural variation or achieve haplotypic disambiguation. Last but not least, lacking such an extensive understanding, no prescriptive framework has resulted for designing the next generation of assemblers that aim to be more accurate, adaptively efficient and extensible to deal with possible future technologies.

The Sense of the Approximation

Although the process of assembling a genome is often described as solving a complex jigsaw puzzle, in reality there are even further complications. This is mostly due to the fact that we still do not completely understand the global structure of the human genome (as well as of many other species). How the genome is modeled has strong implications not only in the way the problem is formulated but also in the accuracy of the methods used to tackle it. To understand this point it is helpful to observe that the sequence assembly problem is unfortunately a "wicked problem" [83]. This is a term used to describe a problem that is difficult or im-

possible to solve because of *incomplete*, *contradictory*, and *changing* requirements that are often difficult to recognize. Moreover, because of complex interdependencies, the effort to solve one aspect of a wicked problem may reveal or create other problems. In the context of genome assembly, the incomplete, contradictory, and changing requirements correspond to the underline genome structure. Since its discovery in 1953 by James D. Watson and Francis Crick, scientists have made a lot of progress in understanding the DNA chemical structure, but only more recently, after the first draft of the human genome was assembled, we know more about its base-by-base structure. Real genomes (especially Eukaryotic ones) contain non-random structures (e.g., repeated regions, rearrangements, segmental duplications etc.) which complicate the assembly problem and any mathematical formulation will never be complete and correct until these structures are fully characterized. As a consequence, in the presence of various unmodeled genome structure, an assembly method must make a choice among quality, quantity and nature of the input data, computational complexity, false discovery rate (chimeric contigs) and false negative rates (gaps in the assembly). We have chosen to emphasize a lower false discovery rate (fdr). Though it is difficult to have a *sense of the approximation* of the sequence to the *true* sequence while they are being assembled, it is possible to analyze the accuracy of an assembler using simulated genomes (of varying complexities) and feature-response-curves with features that are highly indicative of certain kinds of errors. Nonetheless, the approach presented in this dissertation is not yet immune to these criticisms, but it contributes a new framework that has potential to be more faithful to biology.

Thesis Contributions

This dissertation contributes three ways to the field of Sequence Assembly and Genomics in general:

1. A novel DNA sequence assembler, called **Scoring-and-Unfolding Trimmed Tree Assembler (SUTTA)**. Despite the negative theoretical results of the complexity of the assembly problem ($\mathcal{NP}$-hard), SUTTA dispenses with the idea of using greedy and heuristic methods (standardly used by all the current state-of-the-art sequence assemblers) in favor of a *brute-force* approach whose complexity is carefully tamed with constrained search method (branch-and-bound). To achieve this goal SUTTA relies on flexible designed score functions that can combine data from different technologies.

2. A new assembly metric, named **Feature-Response Curve (FRC)**. Widely used metrics to compare the assembled sequences up to now emphasize only size, while scant attention has been paid to evaluate quality and accuracy. To address this problem the FRC has been designed using ideas from the classical ROC (receiver-operating characteristic) curve, to capture the trade-offs between quality and sequence size into a single metric.

3. A new Base-Caller, called **TotalReCaller**. By improving the quality of the input DNA sequences, base-calling and error correction tools play a significant role to enable more accurate sequence assemblies. TotalReCaller is a novel tool that combines base-calling and alignment into a unified framework that concurrently performs base-calling, alignment, and error correc-

tion. Like SUTTA, TotalReCaller also relies on global optimization search methods (branch-and-bound and beam-search) to optimize a Bayesian score function that takes into accounts both intensity signals and alignments to a reference genome.

Thesis Outline

The dissertation is organized as follows. Chapter 1 covers the fundamental background in genome sequencing and sequence assembly. Chapter 2 investigates the major complexity results in sequence assembly, in particular highlighting the inconsistencies in the early formulations. Chapter 3 reviews the different assembly paradigms that have been designed over the years to adapt to various sequencing projects and technologies. Chapter 4 presents a detailed description of the first contribution of this dissertation: the Scoring-and-Unfolding Trimmed Tree Assembler (SUTTA). SUTTA is based on a different formulation of the sequence assembly problem (as constrained optimization) and uses the branch-and-bound method to quickly identify and prune implausible overlays. Chapter 5 introduces and describes the second contribution of the dissertation: the Feature-Response curve (FRC). This new metric more satisfactorily captures the trade-offs between quality and size of the assembled sequences. Chapter 6 presents an extensive set of experimental results to compare SUTTAs performance relative to many state-of-the-art assembly algorithms in the literature. The analysis is performed under both standard metrics (N50, coverage, contig sizes, etc.) as well as the Feature-Response Curves introduced in 5. Finally, chapter 7 presents the third

contribution of this dissertation: the TotalReCaller base-caller, which combines base-calling and alignment into a new re-sequencing framework. This final chapter also demonstrates the advantages of a complete de novo assembly pipeline integrating TotalReCaller (for base-calling) with SUTTA (for sequence assembly) in a Bayesian manner.

Chapter 1

Genome Sequencing and Assembly

1.1 Introduction

A combination of tremendous advances in sequencing technologies, chemistry and computer science has revolutionized biological research by allowing scientists to decode the genomes of many organisms and in particular the human genome [38]. Moreover, the following advent of high-throughput next- and subsequent generations of sequencing technologies (Gen-1, Gen-2 and Gen-3, respectively) now promise to considerably reduce the genome sequencing cost in what now seems to be a personal genomics revolution. This chapter covers the fundamental background required to understand the research in genome sequencing and assembly which is at the core of this revolution. Specifically the chapter is organized as follows: a quick introduction to DNA is first given; next, the traditional shotgun

sequencing technology is presented; then next-generation sequencing technologies are discussed especially in light of various challenges that they introduce; next, the standard assembly pipeline for large genome projects is described; finally, the assembly of the first draft of the human genome is reviewed from a historical perspective.

1.2 DNA: Deoxyribonucleic acid

Deoxyribonucleic acid (DNA) is a double-stranded polymer that contains all the fundamental building blocks for an individual's entire genetic makeup, and it is a component of virtually every cell in the human body. Specifically, it contains the genetic instructions used in the development and functioning of all known living organisms as well as the hereditary information that gets transmitted from organism to organism. DNA has a unique structure composed of two long polymers (Watson and Crick strands) of simple units called nucleotides (A,T,C,G) that run in opposite directions to each other (anti-parallel). The two polymers are held tightly together forming a double-helix shape (see figure 1.1). In the nucleus of each cell, the DNA molecule is packaged into thread-like structures called *chromosomes*. Each chromosome consists of a single piece of coiled DNA containing many genes, regulatory elements and other nucleotide sequences. In humans, each cell normally contains 23 pairs of chromosomes, where each chromosome has two homologous copies, one from the mother and one from the father (except for the sex chromosomes X and Y: females have two copies of the X chromosome, while males have one X and one Y chromosome). Because of this dual represen-

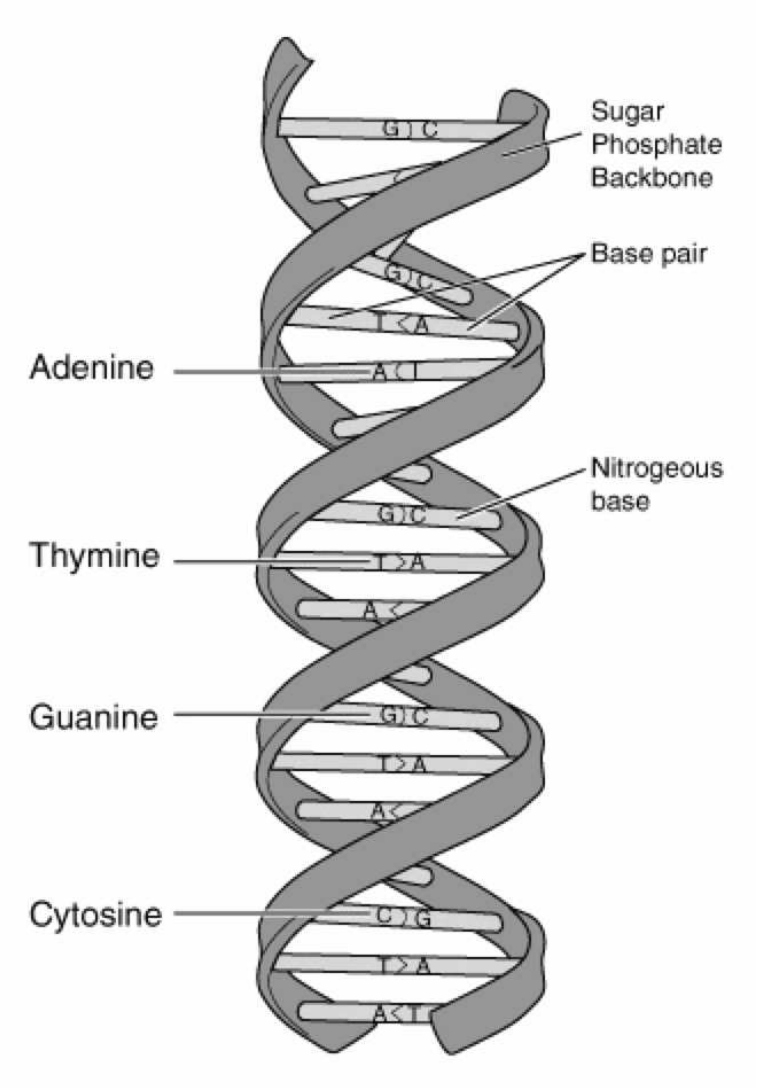


Figure 1.1: The double-helix structure of the DNA.

tation of each chromosomes, DNA is said to have an *haplotypic* structure. Note however that the current sequencing technologies do not distinguish between the two strands and the two homologous copies. So when DNA is assembled the two homologous copies are merged into a single sequence. Another important observation from the point of view of the assembly process is the following: al-

though the two strands are simply the complement of each other, because of the double-strand "antiparallel" structure of the DNA, when the fragments are sampled (using the available technology) from the DNA molecule two cases can happen: (1) if the fragment is sampled from Watson's strand it is read in the forward direction; (2) if it is sampled form Crick's strand it is read in reverse direction, and because of the complementarity rule (A-T,C-G) this fragment is in fact reverse-complemented. This properties has implications in the way the overlap between the fragments are computed (in section 2.2.1).

1.3 Shotgun Sequencing

For more than three decades, starting with the pioneering DNA sequencing work of Frederick Sanger in 1975 based on the Sanger chemistry, which is still universally used [87], every large-scale sequencing project has been organized around one single goal of overcoming the following obstacle: How can one generate the sequence of gigabases of genome as one uninterrupted string, if from the genome of an organism, it is only possible to obtain short sequence reads, limited to about 1000bp, which carry no contextual (chromosomal location or haplotypic disambiguation) information? Since the short read-lengths place a critical barrier against direct reading of long genomes, one requires an algorithmic solution to indirectly derive the full-genome sequence from an overly redundant set of read data. Thus, most sequencing projects have adopted a *shotgun sequencing strategy* [49], which, as currently practiced in many genome projects, is organized in several steps: namely, first genomic DNA of multiple copies of a target DNA

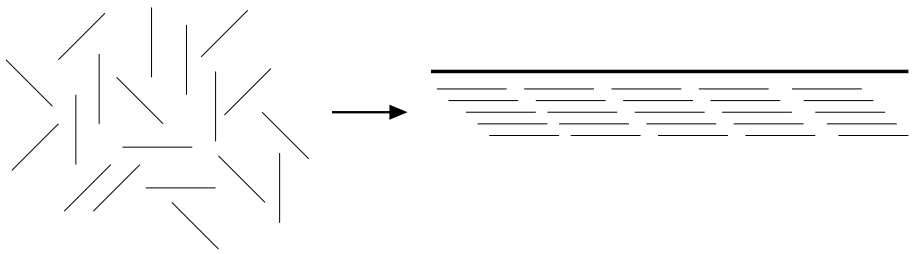


Figure 1.2: Shotgun sequencing.

molecule of an organism is sheared into a very large number of small fragments (typically 8–10 × coverage), each of whose ends are sequenced (≈ 500–1000 bp); next the resulting sequence reads are fed to a computer program, called a *sequence assembler*, whose purpose is to reconstruct a full genome sequence that can consistently and correctly explain the sequence read data (see figure 1.2).

Intuitively, the assumption is that if two sequence reads (two strings of letters produced by the sequencing machine) share a common overlapping substring of letters, then it is because they *are likely to* have originated from the same chromosomal location in the genome. The basic assumption can be made more precise by additionally taking into account the facts that the sequence reads could come from either Watson or Crick strand, and that if two strings are part of a mate-pair[1] then the estimated distance between the reads and their orientation imposes additional constraints, not to be violated. Once such overlap structures between the sequence reads are detected, then the sequence assembler places the reads in a lay-out and combines the reads together to create a consensus sequence—not unlike how one solves a complex jigsaw puzzle. In addition this assembly process is complicated by the presence of non-random structures (e.g.,

[1]Reads sequenced from the ends of the same clone.

repeated regions, rearrangements, segmental duplications) in the genome that affect the correctness of the overlaps by producing many false-positives.

1.4 Next Generation Sequencing and their challenges

Although quite reliable and considerably optimized, the Sanger process is quite expensive in cost and time. For example, the monetary cost necessary to assemble mammalian genomes was estimated to be about $12 million dollars in 2005 [63]. Since the final goal still remains personalized medicine, it remains fundamental to significantly reduce the sequencing cost (ultimately below $1000 dollars) in order to have an affordable technology for mass application.

In response to these requirements, recent advances in sequencing technology have produced a new class of massively parallel Next-Generation Sequencing (NGS) platforms such as: Illumina[2] Inc. Genome Analyzer, Applied Biosystems SOLiD[3] System, 454[4] Life Sciences (Roche) GS FLX, and Helicos[5] Heliscope Sequencer. Although they have significantly reduced the production cost and have orders of magnitude higher throughput per single run (200x coverage and higher) than older Sanger technology, the reads produced by these machines are typically shorter (35 - 500 bps). As a result they have introduced a succession of new computational challenges, for instance, the need to assemble millions of

[2]http://www.illumina.com/
[3]http://www.appliedbiosystems.com/
[4]http://www.454.com/
[5]http://www.helicosbio.com/

reads even for bacterial genomes [82]. The short read length complicates the assembly problem since repeat regions are now harder to disambiguate and so higher coverage depth is generally required. Furthermore, the assembly tools originally developed for Sanger sequencing data cannot be directly applied to NGS technologies for different reasons:

1. because of specific algorithmic choices that rely on long read lengths available with older Sanger reads;

2. because of the specific error profiles of NGS data (e.g. pyrosequencing technologies are characterized by high error rates in homopolymer regions);

3. because of the computational requirements of the vastly larger number of reads generated by NGS projects (e.g., 8 times coverage of a 3 Gbp mammalian genome requires 30 million Sanger reads but 750 million Illumina reads);

As a consequence new assembly tools have been designed to specifically deal with the new features of the data generated by NGS (see chapter 3 for a review of sequence assemblers). However developing a *universal* assembler that can handle multiple types of sequencing data still remains a noteworthy goal.

1.5 Lander-Waterman statistics

The statistical properties of the sequencing process play a critical role on the performance of an assembler. In fact, even in absence of repeats, the output

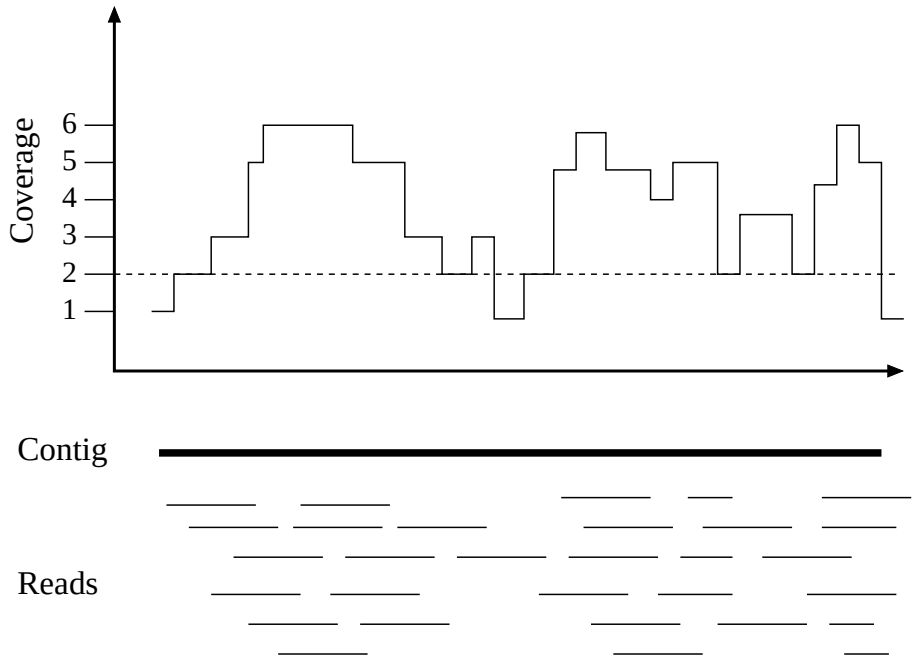


Figure 1.3: Read coverage illustration (inspired by a lecture given by Michael Schatz in 2006 at the University of Hawaii).

of a sequence assembler may consist of multiple contigs (contiguous sequence of the genome) if not enough depth of coverage is available. This phenomenon can be explained with an analogy to the water covering a sidewalk as it starts to rain: as droplets fall, the sidewalk becomes increasingly wet, though many spots remain dry for a while. Similarly, as the fragments are being sequenced, the randomness of the shearing process leads to cover successively more new sections of the original DNA not yet represented in the collection of reads (see figure 1.3). If some region of the genome are not covered, the best possible assembly consists of the collection of contigs with gaps in between representing the location of the DNA not covered by the reads.

This phenomenon was initially analyzed by Lander and Waterman [55]. Specif-

ically they approximate the *arrival* of N reads of equal length l along a genome of length G as a Poisson process whose parameters are defined as follows.

Definition 1 (Lander-Waterman parameters)**.** *Consider a genome of length G that has been uniformly randomly sampled to collect N fragments/reads each one of length l. We can define the following Lander-Waterman parameters:*

- $c = \frac{lN}{G}$*, coverage.*
- k*, size of the minimum detectable overlap: number of base pairs two fragments must have in common to ensure their overlap (overlap parameter).*
- $\sigma = 1 - \frac{k}{l}$*, fraction of a read not involved in the minimum detectable overlap; where* $\theta = \frac{k}{l}$*.*

For example $1\times$ (1 times) coverage of the human genome requires $N = \frac{cG}{l} = \frac{1(3\times10^9)}{500} = 6$ million reads. However higher coverage ($\sim 10\times$ coverage) is typically used to assembled genomic data, and in that case $N = 60$ million reads would be required.

Considering the reads in order of their arrival along the genome (from left to right), the number of contigs is the same as the number of reads that *do not overlap*. The following theorem precisely formalize this intuition.

Theorem 1 (Contigs statistics)**.** *If we model the "arrival" of N fragments of length l along a genome of length G as a Poisson process then the expected number of non-trivial contigs*[6] *and their sizes are:*

[6]contig composed of two or more reads.

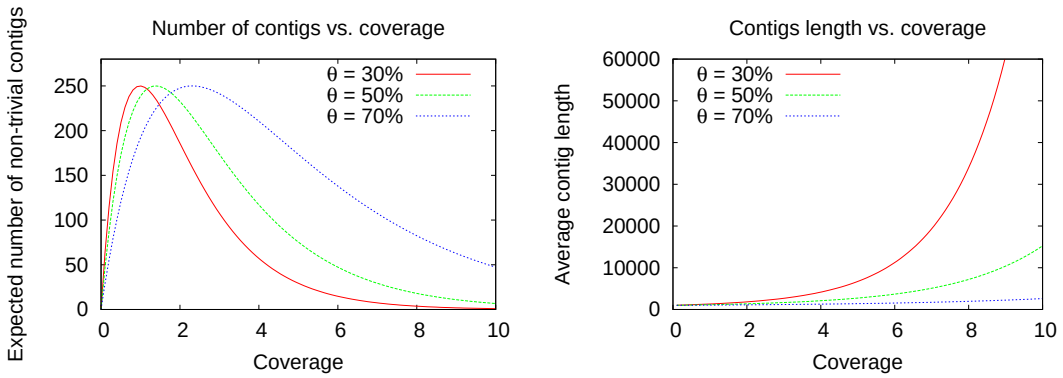


Figure 1.4: Expected number of contigs and their average length as a function of coverage for different values of the minimum detectable overlap θ.

$$E[\#\ non\text{-}trivial\ contigs] = Ne^{-(c\sigma)} - Ne^{-(2c\sigma)} \tag{1.1}$$

$$E[contig\ size] = l\left[\frac{(e^{(c\sigma)}-1)}{c} + (1-\sigma)\right] \tag{1.2}$$

Figure 1.4 shows the relation between the expected number of contigs and their average length as a function of coverage. As the coverage increases more and more contigs can be created, however when the coverage reaches a specific threshold the number of contigs starts to decrease since now the probability that a new fragment connects two previously created contigs becomes close to 1. As this process continues the average contigs' length simply increases monotonically. Note that decreasing the minimum detectable overlap θ greatly reduces the expected number of contigs and increases their lengths. However it is important to emphasize that this is only a mathematical model and in real applications smaller values of θ must be balanced with the possibility of making erroneous assemblies.

Proof. Presented below is the proof for the expected number of non-trivial contigs

(equation 1.1), while a proof for equation 1.2 can be found in [55]. The proof is based on the following simple observations: a contig stops at the last overlapping fragment with no following overlapping fragments. Thus computing the number of contigs is equivalent to counting the number of "stopper" fragments. A stopper fragment is defined as a read with no further overlaps at any of the first $l(1-\theta)$ positions, where $\theta = \frac{k}{l}$ models the notion of detectable overlap. Now note that the probability of a fragment starting at any position in the genome is:

$$Pr[\text{fragment start at position } i] = \frac{N}{G} \tag{1.3}$$

so the probability of having a stopper fragment is:

$$\begin{aligned} Pr[\text{"stopper" fragment}] = \left(1 - \frac{N}{G}\right)^{l(1-\theta)} &= \left[1 + \left(-\frac{N}{G}\right)\right]^{\frac{1}{-\left(\frac{N}{G}\right)}\left[-\frac{lN}{G}(1-\theta)\right]} \\ &\approx e^{-\frac{lN}{G}(1-\theta)} = e^{-(c\sigma)} \end{aligned} \tag{1.4}$$

The approximation is possible because $\frac{N}{G} \approx 0$ for large G. Hence, the expected number of contigs and singleton contigs are given respectively by the following equations:

$$E[\#\text{ of contigs}] = Ne^{-(c\sigma)}, \tag{1.5}$$

$$E[\#\text{ of singleton contigs}] = Ne^{-(2c\sigma)} \tag{1.6}$$

Thereof the expected number of non-trivial contigs is:

$$E[\#\text{ non-trivial contigs}] = E[\#\text{ of contigs}] - E[\#\text{ of singleton contigs}]$$
$$= Ne^{-(c\sigma)} - Ne^{-(2c\sigma)}$$

□

1.6 Trade-off in sequencing technology

There is a critical trade-off between sequence reads and the number of reads that can be generated by current sequencing machines (coverage). Ideally we would like to generate very long reads with high coverage for de novo assembly projects, but currently no technology generates reads longer then 1Kb in length. Some technologies, e.g., Life Technologies[7] and Pacific BioSystems[8], have announced new single molecule sequencing (SMS) technology that can combine virtually unlimited continuous long read lengths with unmatched accuracy to deliver targeted genomic sequence data in a matter of hours. However there is a lot of speculation whether these promises are too optimistic and whether we will see very long sequence reads ($> 10kb$) in the near future. On the other side, next-generation sequencing technologies have been able to significantly increase the throughput (and therefore the coverage) while paying a high price by generating much shorter reads. Figure 1.5 depicts the trade-off scenario. However even very high coverage is not the final solution to the shorter read length and long-range data (mate-

[7] http://www.lifetechnologies.com/
[8] http://www.pacificbiosciences.com/

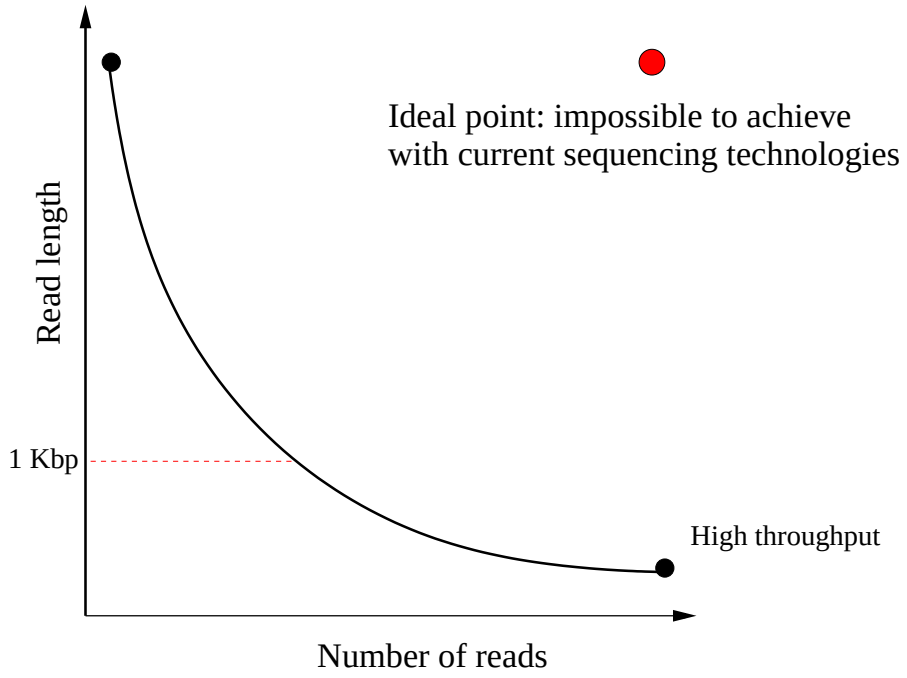


Figure 1.5: Trade-off between read length and coverage.

pairs, optical maps, etc) is necessary to correctly assemble the most complex genome structures. Although initially unavailable, paired-end sequencing has now become a common routine for almost all of the next-generation sequencing technologies.

1.7 Assembly Pipeline

The process of reconstructing the complete genome sequence given the input sequence reads and, if available, additional long-range information (mate-pairs, optical maps, etc.), is typically carried out in multiple steps (sub-problems). Each sub-problem is tackled using a specific module and all the modules are

sequentially performed in a pipeline. Note that in the following we will focus on describing the modules that involve the use of computational approaches, while omitting the steps that involve laboratory experiments. All together this assembly pipeline is the general procedure for Whole-Genome Shotgun (WGS) Sequencing, but we emphasize that the specific procedure used at the various genomic centers may differ in details.

1. **Fragment Readout or Base-calling** - The first step of the assembly pipeline consists of determining each fragment sequence using automatic base-calling software. The base-caller task is to interpret the signal intensities generated by the sequencing machines and *call* a base for each position in the sequence. In the case of longer Sanger reads, Phred [25] was the standard base-caller used to analyze the fluorescence "trace" data generated by the DNA sequencer. New base-callers have been designed for next-generation sequencers that use sophisticated signal processing methods such us statistical learning (BayesCall [42]), supervised learning and support vector machine (Alta-Cyclic [24], Ibis [51]), and model-based clustering and information theory (Rolexa [85]). More recently, we have proposed a novel unified re-sequencing framework, TotalRecaller [62], that has the ability to concurrently perform base-calling, alignment and SNP detection. In the last chapter of the thesis, we will demonstrate the advantages of a complete pipeline integrating Base-Calling (TotalReCaller) with assembly (SUTTA) in a Bayesian manner.

2. **Trimming low-quality sequences** - The sequence reads often contain

poor quality regions which, if removed, can lead to more accurate sequence assembly. However this step is optional, for example one may rely on the overlapper (next step) to filter false overlaps using the quality values.

3. **Overlap Detection/Pairing** - The specific algorithmic approaches for this task have evolved throughout the years, as increasingly more complex sequencing projects were tackled through the shotgun method and next-generation sequencing machines were developed. The earliest algorithms involved either iteratively aligning each read to an already generated consensus or comparing all the reads against each other [77]. Most recently the detection of read overlaps involves sophisticated techniques meant to reduce the number of pairs of reads being analyzed. For example the UMD overlapper [84] keeps the number of repeat-induced spurious overlaps small and it builds the initial overlapping-phase of the algorithm with a reasonably small number of k-mers, whose cardinality is optimized by an order of magnitude through the use of minimizers. Because of the higher throughput of NSG data the standard approach used for finding overlaps is instead exact matching which makes use of suffix- and prefix- trees or arrays. These are data-structures designed to efficiently store all the suffixes or prefixes of a particular string. In order to identify all reads that overlap a particular read r, it is sufficient to identify the reads whose suffixes match a prefix of r. This approach allows one to perform the overlap computation for millions of reads both in a time and memory efficient manner although sequencing errors are not tolerated. The output of this module is a set of fragment

pairs with associated overlap score.

4. **Fragment Assembly** - Given the set of fragments/reads and their pairwise overlaps (and possible long range information), this step generates a layout or arrangement of the reads that is consistent with the overlap information and satisfies the long range information constraints. This is the most complicated step in the pipeline and requires sophisticated techniques and heuristics which are the central problems addressed by this thesis. In particular, this problem can be formulated in graph-theoretic terms and it can be shown to belong to the class of $\mathcal{NP}$-complete problems (see chapter 2). Several assembly paradigms have been developed to efficiently tackle the fragment assembly problem (see chapter 3). The output of this step is a set of assembled layouts (arrangements of the reads) for the contigs.

5. **Consensus Generation** - Once a layout (or set of layouts) is generated, the sequence of base pairs nucleotides for each layout is computed using a consensus program. If overlaps are detected using exact match, then the consensus generation is a trivial task, however, if dynamic programming is used for overlap detection, the consensus generation requires solving a multiple sequence alignment problem [44].

6. **Scaffolding** - The scaffolding process groups the contigs together into subsets with a known order and orientation. Researchers generally infer the relationships between contigs from mate-pair information. Similarly to step 4, this problem can be also formulated in graph-theoretic terms where each

node in the graph is a contig and a directed link between two contigs exists if there are mate-pairs bridging the gap between them. The goal is to find a consistent orientation for all nodes in the graph according to the mate-pair information. Because of errors in the pairing data and possible mis-assembly errors in the contigs, this problem can also be shown to be $\mathcal{NP}$-complete; several greedy and heuristic strategies have been proposed. Many sequence assemblers include a scaffolding step, however stand-alone options are also available (e.g., Bambus [81], SSPACE [14]). The output of this step is a set of ordered contigs and estimated distances between them.

7. **Assembly Validation** - In practice the assembly problem is never an error-free process and even the most sophisticated assemblers are affected by mis-assembly errors. So it is important to include in the pipeline a validation step that checks the quality of the assembled contigs. Unfortunately no standardized method yet exists and most reported measures of assembly quality are aggregate measures, such as the number and sizes of contigs, which do not account for the possibility of misassemblies. Thus they are only marginally useful. If a reference genome is available, the contigs can be aligned to the reference in order to identify region that have been mis-assembled. However, if the true layout is unknown, which is common in *de novo* sequencing projects, then additional data must be used to validate the contigs. For example, physical maps provide markers (with approximate order and/or distance) that can be used to validate large contigs. Similarly, the sequence of a closely related organism can be used to

confirm areas where we expect not to find significant discrepancy (or divergence). In the absence of any other types of information, clone mates (also known as mate-pairs or paired-ends) have been widely used to detect assembly errors by checking areas of the genome that violate the orientation and distance constraints imposed by the clone mates. Some tools have recently appeared that perform automatic validation of contigs [80]. This thesis contributes many improvements to this step by introducing a novel metric, which satisfactorily captures the trade-offs between contig's length and quality (see chapter 5). The typical output of this step is a quality value for each contig together with pointers to locations in the contigs that might be mis-assembled.

8. **Finishing** - In practice, imperfect coverage, repeats, and sequencing errors cause the assembler to produce not one but hundreds or even thousands of contigs. After scaffolding, the contigs are oriented and the sequence gaps between them now need to be filled with their genomic sequence. The task of closing the gaps between contigs and obtaining a complete molecule is called finishing. Although they represent genuine gaps in the sequence, researchers can retrieve the original clone inserts spanning the gap and use a straightforward walking technique to fill in the sequence. However, filling these gaps involves a large amount of manual labor and complex laboratory techniques, so any improvement in assembly that could reduce the cost of this step would have significant importance. The output from this final step is the whole genome sequence.

1.8 History of the assembly of the Human Genome

Among the thousand genomes that have been sequenced and assembled in the last 20 years, the human genome has a very intriguing history. For obvious reasons, we have been particularly interested in decoding our own DNA. The potential discoveries, applications, and the sheer glory of such an achievement has motivated scientists all over the world to compete fiercely against each other. Here we briefly outline the major historical events that led to the first draft of the Human genome.

[1990] The Human Genome Project was launched through funding from the US National Institutes of Health (NIH) and Department of Energy, whose labs joined with international collaborators and resolved to sequence 95% of the DNA in human cells in just 15 years. This joint effort was named the International Human Genome Sequencing Consortium (IHGSC). Researchers at IHGSC proposed that a feasible assembly strategy should follow the BAC-by-BAC hierarchical method where the genome is first broken up into a collection of large overlapping in-vivo-clonable fragments (between 160 and 200 Kb) – called Bacterial Artificial Chromosomes or BACs. Each BAC would be independently assembled by shotgun strategy and then mapped together using restriction-finger-print-based overlaps. There was also a YAC map but it had too many errors due to chimerisms in the clones.

[1998] A new private venture was launched to sequence the human genome: the enterprise - named Celera Genomics - aimed to create its own database of human genomic data, which users would be able to subscribe to for a fee. In contrast to the IHGSC and despite popular skepticism, the Celera Genomics opted for the Fred Sanger's "whole genome shotgun" method, skipping the mapping phase of the BAC-by-BAC hierarchical process.

[2000] On 26 June 2000 the public and private enterprises both announced that they had completed their respective draft genome sequences.

[2001] Each effort published an account of its draft human genome sequence: Celera's effort appeared in Science [100], and the International Human Genome Sequencing Consortium's effort was published in Nature [38].

[2003] Two years ahead of schedule, with contributions from countless scientists from 20 institutions across the globe, the International Human Genome Sequencing Consortium announced that they had completed the gold-standard reference human genome, according to the guidelines of the original Human Genome Project, with 99.99% accuracy [39].

What about quality? Although it is widely believed that the Herculean task of the Human Genome project (HGP) was completed in 2003, 13 years after its initial project announcement, it is legitimate to ask: how well did we do? "Of particular interest are the relative rates of mis-assembly (sequence assembled in the wrong order and/or orientation) and the relative coverage achieved by the

three protocols" [91]. Unfortunately, prior to 2003, the IHGSC group were alone in having published assessments of the rate of misassembly in the contigs they produced (see [40] on subsequent assembly analysis and comparison). Using artificial data sets, they found that, on average 10 per sent of assembled fragments were assigned the wrong orientation and 15 per cent of fragments were placed in wrong order by their protocol [47]. A more recent article, entitled "Revolution Postponed" in Scientific American [31] asserted, "The Human Genome Project has failed so far to produce the medical miracles that scientists promised. Biologists are now divided over what, if anything, went wrong...". Due to these and other related recent events, genome assembly is again receiving a lot of attention from the genomic community. In particular, the central problem is the development of new assembly and validation tools that can overcome the limitations of previous sequence assemblers thus leading to more accurate genome sequences: this thesis will focus on both aspects of the problem by contributing not just to a novel sequence assembler but also to a better metric for assembly validation.

Some recent advances. The history of the assembly of the human genome did not stop with the successful completion of the first draft in 2001. Several other teams of researchers have continued on a similar journey tackling these and other massive assembly problems. Below is a list of several such major efforts.

[2007] A team of researchers mostly from the Craig Venter Institute[9] published an updated version of the human genome produced from 32 million random DNA

[9]http://www.jcvi.org/

fragments sequenced using Sanger technology [57]. Similar to the Celera effort, the whole-genome shotgun sequencing method was used also in this case. In particular they developed a modified version of the Celera assembler to facilitate the identification and comparison of alternate alleles within the individual diploid genome. Comparison of this genome and the National Center for Biotechnology Information human reference assembly revealed more than 4.1 million DNA variants, encompassing 12.3 Mb.

[2008] The first human genome sequenced by next-generation technologies was published, using massively parallel sequencing in picolitre-size reaction vessels [102]. This genome belonged to James D. Watson, one of the co-discoverers of the structure of DNA (with Francis Crick) in 1953. The assembly results seem to agree well with the previous results published in 2007 by traditional sequencing methods [57].

[2009] First human genome assembled using next-generation short read data [92]. Specifically 3.5 billion paired-end reads from the genome of an African male publicly released by Illumina, Inc. were assembled using a new parallel assembler based on a distributed representation of a De Bruijn graph. Approximately 2.76 million contigs ≥ 100 base pairs (bp) in length were created with an N50 size of 1499 bp, representing 68% of the reference human genome. Also in this case the analysis of the assembled sequences revealed polymorphic and novel sequences not present in the initial human reference assembly.

[2010] The second human assembly using next-generation short read data was published using the assembler ALLPATHS-LG [29]. Using data from the Illumina GAII and HiSeq sequencers, a team from the Broad Institute[10] of MIT created an assembly of comparable quality to the one previously obtained using capillary-based sequencing (base accuracy of $\geq$ 99.95%, N50 contig length of 24 kb, and N50 scaffold sizes of 11.5 Mbp) and it has been described as the current best human de novo assembly.

[10] http://www.broadinstitute.org/

Chapter 2

Sequence Assembly: Problem and Complexity

2.1 Introduction

Because of its natural connection to a well known $\mathcal{NP}$-complete combinatorial optimization problem (shortest superstring problem), for many years the sequence assembly problem (SAP) has been investigated using rather simple string and graph-theoretic formulations. This chapter will address such problems through the following sequence of steps: (1) the standard formulations of the SAP and their complexity results; (2) solutions in terms of these formulations which are infeasible/intractable in the context of biology; (3) a newly proposed formulation (more faithful to biology) of the problem as a constrained optimization problem and the complexity issues in this new framework. The chapter also contains a probabilistic analysis of unique reconstruction for random DNA sequences.

2.2 The dovetail-path framework

The first attempt to mathematically formalize the sequence assembly problem is due to Myers (1995) in his seminal paper [67] where the *dovetail path framework* was first introduced. An essential set of definitions from this notational framework is given below.

2.2.1 Basic definitions: reads, overlaps and layouts

The output of a sequencing project consists of a set of reads (or fragments) $F = \{r_1, r_2, \ldots, r_N\}$, where each read r_i is a string over the alphabet $\Sigma = \{A, C, G, T\}$. To each read is associated a pair of integers $(s_i, e_i), i \in [1, |F|]$ where s_i and e_i are respectively the starting and ending points of the read r_i in the reconstructed string R (to be computed by the assembler), such that $1 \leq s_i, e_i \leq |R|$. The order of s_i and e_i encodes the orientation of the read (whether r_i was sampled from Watson or Crick strand).

The overlaps (best local alignment) between each pair of reads may be computed using the Smith-Waterman algorithm [94] with match, mismatch and gap penalty scores dependent on the errors introduced by the sequencing technology. Exact string matching is instead typically used for short read from next generation sequencing, since they usually provide high coverage, thus allowing tolerance for increased false negatives. Note also that by restricting the problem to exact matches only, the time complexity of the overlap detection procedure is reduced from a quadratic to a linear function of the input size. The complete description of an overlap π is given by specifying:

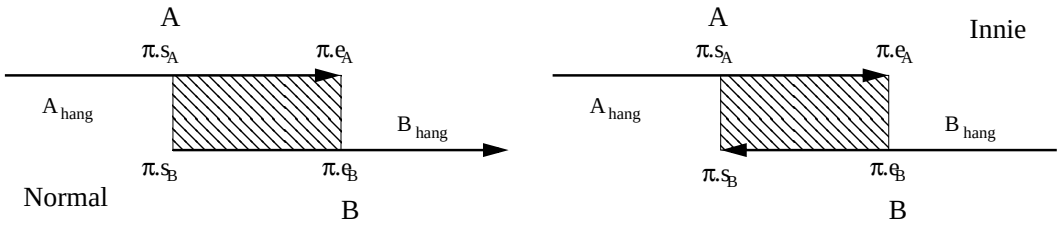


Figure 2.1: Two possible overlaps (illustration): left overlap is *normal* (both reads pointing to the same forward direction) right overlap is *innie* (the second read B is reverse complemented and is pointing in the backward direction); The suffix predicate for the left (normal) overlap is s.t. $\mathit{suffix}_\pi(A) = true$ and $\mathit{suffix}_\pi(B) = false$.

1. the substrings $\pi.A[\pi.s_A, \pi.e_A]$ and $\pi.B[\pi.s_B, \pi.e_B]$ of the two reads that are involved in the overlap;

2. the offsets from the left-most and right-most positions of the reads $\pi.A_{hang}$ and $\pi.B_{hang}$;

3. the relative directions of the two reads: Normal (N), Innie (I);

4. a predicate $\mathit{suffix}_\pi(r)$ on a read r s.t.:

$$\mathit{suffix}_\pi(r) = \begin{cases} true \;\; \textit{iff} \;\; \text{suffix of } r \text{ participates in the overlap } \pi \\ false \;\; \textit{iff} \;\; \text{prefix of } r \text{ participates in the overlap } \pi \end{cases} \tag{2.1}$$

Figure 2.1 illustrates two possible overlaps. Note that a right arrow represents a read in forward orientation, conversely a left arrow represents a read that is reverse-complemented. Because of the double-stranded nature of the DNA molecule, each read can be sampled from either the Watson or Crick strands and they have different orientation. This formulation gives rise to a taxonomy of

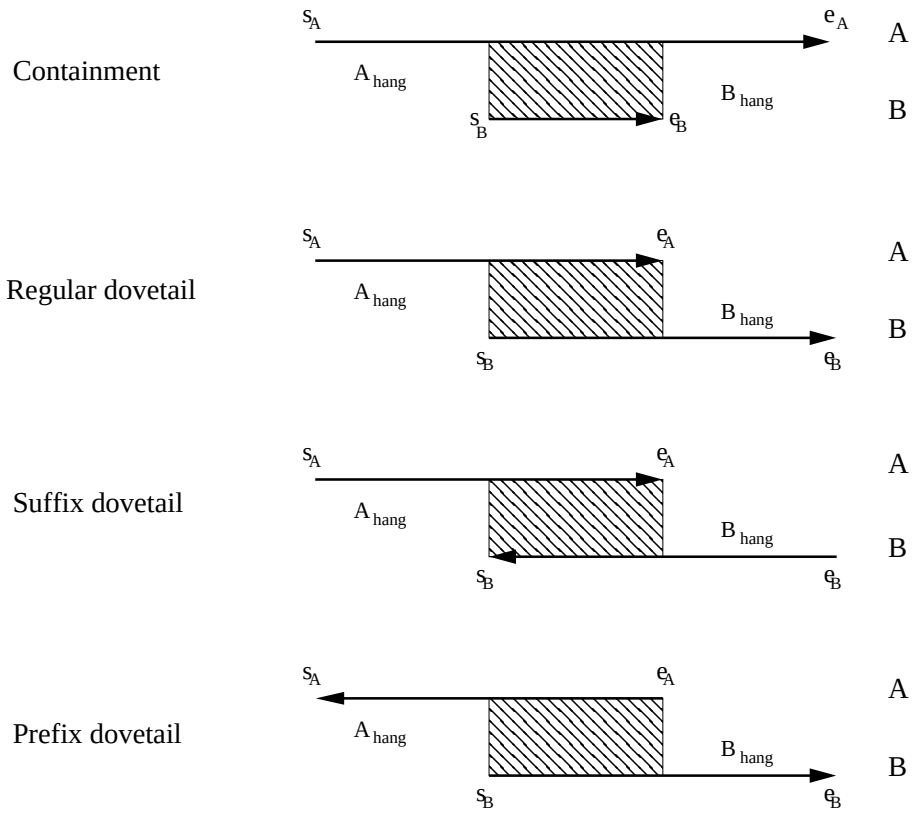


Figure 2.2: Overlap taxonomy.

overlaps illustrated in Figure 2.2.

Definition 2 (Layout)**.** *A layout L induced by a set of reads $F = \{r_1, r_2, \ldots, r_N\}$ is defined as:*

$$L = r_{j_1} \overset{\pi_1}{\rightleftharpoons} r_{j_2} \overset{\pi_2}{\rightleftharpoons} r_{j_3} \overset{\pi_3}{\rightleftharpoons} \cdots \overset{\pi_{N-1}}{\rightleftharpoons} r_{j_N}. \tag{2.2}$$

Informally a layout is simply a sequence of reads connected by overlap relations. Note that the order of the reads in L is a permutation of the reads in F. The previous definition assumes that there are no *containments*[1]; without any loss of correctness in the base-pair sequences that can be generated, contained

[1]Reads that are proper subsequences of another read.

reads can be initially removed (in a preprocessing step) and then reintroduced later after the layout has been created. However, note that their reintroduction is important to provide additional support for mate-pair constraints (if available). Out of all the possible layouts (possibly, exponential in the number of reads), it is imperative to efficiently identify the ones that are consistent according to the following definition:

Definition 3 (Consistency Property)**.** *A layout L is* **consistent** *if the following property holds for $i = 2, \ldots, N-1$:*

$$\overset{\pi_{i-1}}{\rightleftharpoons} r_{j_i} \overset{\pi_i}{\rightleftharpoons} \text{ iff } \quad \mathit{suffix}_{\pi_{i-1}}(r_{j_i}) \neq \mathit{suffix}_{\pi_i}(r_{j_i}). \tag{2.3}$$

The consistency property imposes a directionality to the sequence of reads in the layout. The directionality of each internal read in the layout must be preserved. Figure 2.3 shows an example of layout associated to 7 overlapping reads. The estimated start positions for each read are given by the formula:

$$sp_1 = 1, \quad sp_i = sp_{i-1} + \pi_{i-1}.hang_{r_{j_{i-1}}} \quad \text{if } i > 1 \tag{2.4}$$

It should be clear at this point why this framework is called the *dovetail-path framework*: layouts consist of a sequence of dovetail overlaps that satisfy the consistency property with containment overlaps hanging off the main dovetail-path.

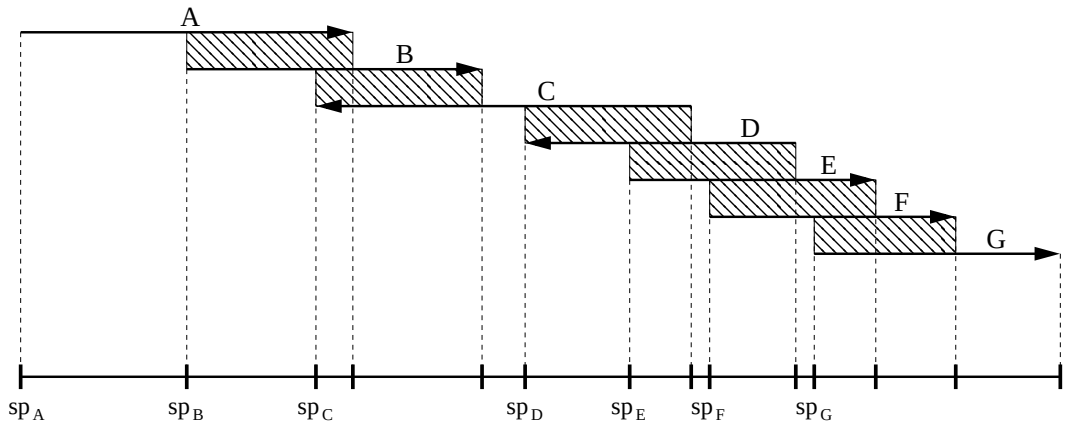


Figure 2.3: Example of layout for a set of reads $F = \{A, B, C, D, E, F, G\}$ with overlaps $\pi^N_{(A,B)}, \pi^I_{(B,C)}, \pi^N_{(C,D)}, \pi^I_{(D,E)}, \pi^N_{(E,F)}, \pi^N_{(F,G)}$.

2.2.2 Min-length reconstruction theorem

Appealing to parsimony, we are typically interested in a layout whose length is minimal (although we will see that this assumption may lead to biologically incorrect solutions). The following theorem shows the correlation between the length of a layout and the sizes of its overlaps. Let us define the length of an overlap to be the average length of the two overlapping substrings:

$$length(\pi) = \frac{(|\pi.s_A - \pi.e_A| + |\pi.s_B - \pi.e_B|)}{2}. \tag{2.5}$$

Note that $|\pi.s_A - \pi.e_A|$ and $|\pi.s_B - \pi.e_B|$ can have different values when the overlaps are computed using the Smith-Waterman alignment algorithm, though they are the same if exact match is used. Let us define the weight of a layout $|L|$

to be the sum of the lengths of its overlaps:

$$weight(L) = \sum_{\pi \in L} length(\pi) \tag{2.6}$$

then the following theorem holds [97, 98]:

Theorem 2 (Min-length reconstruction)**.** *A layout of maximum weight results in a reconstruction of minimum length.*

Proof. First note that:

$$|L| = sp_n + |r_N| = \sum_{i=1}^{N-1} \pi_i.hang_{r_i} + |r_N| \tag{2.7}$$

using the facts that:

1. $sp_1 = 1, sp_i = sp_{i-1} + \pi_{i-1}.hang_{r_{i-1}}$, *if* $i > 1$,
2. $\pi_i.hang_{r_i} \approx |r_i| - length(\pi_i)$,
3. $length(\pi_i) \approx |g|$ when π is a containment edge and g is the contained fragment,

it follows that:

$$|L| = \sum_{r \in F} |r| - \underbrace{\sum_{\pi} length(\pi)}_{weight(L)} . \tag{2.8}$$

But since the second sum is the weight of the layout, maximizing weight minimizes length. □

2.3 Shortest Superstring Problem (SSP)

Researchers first approximated the shotgun sequence assembly problem as one of finding the shortest common superstring of a set of sequences. This formulation was favored because of the results of the previous theorem and the availability of efficient algorithms to solve the SSP.

Definition 4 (Shortest Superstring Problem)**.** *Given a set of strings or sequences* $S = \{r_1, r_2, \ldots, r_n\}$ *find the shortest string* R *(reconstruction) such that* $\forall i$*,* r_i *is a substring of* R*.*

This shortest common superstring (SCS) formulation led to a simple theoretical abstraction, but by being oblivious to how biological sequences are organized by evolution, it often yielded biologically implausible and incorrect solutions. Its inability to correctly model the assembly problem is owed to a multitude of reasons, but primarily because:

1. it does not account for possible errors arising during the process of sequencing the fragments,

2. it does not model fragment orientation (the sequence source can be one of the two DNA strands, Watson or Crick), and

3. most importantly, it fails in the presence of *repeats*, as it encourages repeat-induced compressions.

To emphasize the last point it is of interest to note Richard Karp's statement from 2003 [43]: *The shortest superstring problem [is an] an elegant but flawed*

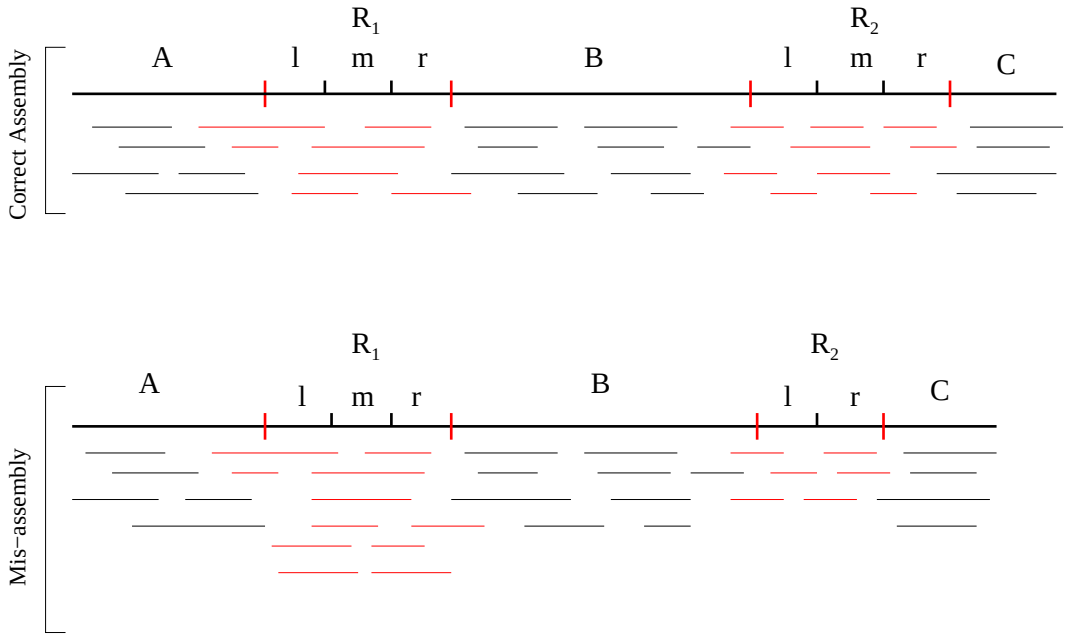


Figure 2.4: Example of compression: the two copies of repeat (R_1 and R_2) are compressed into one leading to a shorter but misassembled sequence.

abstraction: [since it defines assembly problem as finding] a shortest string containing a set of given strings as substrings. Figure 2.4 shows an example of the kind of errors that such formulation could lead to: since strings contained inside a repeat regions cannot be disambiguated, multiple copies of a repeat are compressed into a single one.

Because of the theoretical computational intractability ($\mathcal{NP}$-completeness [27]) of the SSP, most of the approaches for genome sequence assembly have resorted to greedy and heuristic methods that, by definition, restrict themselves to near-optimal solutions, where the "nearness" may be guaranteed within a multiplicative competitiveness factor. The best known greedy algorithm for the SSP has an approximation factor of $2\frac{2}{3}$ [8].

2.4 Graph-Theoretic formulation

Since the SSP formulation was not able to correctly model the sequence assembly problem, researches opted for more sophisticated graph-theoretics ideas. Specifically, graph-theoretic approaches convert sequence assembly into solving specific problems for general graphs constructed using the overlap information of the input set of reads. This mapping has the advantage of allowing us to apply the large collection of algorithms and heuristics that have been developed in graph theory for many decades. However, this formulation still suffer from the same problems and limitations of the SSP model, since it can produce mis-assembly errors (as shown later). In this section we introduce the two most used graphical models for the sequence assembly problem: *String Graph* and *De Bruijn graph*. But before formally defining them, we need to give a minimal set of definitions.

2.4.1 Strings, Overlaps and Overlap Graph

Let x and y be two strings over the alphabet Σ. Let us denote the length of x by $|x|$. The i^{th} character of x is denoted by $x[i]$. If $1 \leq i \leq j \leq |x|$, we use $x[i, j]$ to denote the substring of x starting at position i and ending at position j. Given two strings x and y over the alphabet Σ, we say that there is an *overlap* between x and y, and we denote it with $x \rightleftharpoons y$, if there exists a suffix of x that matches[2] a prefix of y. Let us denote with $o(x, y)$ the length of the longest such match.

Definition 5 (Overlap Graph)**.** *Given a set of strings* $S = \{r_1, r_2, \ldots, r_n\}$ *and*

[2]The matching does not have to be perfect and it can be approximated allowing up to ϵ percent error on real data.

a minimum overlap threshold value k, the overlap-graph for S is a weighted bidirected graph $OG^k = (V, E)$ where:

- $V = S = \{r_1, r_2, \ldots, r_n\}$;
- $E = \{(r_i, r_j) : (r_i \rightleftharpoons r_j) \wedge o(r_i, r_j) \geq k, r_i, r_j \in V\}$;
- *the weight of each edge (r_i, r_j) is $w(r_i, r_j) = |r_j| - o(r_i, r_j)$.*

The *overlap graph* represents all the inferable relationships between the strings in the set S. Note that $|r_j| - o(r_i, r_j)$ is the length of the overhang[3] for string r_j, Since each vertex/string r_i has an orientation, every edge has two orientations, one with respect to each of its endpoints. Because the graph is bidirectional, we need to describe how to explore the nodes of the graph to generate the set of *valid* paths.

Definition 6 (Path validity)**.** *A path $P = \langle r_1 \overset{e_1}{\rightleftharpoons} r_2 \overset{e_2}{\rightleftharpoons} r_3 \overset{e_3}{\rightleftharpoons} \cdots \overset{e_{m-1}}{\rightleftharpoons} r_m \rangle$ in G is valid if $\forall i, 2 \leq i \leq m-1$, e_{i-1} and e_i have opposite directions at r_i.*

Note that this definition is equivalent to the consistency property for a layout. In order to traverse a node in the graph we need that the entry edge and the exit edge have opposite directions at the node. So we are allowed to enter a node x even if the edge e_i is pointing out of the node as long as we use an edge e_j with opposte direction to e_i when we exit the node (see figure 2.5 for an example of overlap graph).

Given any path P in the overlap graph, we associate a *path-string* to P that consists of the concatenation of the strings according to the order in the path,

[3]The size of the read portion that is not involved in the overlap.

where only one copy of the overlap is kept. Clearly the weight of a path P is given by the sum of the weights of its edges:

$$w(P) = \sum_{(r_i,r_j)\in P} w(r_i, r_j) = \sum_{(r_i,r_j)\in P} (|r_j| - o(r_i, r_j)) \tag{2.9}$$

Note that because of the weight function associated to the edges of the graph, a path of minimum weight defines a path-string of minimum length.

2.4.2 String Graph

The size of the overlap graph can be dramatically reduced by a sequence of transformations whose goal is to eliminate edges that can be *transitively* inferred.

Definition 7 (transitively inferable edge)**.** *If $x \overset{e_1}{\rightleftharpoons} y \overset{e_2}{\rightleftharpoons} z$ and $x \overset{e_3}{\rightleftharpoons} z$ are "mutually consistent" overlaps among nodes x, y and z then the edge e_3 is said to be transitively inferable from the sequence of edges e_1 and e_2.*

Informally the overlap between strings x and z is implied by the concatenation of the overlaps between x, y and z. It is important to note the edges must be mutually consistent: entry edge and the exit edge must have opposite directions. The *string graph* is a particular graph where all the contained string and transitivity inferable edges are removed [68].

Definition 8 (String Graph)**.** *Given a set of strings $S = (r_1, r_2, \ldots, r_n)$ and a minimum overlap threshold value k, the string graph SG^k for S is obtained from the overlap graph OG^k by removing contained strings (strings that are substrings of other strings) and transitively inferable edges.*

Such transformation can be computed in polynomial time using the algorithm proposed by Myers in [68]. In order to correctly apply the transitivity reduction step to the graph, it is important to first mark all transitivity edges and then remove all marked edges. This is because this process is not Church-Rosser [20] and any arbitrary strategy would fail to remove some of the transitivity edges.

Equipped with the notion of string graph, the sequence assembly problem can be formulated as follows:

Definition 9 (Sequence Assembly Problem (SAP_1))**.** *Given a set of fragments or reads $S = (r_1, r_2, \ldots, r_n)$ and a minimum overlap threshold k, the Sequence Assembly Problem (SAP) is the problem of finding a Hamiltonian Path in the string graph SG^k for S such that its weight is minimal.*

Note that we seek a Hamiltonian Path because we would like all the reads to be part of the assembly without repetition. The problem is clearly a special case of the Traveling Salesman Problem (TSP) with the following two differences: (1) instead of a Hamiltonian cycle we look for a Hamiltonian path; (2) we work with bi-directed graphs instead of undirected or directed graphs. However, for circular genomes (such as plasmids and bacterial genomes), the first difference does not apply anymore as we need to find a Hamiltonian cycle as well.

Note that this formulation differs from the one presented in [70]. Specifically Nagarajan and Pop define the sequence assembly problem as one of finding a generalized Hamiltonian path (every node is visited at least once) of minimum weight in the string graph of the reads. This is in accordance with the solution proposed in [68] where they seek a cyclic tour. In such a model each edge has

assigned to it a selection constraint c that dictates how many times the edge should appear in the target solution. Specifically, there are three cases:

1. *exact* edge ($c = 1$),
2. *required* edge ($c \geq 1$) and
3. *optional* edge ($c \geq 0$).

The argument for allowing an edge to appear multiple times in the solution is to model the repeated regions of the genome. If an edge belongs to a repeat then it should be possible to reuse it in when the second (or more) copy of the repeat are assembled. However, notice that if one assumes uniform coverage of the genome, it is reasonable to presume that the other copies of the repeat are covered by a different set of homologous reads. The problem for the assembler is then how to avoid compressing these reads together into one single copy of the repeat.

Before analyzing the complexity of this formulation it is important to observe that this graph-theoretical framework suffers from a similar kind of problem as the shortest superstring approach. Figure 2.5 shows an example of a string graph where all the possible Hamiltonian paths create mis-assembly errors due to the presence of a repeat. The compression error is due to the fact that repeats can induce false positive transitivity edges. For example consider the reads 3, 7 and 8 in figure 2.5, we have that $7 \rightleftharpoons 3$, $3 \rightleftharpoons 8$ and $7 \rightleftharpoons 8$, so the edge $7 \rightleftharpoons 8$ is removed with the negative effect of merging together reads that belong to two different copies of the repeat R_2. In particular, after removal of the transitivity edges, there

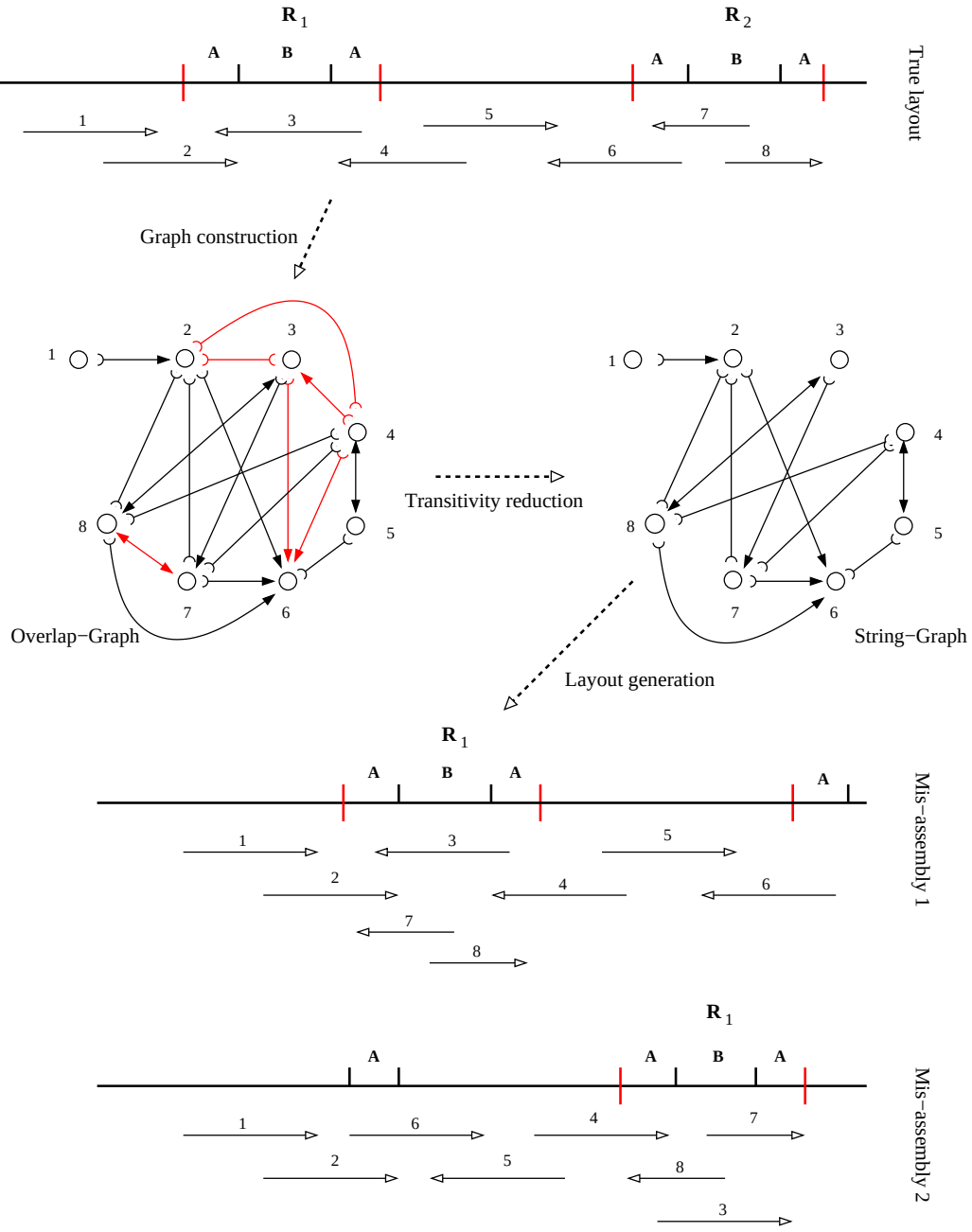


Figure 2.5: Example of mis-assembly using a string graph: the removal of the transitivity edges (in red) produces a string graph where every (Hamiltonian) paths through all nodes creates misassemblies. The layout for two of these paths are shown at the bottom: the first one with errors due to compression and the second with both compression and inversion errors.

is more then one path that traverses all the nodes and it always produces mis-assembled layouts. Note that edge $2 \rightleftharpoons 6$ cannot be removed because, although there are edges $2 \rightleftharpoons 7$ and $7 \rightleftharpoons 6$, the edge directions at node 7 do not match thus obstructing it from being traversed (the edges are not *mutually consistent*).

This example also shows another problem inherent to this framework. Even if it would be possible to efficiently compute the Hamiltonian path, the string graph might have many different Hamiltonian paths (as in this example) of minimal length and all these paths represent a possible reconstruction of the genome. Additional long-range information (e.g., mate-pais, optical maps, etc.) must be then used to resolve these ambiguities.

The problem of finding a minimal weight Hamiltonian path in a directed or undirected graph is known to be $\mathcal{NP}$-complete. Since directed graphs are special types of bidirected graphs, it follows that the Sequence Assembly Problems is also $\mathcal{NP}$-complete:

Theorem 3. *The Sequence Assembly Problem is $\mathcal{NP}$-complete.*

2.4.3 De Bruijn graph

A different approach was first developed by Idury and Waterman in 1995 [37] and later expanded by Pevzner et al. in 2001 [78], which is based on the notion of De Bruijn graph. In graph theory, an n-dimensional De Bruijn graph of m symbols is a directed graph representing overlaps between sequences of symbols. It has m^n vertices, consisting of all possible length-n sequences of the given symbols. For example, given a set of m symbols $S = \{s_1, s_2, \ldots, s_m\}$, the set of nodes are:

$$V = S^n = \{(s_1, \ldots, s_1, s_1), (s_1, \ldots, s_1, s_2), \ldots, (s_1, \ldots, s_1, s_m),$$
$$(s_1, \ldots, s_2, s_1), \ldots, (s_m, \ldots, s_m, s_m)\}$$

Any two nodes v_1 and v_2 have a direct edge between them if the $n-1$ suffix v_1 is equal to the $n-1$ prefix of v_2.

Unlike the string graph formulation, in a De Bruijn graph the notions of nodes and edges are in some sense the dual of the overlap graph. In the context of the assembly of sequence reads, a De Bruijn graph is formally defined as follows:

Definition 10 (De Bruijn Graph)**.** *Given a set of strings $S = (r_1, r_2, \ldots, r_n)$ and a minimum overlap threshold value k, the De Bruijn graph for S is a directed graph $BG^k = (V, E)$ where:*

- $V = \{d \in \Sigma^k \mid \exists i \text{ s.t. } d \text{ is a substring of } r_i \in S\}$;
- $E = \{(d_i, d_j) : \text{if the } k-1 \text{ prefix of } d_i \text{ is a suffix of } d_j\}$;

Informally, the set of vertices of BG^k is the set of k-mers for the set of strings S, and the edges correspond to their perfect $k-1$ overlap. Clearly every read $r_i \in S$ is translated into a path composed of $(|r_i| - k)$ nodes. Let us call such a path a *walk* and define it $w(r_i)$. Also note that the graph is directed (not bidirected as in the case of the string graph) and there is no weight associated to the edges (the overlap weight is $k-1$ for all the edges and it can be disregarded). Figure 2.6 shows an example of De Bruijn graph for the list of strings $L = \{AAA, AAC, ACA, CAC, CAA, CGC, GCG\}$ with parameter $k = 2$. We

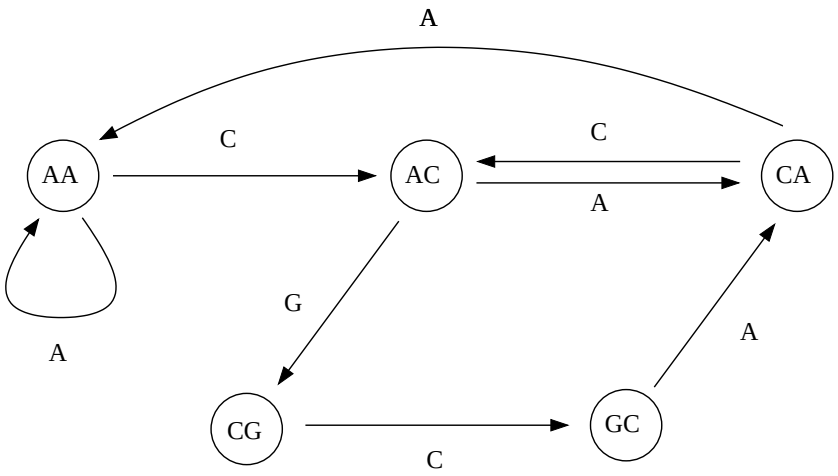


Figure 2.6: De Bruijn graph with parameter $k = 2$ for the list of strings $L = \{AAA, AAC, ACA, CAC, CAA, CGC, GCG\}$.

create one node for each 2-mer in the set L and a directed edge from node x_1 to node x_2 if the $k-1(=1)$ suffix of x_1 is a prefix of x_2 and we label the edge with the rightmost character in x_2. Hence, in this graph each edge corresponds to just one of the k-mers and so the general problem consists of finding a path that visits all the edges exactly once, an *Eulerian path*. The string S corresponding to a path in this graph can be reconstructed in the following way: begin the string S with the label of the first node and then concatenate, in order, all the labels of the edges in the path. For example, in figure 2.6 one Euler path is $AC \overset{A}{\rightarrow} CA \overset{C}{\rightarrow} AC \overset{G}{\rightarrow} CG \overset{C}{\rightarrow} GC \overset{A}{\rightarrow} CA \overset{A}{\rightarrow} AA \overset{A}{\rightarrow} AA \overset{C}{\rightarrow} AC$ and the reconstructed string associated to the path is $S = ACACGCAAAC$.

Although Eulerian paths in the De Bruijn framework can be computed in polynomial time, in reality there are many complications: (i) The De Bruijn graph might have more than one Eulerian path and, as for the earlier string graph framework, choosing the correct one is a non trivial task (Kingsford *et al.*

in [50] give a precise formula for the number of possible genomes that can be constructed from a De Bruijn graph). For example, another possible Eulerian path for the De Bruijn graph in figure 2.6 is $AC \xrightarrow{G} CG \xrightarrow{C} GC \xrightarrow{A} CA \xrightarrow{C} AC \xrightarrow{A} CA \xrightarrow{A} AA \xrightarrow{A} AA \xrightarrow{C} AC$ and the reconstructed string is $S = ACGCACAAAC$. (*ii*) Errors in the data can introduce many erroneous edges which complicate the graph structure; (*iii*) Even when an Eulerian path can be computed and it represents a possible assembly of the k-mers, it still might not constitute a correct assembly of the input reads (due to the presence of repeats). However, as mentioned before, since each read corresponds to a particular walk in the De Bruijn graph, and any walk that contains all the reads as subwalks represents a possible assembly of the reads.

Definition 11 (Superwalk)**.** *A walk is called a superwalk of BG^k if $\forall i$, $w(s_i)$ is a subwalk of it.*

In this framework a parsimonious solution corresponds to a superwalk of minimum length:

Definition 12 (Superwalk Problem (SAP_2))**.** *Given a set of fragments/reads $S = (r_1, r_2, \ldots, r_n)$ find a minimum length superwalk in the De Bruijn graph BG^k of S.*

The sequence assembly problem in the De Bruijn graph framework corresponds to Superwalk Problem, and, unsurprisingly this problem is also $\mathcal{NP}$-complete by reduction from the Shortest Superstring Problem [61]:

Theorem 4. *The Superwalk Problem is $\mathcal{NP}$-complete.*

2.5 Probability of unique reconstruction

The difficulty of assembling any set of reads (even if error-free) both in the String and De Bruijn graph frameworks is related to the overall structure of the genome and in particular to the presence/absence of repeats. Repeats can produce branches and cycles in the graph leading to multiple reconstructions. If we (incorrectly) assume that the genome has size n and its letters are uniformly random distributed (the nucleotide at each position is chosen independently and uniformly with probability $p = \frac{1}{4}$) , then we can ask the following question: What is the probability that a random genome can be uniquely reconstructed? Here we give a simple analysis based on the k-mer size in the De Bruijn framework. Since two k-mers overlap only if they have a prefix-suffix perfect match of size $k-1$, this analysis can be applied also to the special case of the String graph method with minimum overlap threshold k. Note that although this analysis is quite far from reality (genomes are not random), it helps to compute an estimate of the minimum k-mer size that should be used in any real application.

Note that a repeat of size $\geq k$ will always lead to a non-unique reconstruction, so we need to calculate the probability that a random DNA sequence of length n has no repeats of size $\geq k$. For very large n ($n >> 0$), we can define a random variable X whose values represent the number of times a k-mer occurs in the DNA sequence. The random variable X follows a binomial distribution with parameters $X \sim B(n, p^k)$. Therefore:

$$Pr(X = i) = \binom{n}{i} (p^k)^i (1 - p^k)^{n-i} \tag{2.10}$$

which is the probability of having exactly i repeats of k-mer in the random DNA sequence. Note that here we assume all the events to be independent. This assumption is not generally true since strings of size k can overlap in the DNA sequence, however, for very large genomes (e.g., human), the length n is big enough to assume these events to be independent with good approximation. If there are no repeats then for any k-mer only the events for $i = 0$ and $i = 1$ can happen, therefore we wish to approach the following equality as closely as possible.

$$Pr(X = 0) + Pr(X = 1) = (1 - p^k)^n + np^k(1 - p^k)^{n-1} = 1. \tag{2.11}$$

For large n (say, $n > 10^5$) and for a suitable ϵ, we may wish to find a k such that:

$$e^{-np^k} + \frac{np^k}{(1 - p^k)} e^{-np^k} > 1 - \epsilon. \tag{2.12}$$

Which follows from the fact that:

$$\lim_{n\to\infty} \left(1 - \frac{\lambda}{n}\right)^n = e^{-\lambda} \tag{2.13}$$

where $\lambda = np^k$. Equation 2.12 can be easily rewritten as follows:

$$1 + np^k > e^{np^k}(1 - \epsilon)(1 - p^k) + p^k > e^{np^k}(1 - \epsilon^*_{(k)}) \tag{2.14}$$

for some $\epsilon^*_{(k)} > \epsilon$. Note that $\epsilon^*_{(k)}$ depends on the choice of k. If we substitute

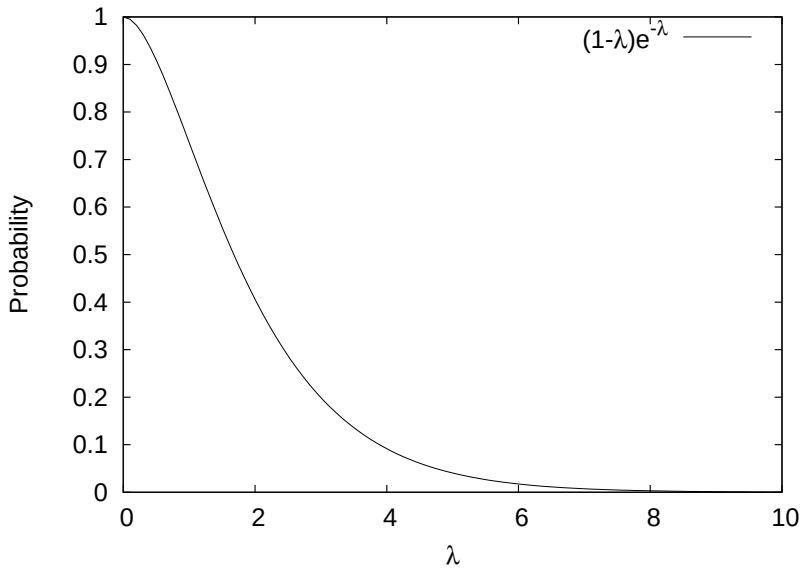


Figure 2.7: Plot of function $f(\lambda) = (1-\lambda)e^{-\lambda}$ in the range $[0, 10]$.

$np^k = \lambda_{\epsilon^*}$ we obtain:

$$(1 + \lambda_{\epsilon^*})e^{-\lambda_{\epsilon^*}} > (1 - \epsilon^*_{(k)}). \tag{2.15}$$

The function $f(\lambda) = (1-\lambda)e^{-\lambda}$ is plotted in figure 2.7. When $n \neq 0$, probability 1 can only be approached from below, which means it is never guaranteed that there are no repeats in the random DNA sequence. However from $\lambda_{\epsilon^*} = np^k$ it follows that:

$$k = log_{\frac{1}{p}}(n) + log_{\frac{1}{p}}\left(\frac{1}{\lambda_{\epsilon^*}}\right). \tag{2.16}$$

For a random sequence of human genome size we have that:

$$k = \underbrace{log_4(3 \times 10^9)}_{\approx 15.7} + 1.3 = 17, \tag{2.17}$$

where $p = \frac{1}{4}$. Then $\lambda_{\epsilon^*} = 4^{(-1.3)} = .165$ and the probability that a human-

sized random genome is 17-mer-repeat-free is bigger than $(1 - \epsilon = .988)$. This means that sequences of size $k > 17$ are already very unlikely to be repeated. Unfortunately the human genome is not random and contains many longer and complicated repeat structures than simple repeats of length 17 bps, so it would be required to use much higher values for k.

2.6 Sequence Assembly as a Constrained Optimization Problem

All the previously described formulations nicely convert the sequence assembly problem into well-defined combinatorial optimization problems; however, these formulations are inherently wrong since the best solution so obtained can be biologically incorrect (mis-assembled). Moreover, since all these problems are $\mathcal{NP}$-hard, many of the algorithmic solutions proposed over the last 20 years use greedy and heuristic methods which are inherently approximate. Thus, before discussing the various techniques used in sequence assemblers (chapter 3), it is important to give a new formulation of the sequence assembly problem whose solutions are more faithful in the context of biology. This formulation obviously will require the use of long-range information to concurrently validate and resolve the complex structures present in large genomes.

2.6.1 Modeling sequencing errors

So far, we have omitted from the discussion the possibility of errors in the reads. However, all currently available sequencing technologies are error-prone and these errors must therefore be modeled in our study of assembly. If there are errors in the reads then consistent layouts must also satisfy the following property:

Definition 13 (ϵ-valid layout)**.** *Let $0 \leq \epsilon < 1$ be the maximum error rate of the sequencing process. A layout L is* **ϵ-valid** *if each read $r_i \in L$ can be aligned to the reconstructed string R with no more than $\epsilon|r_i|$ differences.*

Note that in practice, the maximum error rate ϵ is used during the overlap computation to filter only detected overlaps between two reads r_1 and r_2 whose number of errors is no more than $\epsilon(|r_1| + |r_2|)$.

2.6.2 A new formulation of SAP

The sequence assembly problem (SAP) can be now formulated as follows:

Definition 14 (Sequence Assembly Problem (SAP_3))**.** *Given a collection of fragment reads $F = \{r_i\}_{i=1}^{N}$ and a tolerance level (error rate) ϵ, find a reconstruction R whose layout $L = \langle r_{j_1} \overset{\pi_1}{\rightleftharpoons} r_{j_2} \overset{\pi_2}{\rightleftharpoons} \cdots \overset{\pi_{N-1}}{\rightleftharpoons} r_{j_N} \rangle$ is* **ϵ-valid**, **consistent** *and such that the following set of properties (oracles) are satisfied :*

- *(Overlap-Constraint (O)): The cumulative overlap score O of the layout L is optimized:*

$$O(L) = \sum_{\substack{(r_i, r_j) \in L \\ \pi(r_i, r_j)}} S_O(r_i, r_j) \tag{2.18}$$

where S_O is the score of the overlap π between the two reads r_i and r_j.

- *(Mate-Pair-Constraint (MP)): The cumulative mate-pair score S_{MP} of the distance between reads in the layout L is consistent with the mate-pair constraints:*

$$MP(L) = \sum_{\substack{(r_i,r_j)\in L \\ (r_i \leftrightarrow r_j)}} S_{MP}(r_i, r_j) \tag{2.19}$$

where $r_i \leftrightarrow r_j$ indicates that the two reads r_i and r_j are oriented according to the mate library.

- *(Optical-Map-Constraint (OM)): The observed distribution of restriction enzyme sites in the layout L, $C_{obs} = \langle a_1, a_2, \ldots, a_n \rangle$, is consistent with the distribution of experimental optical map data $C_{src} = \langle b_1, b_2, \ldots, b_n \rangle$ (obtained by a restriction enzyme digestion process).*

This thesis focuses on constraints $O(L)$ and $MP(L)$ for which detailed formulations are given in chapter 4. The details for the Optical-Map-Constraint (OM) score will be addressed in the future work with suitable modification of the framework presented here. The general idea would be to use a scoring system based on the following scoring function:

$$\chi^2 = \sum_{k=1}^{n} \left(\frac{a_k - b_k}{\sigma_k} \right)^2 \tag{2.20}$$

where σ_k is the standard deviation of the observed distribution of restriction enzyme sites (modeled as normally distributed random variables). In practice, optical maps are not error-free and the presence of sequencing errors complicates

the matching process by introducing *false-cuts* and *missing-cuts* that must be properly accounted for. A possible solution to reduce the effect of these errors is to employ multiple restriction fragments in order to increase the map resolution. Another option is to design a Bayesian score function analogous to the validation approach presented in [6], but this method requires a dynamic programming implementation which could be too computationally expensive to use during the assembly process. A similar approach can be used in conjunction with long range data generated by the Bionanomatrix[4] platforms. An efficient and accurate solution must be designed to somehow combine the best of these two approaches.

Each property in definition 14 plays an important role in resolving problems that arise when real genomic data is used (e.g., data containing repeat-regions, rearrangements, segmental duplications, etc.). Note that in absence of additional information, among all possible layouts, the minimum length layout is typically preferred (*shortest superstring*), although, as previously explained, this choice is difficult to justify. As the genomic sequence deviates further and further from a random sequence, minimum length layouts typically introduce various mis-assembly errors (e.g., compression, insertions, rearrangements, etc.). Note that, traditionally, assemblers have only optimized/approximated one of the properties (i.e., (O)) listed above, while checking for the others in a post-processing step. If the formulation were only to contain the overlap-constraint O, then the problem would correspond to the sequence assembly problem as defined in 9 for the String-graph. The other two constraints do not reduce the complexity of the problem so this new formulation still belongs to the class of $\mathcal{NP}$-complete problems. Finally,

[4]http://www.bionanomatrix.com/

note that this list of constraints is not exhaustive, and it will likely change from year to year as new sequencing technologies become available and new types of long range information become possible to produce. It is thus important to have an assembly framework that can dynamically and effortlessly adapt to the new technologies and constraints.

2.6.3 Relation to the prior art

At this point it is important to relate the new SAP_3 formulation to the one originally presented in Myers' paper in 1995 [67]. In his seminal paper, Myers defines the fragment assembly problem as follows:

Definition 15 (Fragment Assembly Problem (FAP))**.** *Given a set of fragments $S = (r_1, r_2, \ldots, r_n)$ and a maximum error rate $\epsilon \in [0, 1]$, the Fragment Assembly Problem (FAP) is the problem of finding a reconstruction R and ϵ-valid layout of the reads whose observed distribution of fragment read start points, D_{obs}, has the minimum relative deviation from D_{src} (the source distribution of the sampling process).*

In particular, the goodness-of-fit between the observed distribution and the hypothesized source distribution is modeled using the density function of the Kolmogorov-Smironv (KS) test statistic [52, 93], which gives a likelihood function to optimize (the details of this function can be found in the original paper [67]).

Although this formulation also defines the problem as an optimization problem (under the KS statistic), it is not general enough to make use of additional long-range information (used as constraints in the SAP_3 formulation). But most

importantly, although the problem is modeled as an optimization problem, the proposed algorithmic solution in [67] does not tackle the whole assembly problem (with constraints) directly but it proposes a graph-theoretic strategy which is the core of the *overlap-layout-consensus* (OLC) paradigm widely used in many first-generation assemblers (see chapter 3 for a review of assembly paradigms). In particular, lemma 4 of Myers' paper proves that the shortest-common-superstring of the "chunk"[5] graph is the shortest-common-superstring (SSP) of the reads themselves. This means that the set of graph transformation used to create the chunks is lossless under that model of sequence assembly. However, as we have shown in section 2.3, the SSP can yield biologically implausible and incorrect solutions because it is oblivious to how biological sequences are organized by evolution.

It is also relevant to mention that in [67] Myers proposes to design "algorithms that are capable of solving a 'pure' shotgun problem subject to a collection of overlap, orientation, and distance constraints that model the additional information provided by the directed components of the strategy." However, he explains that such a *shotgun-with-constraints* problem is not addressed in his paper, but it should be explored "if there is to be any hope of solving these more difficult constraint problems" [67]. The formulation of sequence assembly problem presented in this section covers such theoretical gap and, together with the algorithmic solution presented in chapter 4, it represents a further step toward algorithms that solve the *pure* sequence assembly problem.

[5]Chunks correspond to unitigs (uniquely assembled contigs) in the OLC framework.

Chapter 3

Sequence Assemblers and Assembly Paradigms

3.1 Introduction

The history of sequence assembly can be seen as a sequence of responses to changes in sequencing technologies, and, as a result, several sequence assemblers and assembly paradigms have been designed over the last 30 year to accommodate these changes. This chapter first presents a historical perspective on sequence assembly, then it reviews the major assembly paradigms that have been successfully applied to large sequencing projects. All of the major sequence assemblers are categorized according to the assembly paradigm that they adopt.

3.2 A Historical Perspective on Sequence Assembly

Computational biologists first formalized the shotgun sequence assembly problem in terms of an approximation to finding the shortest common superstring (SCS) of a set of sequences [97]. Because of the theoretical computational intractability ($\mathcal{NP}$-completeness) of exact SCS-problem (SCSP), most of the approaches for genome sequence assembly have resorted to greedy and heuristic methods that by definition restrict themselves to near-optimal solutions, where the "nearness" may be guaranteed within a multiplicative competitiveness factor, e.g., four (see next section for a review of sequence assembly paradigms). In this context, the accuracy of the resulting reference genome sequences and their suitability for biomedical applications play a decisive role, as they additionally depend upon many parameters of the sequencing platforms: read lengths, base-calling errors, homo-polymer errors, etc. These parameters continue to change at a faster-and-faster pace as the platform chemistry and engineering continue to evolve.

To further emphasize this point, we note that there are now several efforts to develop a relatively cheap (e.g., $1000) genome sequencing technology of acceptable accuracy (e.g., one base error in 10,000 bps) and high-speed (e.g., a turn-around time of less than a day). However, it would be more useful to design assembly algorithms that are independent of the particular technology. Such a strategy would allow scientists to accommodate changes in technology more rapidly in the future years.

Name	Read Type	Algorithm	Reference
SUTTA	long & short	B&B	(Narzisi and Mishra [74], 2010)
Arachne	long	OLC	(Batzoglou et al. [11], 2002)
CABOG	long & short	OLC	(Miller et al. [64], 2008)
Celera	long	OLC	(Myers et al. [69], 2000)
Edena	short	OLC	(Hernandez et al. [32], 2008)
Minimus (AMOS)	long	OLC	(Sommer et al. [95], 2007)
Newbler	long	OLC	454/Roche
CAP3	long	Greedy	(Huang and Madan [34], 1999)
PCAP	long	Greedy	(Huang et al. [35], 2003)
Phrap	long	Greedy	(Green [30], 1996)
Phusion	long	Greedy	(Mullikin and Ning [66], 2003)
TIGR	long	Greedy	(Sutton et al. [96], 1995)
ABySS	short	SBH	(Simpson et al. [92], 2009)
ALLPATHS	short	SBH	(Butler et al. [18], 2008)
ALLPATHS-LG	short	SBH	(Gnerre et al. [29], 2010)
Contrail	short	SBH	(Schatz M. et al., 2010)
Euler	long	SBH	(Pevzner et al. [79], 2001)
Euler-SR	short	SBH	(Chaisson and Pevzner [19], 2008)
Ray	long & short	SBH	(Boisvert et al. [15], 2010)
SOAPdenovo	short	SBH	(Li et al. [60], 2010)
Velvet	long & short	SBH	(Zerbino and Birney [104], 2008)
PE-Assembler	short	Seed-and-Extend	(Nuwantha and Sung [75], 2010)
QSRA	short	Seed-and-Extend	(Bryant et al. [16], 2009)
SHARCGS	short	Seed-and-Extend	(Dohm et al. [22], 2007)
SHORTY	short	Seed-and-Extend	(Hossain et al. [33], 2009)
SSAKE	short	Seed-and-Extend	(Warren et al. [101], 2007)
Taipan	short	Seed-and-Extend	(Schmidt et al. [88], 2009)
VCAKE	short	Seed-and-Extend	(Jeck et al. [41], 2007)

Table 3.1: List of sequence assemblers. Reads are defined as "long" if produced by Sanger technology and "short" if produced by Illumina technology . Note that Velvet was designed for micro-reads (e.g. Illumina) but long reads can be given in input as additional data to resolve repeats in a greedy fashion.

To a first approximation, the history of genome assembly could be read as a long series of rapidly adapting responses to changes in sequencing paradigms. At the start of the HGP, the proposed assembly strategy was based on the BAC-by-BAC (recursive and hierarchical divide-and-conquer) approach. In this approach, the genome is first broken up into a collection of large overlapping in-vivo-clonable

fragments (between 160 and 200 Kb) – called Bacterial Artificial Chromosomes or BACs. Since the location of the BACs could be mapped through restriction-finger-print-based overlaps and the resulting tiling-paths (using specialized laboratory protocols), this strategy only required assembly of sequence reads from each individual BAC, which could be accomplished independently and in parallel using the standard shotgun assembly method, e.g., using the Phrap [30] assembler. In an effort to reduce the assembly cost and time, in the late-90's it was proposed instead to tackle directly the full genome assembly problem by shotgun sequencing. Specialized assemblers, e.g., CELERA [69] and ARACHNE [11], were then developed to deal with this type of data, often collected from long eukaryotic genomes.

However, as previously highlighted in chapter 1, when next generation short read technologies first began to appear, all of the conventional sequence assemblers originally developed for longer Sanger reads failed to scale well on the new data. Consequently, new assemblers needed to be designed and developed *ab initio* to handle the features of the new sequencers (short and high coverage reads). Analogously, since mate-pairs are currently the most common auxiliary information used to resolve ambiguities due to repeats, these next-generation (Gen-1) assemblers have incorporated hard-wired mate-pair-centric heuristics that are practically useless in dealing with other new and higher quality auxiliary data (e.g., dilution sequencing, optical maps, etc.).

Historically, all these assemblers – each representing many man-years of effort – appear to require complete and costly overhauls, with each introduction of a new

short-read or long-range technology. Arguably, there is a need for novel assembly platforms that are flexible enough to handle future sequencing technologies with minimal changes to their infrastructure.

3.3 Assembly Paradigms

Based on the underlying search strategies, most of the assemblers belong to just two major categories: *greedy* and *graph-based*. Table 3.1 presents a nearly exhaustive list of the major sequence assemblers organized by assembly paradigms. It also contains the sequencing platform(s) that each assembler is capable of handling. Note that for clarity of exposition, SUTTA is also included in this table but its description is postponed to the next chapter.

3.3.1 Greedy

Because of its assumed computational intractability ($\mathcal{NP}$-completeness of the related "Shortest Common Superstring Problem"), most of the successful approaches for genome sequence assembly have resorted to greedy methods. Greedy algorithms typically construct the solution incrementally using the following basic steps: (*i*) pick the highest scoring overlap; (*ii*) merge the two overlapping fragments and add the resulting new sequence to the pool of sequences; (*iii*) repeat until no more merges can be carried out. Algorithm 1 shows the pseudo code of a generic greedy assembly strategy, while figure 3.1 illustrates with an example the greedy marge process.

Algorithm 1: GREEDY - pseudo code

Input: Set of reads/fragments
Output: Set of contigs

```
1 repeat
2 |   Calculate pairwise alignments of all fragments;
3 |   Choose two fragments with the largest overlap;
4 |   Merge chosen fragments and add the resulting new sequence to the
  |   pool of sequences;
5 |   heuristically correct errors (e.g., using mate-pairs);
6 until only one fragment is left ;
7 return Set of contigs;
```

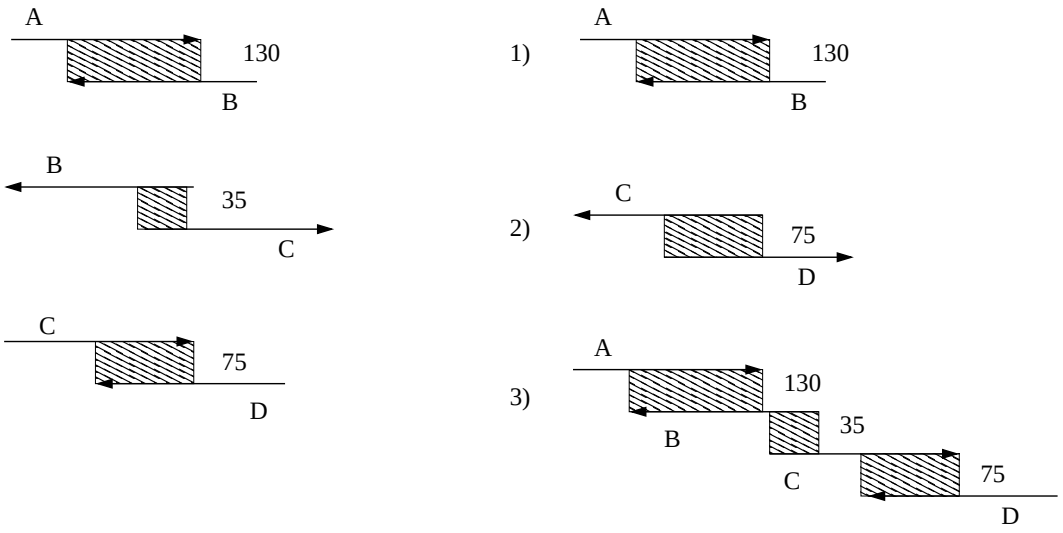


Figure 3.1: Example of greedily merging three fragments (the numbers represent the overlap sizes).

In addition, after each merge operation, the region of the overlay is heuristically corrected in some reasonable manner (whenever possible). Regions that fail to yield to these error-correction heuristics are relinquished as irrecoverable and shown as gaps. Mate-pairs information is used judiciously during the merging

process to further validate the connection between the two sequences. At the end of this process a single solution (consisting of the set of assembled contigs) is generated as output. Note that were the genome completely random, the overlap information could be computed unerringly, and would thus suffice for error-free reassembly of the target sequence. In this case, greedy algorithms would perform satisfactorily. As hinted earlier, the problem is complicated by the presence of non-random structure in genomes (e.g., repeated regions, rearrangements, segmental duplications — which are particularly pervasive in Eukaryotes), and it makes the greedy strategy invariably fail. Well known assemblers in this category include: TIGR [96], PHRAP [30], CAP3 [34], PCAP [35] and Phusion [66].

3.3.2 Graph-based

Graph-based algorithms start by preprocessing the sequence-reads to determine the pair-wise overlap information and represent these binary relationships as edges in a string-graph. The problem of finding a consistent lay-out can then be formulated in terms of searching a collection of paths in the graph satisfying certain specific properties. The paths correspond to *contigs* (contiguous sequences of the genome, consistently interpreting disjoint subsets of sequence reads). Contingent upon how the overlap relation is represented in these graphs, two dominant assembly paradigms have emerged: *Overlap-Layout-Consensus* (OLC) and *Sequencing-by-Hybridization* (SBH).

Overlap-Layout-Consensus. In the OLC approach, the underlying graph (overlap graph) comprises nodes representing reads and edges representing overlaps. Ideally, the goal of the algorithm is to determine a simple path traversing all the nodes — that is, a Hamiltonian path. For a general graph, this problem is known to lead to an $\mathcal{NP}$-hard optimization problem [61] though polynomial-time solutions exist for certain specialized graphs (e.g., interval graphs, etc.) [36]. In order to circumvent the theoretical intractability of this problem, a popular

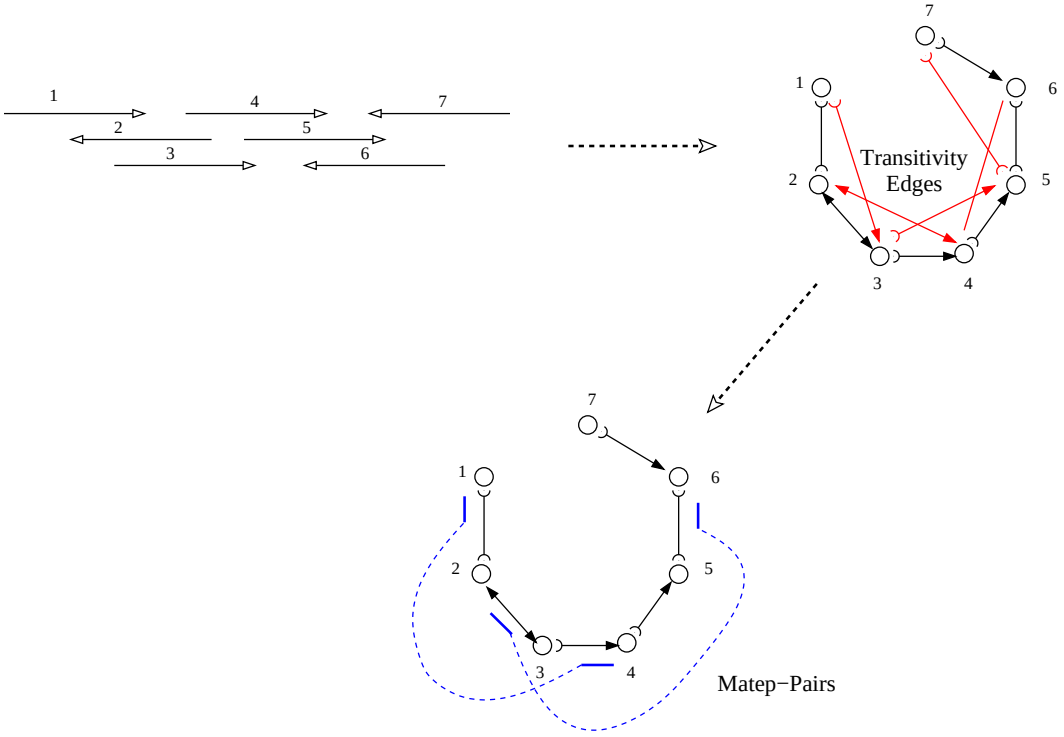


Figure 3.2: Example of layout representation and transformations in the OLC framework.

heuristic strategy is typically employed as follows:

1. remove "contained" and "transitivity" edges;

2. collapse "unique connector" overlaps (chordal subgraph with no conflicting edges) to compute the contigs;

3. use mate-pairs to connect and order the contigs.

Hence, the set of computed contigs corresponds to the set of nonintersecting simple paths in the reduced graph. Figure 3.2 shows an example of OLC graph and its sequence of transformations. Well known assemblers in this category include: CELERA [69], Arachne [11], Minimus [95] and Edena [32].

Sequencing-by-Hybridization. In the SBH approach, the underlying graph encodes overlaps by nodes and the reads containing a specific overlap by edges incident to the corresponding node (for that overlap). This dual representation may be described in terms of the following steps:

1. partition the reads into a collection of overlapping n-mers (an n-mer is a substring of length n);

2. build a DeBruijn graph in which each edge is an n-mer from this collection and the source and destination nodes are respectively the $(n-1)$-prefix and $(n-1)$-suffix of the corresponding n-mer.

In this new graph, instead of a Hamiltonian path, one seeks to find an Eulerian path containing every edge exactly once. Thus the computed genome sequence provides a consistent explanation for every consecutive n-mers on any sequence read. More importantly, such a graph is linear in the size of the input and allows (in theory) the computation of an Eulerian path to be carried out in linear time.

Because of the succinct representation it generates, this approach has become the algorithm of choice for assembling short reads with very high coverage. However, in practice, many complications arise. First, sequencing errors in the read data introduce many spurious (false-positive) edges which mislead the algorithm. Second, for any reasonable choice of n, the size of the graph is dramatically bigger than the one in the overlap-layout-consensus strategy. Third, a DeBruijn graph may not have a unique Eulerian path, and an assembler must find a particular Eulerian path subject to certain extraneous constraints; thus, it must solve a somewhat general problem, namely *the Eulerian-superpath problem*: given an Eulerian graph and a sequence of paths, find an Eulerian path in the Eulerian graph that contains all these paths as sub-paths. It is known that finding the shortest Eulerian superpath is, unfortunately, also an $\mathcal{NP}$-hard problem [61]. As earlier, the preferred work around is to use a heuristic method that computes such a superpath by applying a series of transformations to the original Eulerian graph. Prominent examples of the SBH approach include: Euler [79], Velvet [104], ABySS [92] and SOAPdenovo [60].

3.3.3 Seed-and-Extend

Recently, there has appeared yet another smaller category of assemblers specifically designed for short reads. They are based on a contig extension heuristic scheme, which uses a prefix-tree to efficiently look up potential extensions. In this framework a contig is elongated at either of its ends so long as there exist reads with a prefix of minimal length, provided that it perfectly matches an end of the

contig. Because of their heuristic scheme, these approaches can be categorized as greedy methods, but for the sake of clear exposition they are presented here using a separate category. Example of assemblers belonging to this new category are: PE-Assembler [75], SHARCGS [22], SSAKE [101], QSRA [16] and Taipan [88].

Finally, all the techniques described earlier need to separately incorporate mate-pairs information as they play an important role in resolving repeats as well as in generating longer contigs, thus dramatically reducing the cost of the assembly-finishing. Note that although mate-pairs are typically expensive and slow to obtain, prone to statistical errors and frequently incapable of spanning longer repeat regions, historically, they have played an important role in the assembly of large genomes as other cheaper and more informative mapping techniques have not been widely available. In addition, they have become essential to many emerging sequencing technologies (454, Illumina-Solexa, etc.), which can generate very high coverage short reads data (from 30 bp up to 500 bp).

Chapter 4

SUTTA: Scoring-and-Unfolding Trimmed Tree Assembler

4.1 Introduction

Mired by its connection to a well known $\mathcal{NP}$-complete combinatorial optimization problem — namely, the Shortest Common Superstring Problem (SCSP) — the whole-genome sequence assembly (WGSA) problem has been historically assumed to be amenable only to greedy and heuristic methods. By placing efficiency as their first priority, these methods opted to rely only on local searches, and are thus inherently approximate, ambiguous or error-prone, especially, for genomes with complex structures. Furthermore, since choice of the best heuristics depended critically on the properties of (e.g., errors in) the input data and the available long range information, (i.e., base-calling errors in sequencing technology) and the genome structure, these approaches hindered designing an error free WGSA

pipeline.

We dispense with the idea of limiting the solutions to just the approximated ones, and instead favor an approach that could potentially lead to an exhaustive (exponential-time) search of all possible layouts. Its computational complexity thus must be tamed through a constrained search (branch-and-bound) and quick identification and pruning of implausible overlays. For his purpose, such a method necessarily relies on a set of score-functions (*oracles*) that can combine different structural properties (e.g., transitivity, coverage, physical maps, etc.). In this chapter we give a detailed description of this novel assembly framework, referred to as SUTTA (Scoring-and-Unfolding Trimmed Tree Assembler) [74, 71, 72].

4.2 History and Motivation

There is no unanimous agreement, within the computer science community at least, that the sequence assembly problem has exhausted all reasonable methods of attack. For example, Richard M. Karp observed "The shortest superstring problem [is] an elegant but flawed abstraction: [since it defines assembly problem as finding] a shortest string containing a set of given strings as substrings. The SCSP problem is $\mathcal{NP}$-hard, and theoretical results focus on constant-factor approximation algorithms... *Should this approach be studied within theoretical computer science*?" [43]. In contrast to the work in computational biology, there have now emerged examples within computer science, where impressive progress has been made to solve important $\mathcal{NP}$-hard problems *exactly*, despite their worst-case exponential time complexity: e.g., *Traveling Salesman Problem*

(TSP), *Satisfiability (SAT)*, *Quadratic Assignment Problem (QAP)*, etc. For example, the work of Applegate et al. [7] demonstrated the feasibility of solving instances of TSP (as large as 85,900 cities) using *branch-and-cut*, whereas symbolic techniques in propositional satisfiability (e.g., DPLL SAT solver [21]), employing systematic backtracking search procedure (in combination with efficient conflict analysis, clause learning, non-chronological backtracking, "two-watched-literals" unit propagation, adaptive branching, and random restarts), have exhibited the capability to handle more than a million variables.

Inspired by these lessons from theoretical computer science, a novel approach, embodied in SUTTA algorithm, was developed. In the process, several related issues were addressed: developing better ways to dynamically evaluate and validate layouts, formulating the assembly problem more faithfully, devising superior and accurate algorithms, taming the complexity of the algorithms, and finally, developing a theoretical framework for further studies along with practical tools for future sequencing technologies.

4.3 SUTTA Algorithm

A common view of *de novo* genome assembly that has been quietly accepted among the genome sequence community is well expressed by the following quote [82]: *An assembler must either "guess" the correct genome from among a large number of alternatives (a number that grows exponentially with the number of repeats in the genome) or restrict itself to assembling only the non-repetitive segments of the genome, thereby producing a fragmented assembly.* Since exponential

growth of the time complexity would restrict the analyzable genomes to tiny sizes with very low coverage data, the preceding argument appears to give only a Hobson's choice: just find an approximated solution and trade off correctness against the price of exploring potentially exponentially many possible layouts of the reads. We take exception to this pessimistic view, and develop a new framework that works by forcefully eliminating incorrect solutions (i.e., implausible layouts).

Traditional graph-based assembly algorithms use either the overlap-layout-consensus (OLC) or the sequencing-by-hybridization (SBH) paradigm (as described previously), in which first the overlap/DeBruijn graph is built and the contigs are extracted later. SUTTA instead assembles each contig independently and dynamically one after another using the Branch-and-Bound (B&B) strategy. Originally developed for linear programming problems [54], B&B algorithms are well known searching techniques applied to intractable ($\mathcal{NP}$-hard) combinatorial optimization problems. The basic idea is to search the complete space of solutions. However the caveat is that explicit enumeration is practically impossible (i.e. has exponential time complexity). The tactics honed by B&B are to limit the search to a smaller subspace that contains the optimum. This subspace is determined dynamically through the use of certain *well chosen* score functions. B&B has been successfully employed to solve a large collection of complex problems, whose most prominent members are TSP (traveling-salesman problem), MAX-SAT (maximal satisfiability) and QAP (quadratic assignment problem).

Note that in the past B&B, was already suggested as a method to use in the context of the sequence assembly problem. However, those proposed strate-

gies are substantially different from the branch-and-bound framework adopted in SUTTA. For example, Gene Myers, in his seminal paper in 1995 [67], proposed a general framework to tackle the sequence assembly problem, which consisted of (1) assembling the fragments into "chunks" using a series of graph transformations, and (2) use a branch-and-bound procedure to search the space of all chunk paths. However, this B&B procedure was only proposed in the paper and the details were not resolved at the time of publication. Another example of B&B in the context of sequence assembly can be found in the paper by Kececioglu and Myers of 1995 [44]. Also in this case B&B method is not used to tackle the full problem but only proposed to solve some of its possible sub-problems, such as the *fragment orientation problem*.

At a high level, SUTTA's framework views the assembly problem simply as that of constrained optimization (based on definition 14): it relies on a rather simple and easily verifiable definition of feasible solutions as "consistent layouts." It generates potentially all possible consistent layouts, organizing them as paths in a "double-tree" structure rooted at a randomly selected "seed" read. A path is progressively evaluated in terms of an optimality criteria, encoded by a set of score functions based on the set of overlaps along the lay-out. This strategy enables the algorithm to concurrently assemble and check the validity of the layouts (with respect to various long-range information) through well-chosen constraint-related penalty functions. Complexity and scalability problems are addressed by pruning most of the implausible layouts, via a *branch-and-bound* scheme. Ambiguities resulting from repeats or haplotypic dissimilarities may occasionally delay

immediate pruning and force the algorithm to *lookahead*, but in practice, the computational cost of these events has been low. Because of the generality and flexibility of the scheme (it only depends on the underlying sequencing technologies through the choice of score and penalty functions), SUTTA is extensible, at least in principle, to deal with possible future technologies. It also allows concurrent assembly and validation of multiple layouts, thus providing a flexible framework that combines short and long range information from different technologies.

The high level SUTTA pseudocode is shown in Algorithm 2. Here, two important data structures are maintained: a forest of double-trees (D-tree) $\mathcal{B}$ and a set of contigs $\mathcal{C}$. At each step a new D-tree is initiated from one of the remaining reads in $\mathcal{F}$. Once the construction of the D-tree is completed, the associated contig is created and stored in the set of contigs $\mathcal{C}$. Next the layout for this contig is computed and all its reads are removed from the set of all available reads $\mathcal{F}$. This process continues as long as there are reads left in the set $\mathcal{F}$. Note that for the sake of a clear exposition, both the forest of D-trees $\mathcal{B}$ and the set of contigs $\mathcal{C}$ are kept and updated in the pseudocode; however, after the layout is computed, there is no particular reason to keep the full D-tree in memory, especially, where memory requirements are of concern.

Finally, note that the proposed Algorithm 1 is input order dependent. SUTTA adopts the policy to always select the next unassembled read with highest occurrence as the seed for the D-tree (also used by Taipan; [88]). This strategy minimizes the extension of reads containing sequencing errors. However, em-

Algorithm 2: SUTTA - pseudo code

Input: Set of N reads
Output: Set of contigs

```
1  B := ∅;                                          /* Forest of D-trees */
2  C := ∅;                                          /* Set of contigs */
3  F := ⋃_i^N {r_i};                  /* All the available reads/fragments */
4  while (F ≠ ∅) do
5  |  r := F.getNextRead();
6  |  if ( ¬isUsed(r) ∧ ¬isContained(r) ) then
7  |  |  DT := create_double_tree(r);
8  |  |  B := B ∪ {DT};
9  |  |  Contig CTG := create_contig(DT);
10 |  |  C := C ∪ {CTG};
11 |  |  CTG.layout();                        /* Compute contig layout */
   |  |  F := F \ {CTG.reads};                   /* Remove used reads */
12 |  end
13 end
14 return C;
```

pirical observations indicate that changing the order of the reads rarely affects structure of the solutions, as the relatively longer contigs are not affected. An explanation for this can be obtained through a probabilistic analysis of the data and a 0-1 law resulting from such an analysis.

4.4 Overlap Score (Weighted transitivity)

Like any B&B approach, a major component of SUTTA algorithm is the score function used to evaluate the quality of the candidate solutions that are dynamically constructed using the B&B strategy. SUTTA employs an overlap score based on the following argument. Large de novo sequencing projects typically

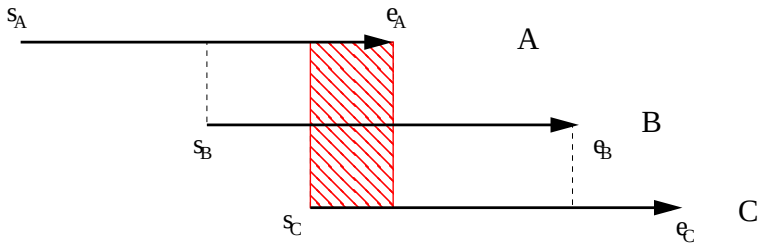


Figure 4.1: Example of transitivity relation: the overlap regions between reads AB and BC share an intersection.

have coverage higher than three, this implies that frequently two overlapping regions of three consecutive reads in a region of correct layout share intersections. Events of this type are "witness" to a transitivity relation between the three reads and they play an important role in identifying *true* positive[1] overlaps with high probability. Figure 4.1 shows an example of transitivity relation between three reads A, B and C. During contig layout construction, the overlap score uses the following basic principle to dynamically compute the score value of a candidate solution: if read A overlaps read B, and read B overlaps read C, SUTTA will score those overlaps strongly if in addition A and C also overlap:

$$\begin{aligned} &\texttt{if}(\pi(A,B) \wedge \pi(B,C))\texttt{then}\{S(\pi(A,B,C)) = \\ &\quad S(\pi(A,B)) + S(\pi(B,C)) + (\pi(A,C)?S(\pi(A,C)) : 0)\} \end{aligned} \tag{4.1}$$

The implicit assumption of a coverage of 3 can be further generalized to higher coverages in an obvious manner. Note that the score of a single overlap corresponds to the score computed by the Smith-Waterman alignment algorithm (for long reads) or exact matching (for short reads). Clearly the total score of a

[1]The two reads correctly originate from the same place in the genome.

candidate solution is given by the sum of the scores along the overlaps that join the set of reads in the layout L plus the score of the transitivity edges (if any):

$$\begin{aligned} g(L) = & \sum_{\substack{r_j \in L \\ j \in 2,\ldots,N-1}} S(\pi(r_{j-1}, r_j, r_{j+1})) \\ = & \sum_{\pi_i \in O} S(\pi_i(r_{j_1}, r_{j_2})) + \sum_{\pi_k \in T} S(\pi_k(r_{j_1}, r_{j_2})), \end{aligned} \tag{4.2}$$

where O and T are respectively the set of overlaps and transitivity edges (the set of reads is defined by the layout L) and $S(\pi)$ is the score (Smith-Waterman, exact match, etc.) for the overlap.

This step in SUTTA resembles superficially to the Unitig (unique regions of the genome) construction step in overlap-layout-consensus assemblers. Specifically, in these assemblers, one of the reduction steps applied to the overlap graph consists of removing all the transitivity edges, as it makes it simpler to find the unitigs directly from the simple paths in the graph (corresponding to sequences of reads connected by transitivity relations). However, unlike SUTTA, in the overlap-layout-consensus approach the weights of the overlaps are ignored in meaningfully scoring the paths. Since Unitig construction can be computationally expensive, large scale assemblers like Celera/CABOG [64] have adopted a strategy, where Unitigs are computed as chains of adjacent reads with best overlap between each other. This technique takes time and space linear in the number of reads. Although Celera's approach uses the overlap score during assembly, it is only applied locally for Unitigs — it neither scores the contigs globally nor generates multiple Unitigs solutions. Finally, it must be noted that the overlap scores are insufficient to resolve long repeats or haplotypic variations. The score

functions must be augmented with constraints (formulated as reward/penalty terms) arising from mate-pair distance information or optical map alignments.

4.5 Node expansion

The core component of SUTTA is the branch-and-bound procedure used to build and expand the D-tree (create_double_tree() procedure in Algorithm 2). The high-level description of this procedure is as follows:

1. Start with a random read (it will be the root of a tree; use only the read that has not been *used* in a contig yet, or that is not *contained*).

2. Create RIGHT Tree: start with an unexplored leaf node (a read) with the best score-value; choose all its non-contained *right*-overlapping reads ($Extensions()$ procedure in Algorithm 3); Filter out the set of overlapping reads by pruning unpromising directions ($Transitivity()$,$DeadEnds()$, $Bubbles()$ and $MatePairs()$ procedures in Algorithm 3); expand the remaining nodes by making them its children; compute their scores. (Add the "contained" nodes along the way, while including them in the computed scores; check that no read occurs repeatedly along any path of the tree). STOP when the tree cannot be expanded any further.

3. Create LEFT Tree: Symmetric to previous step.

Algorithm 3 presents the pseudocode of the expansion routine (details for each subroutine are available in the Appendix 7.6.1). In this framework each path

constructed using Algorithm 3 corresponds to a possible layout of the reads for the current contig. Unlike the graph-based approaches (OLC and SBH), multiple paths/layouts are concurrently expanded and validated. Based on the branching strategy, two versions of SUTTA are available: if at the end of the pruning process there are still multiple directions to follow the branching is either terminated (*conservative*) or not (*aggressive*). In the aggressive case, the algorithm choses the direction to follow with the highest local overlap. Algorithm 3 is applied twice to generate LEFT and RIGHT trees from the start read. Next, to create a globally optimal contig, the best LEFT path, the root and the best RIGHT path are concatenated together. Figure 4.2 illustrates the steps involved in the construction of a contig.

The amount of exploration and resource consumption is controlled by the two parameters K and T: K is the max number of candidate solution allowed in the queue at each time step, while T is the percentage of top ranking solutions compared to the current optimum score. At each iteration the queue is pruned such that its size is always $\leq max(K, T|Q|)$, where $|Q|$ is the current size of the queue. Note that while K remains fixed at each iteration of Algorithm 3, the percentage of top ranking solutions dynamically changes over time. As a consequence, more exploration is performed when many solutions evaluate to nearly identical scores.

A few additional implementation notes must be mentioned: (i) Checking right- or left-overlapping properties between two reads is only required while expanding the root; checking just the *consistency* relation for the non-root node

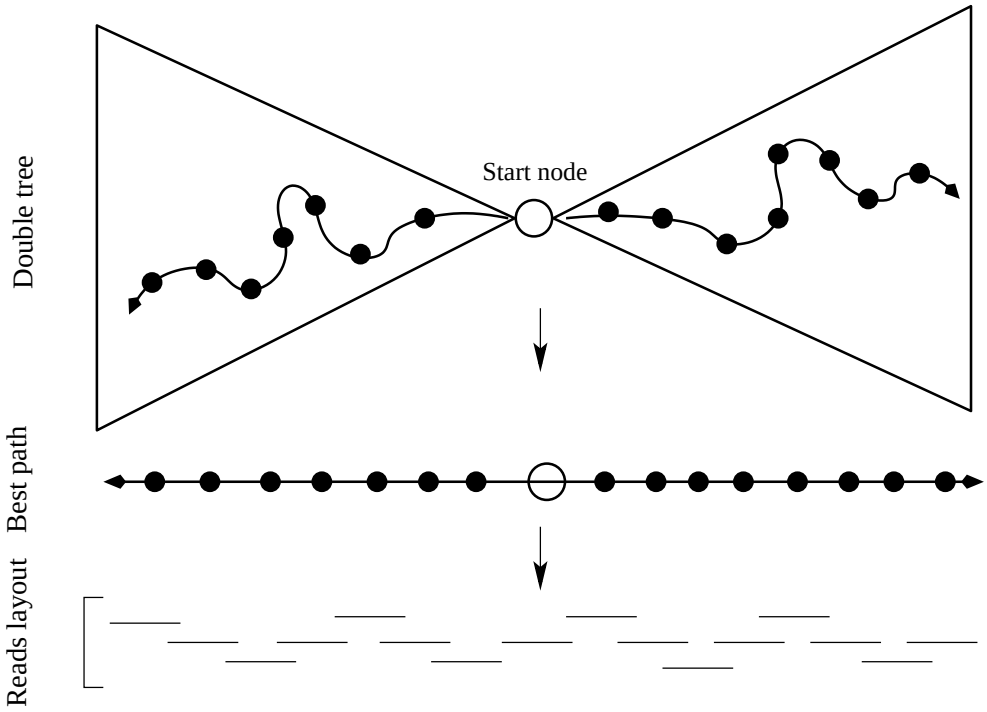


Figure 4.2: Contig construction: (*i*) the D-tree is constructed by generating LEFT and RIGHT trees for the root node; (*ii*) best left and right paths are selected and joined together; (*iii*) the reads layout is computed for the set of reads in the full path.

suffices. (*ii*) Caution must be taken in avoiding reads from the best right-path to be included in any left-path. (*iii*) Some book-keeping must be done to keep track of *used*, *explored*, *overlapping*, and *contained* relationships. Since different intermediate branches of the tree are kept during the expansion, the algorithm can potentially generate multiple solutions in its output and rank them in terms of their scores, etc. However, currently Algorithm 3 only returns in output a single solution with the maximal score (function g described in section 4.4). Ties are broken arbitrarily.

Algorithm 3: Node expansion

Input: Start read r_0, max queue size K, percentage T of top ranking solutions, dead-end depth W_{de}, bubble depth W_{bb}, mate-pair depth W_{mp}

Output: Best scoring leaf

```
1  V := ∅;                                                  /* Set of leaves */
2  L := {(r_0, g(r_0))};                          /* Live nodes (priority queue) */
3  while (L ≠ ∅) do
4  |  L := Prune(L, K, T);                                  /* Prune the queue */
5  |  r_i := L.popNext();                        /* Get the best scoring node */
6  |  E := Extensions(r_i);                              /* Possible extensions */
7  |  E^(1) := Transitivity(E, r_i);                    /* Transitivity pruning */
8  |  E^(2) := DeadEnds(E^(1), r_0, W_de);               /* Dead-end pruning */
9  |  E^(3) := Bubbles(E^(2), r_0, W_bb);                /* Bubble pruning */
10 |  E^(4) := MatePairs(E^(3), r_0, W_mp);               /* Mate pruning */
11 |  if (|E^(4)| == 0) then
12 |  |  V := V ∪ {r_i};                                  /* r_i is a leaf */
13 |  else
14 |  |  for (j=1 to |E^(4)|) do
15 |  |  |  L := L ∪ {(r_j, g(r_j))};
16 |  |  end
17 |  end
18 end
19 return max_{r_i ∈ V} {g(r_i)};
```

4.6 Search Strategy

A critical component of any branch-and-bound approach is the choice of the search strategy used to explore the next sub-problem in the tree. There are several variations among strategies (with no single one being universally accepted as ideal), since these strategies' computational performance varies with the problem type. The typical trade-off is between keeping the number of explored nodes in the search tree low and staying within the memory capacity. The two most common strategies are Best First Search (BeFS) and Depth First Search (DFS). BeFS always selects among the live (i.e., yet to be explored) subproblems, the one with the best score. It has the advantage of being theoretically superior since whenever a node is chosen for expansion, a best-score path to that node has been found. However, it suffers from memory usage problems, since it behaves essentially like a Breadth First Search (BFS). Also checking repeated nodes in a branch of the tree is computationally expensive (linear time). DFS instead always selects among the live subproblems the one with largest level (deepest) in the tree. It does not have the same theoretical guarantees of BeFS but the memory requirements are now bounded by the product of the maximum depth of the tree and the branching factor. The other advantage is that checking if a read occurs repeatedly along a path can be done in constant time by using the depth-first search interval schemes. For SUTTA we use a combined strategy: using DFS as overall search strategy, but switching to BeFS, when choice needs to be made between nodes at the same level. This strategy can be easily implemented by ordering the set of live nodes $\mathcal{L}$ of Algorithm 3 using the following *precedence*

relation between two nodes x and y:

$$x \prec y \text{ iff } \begin{cases} depth(x) > depth(y) \\ or \\ depth(x) == depth(y) \wedge score(x) > score(y) \end{cases}, \tag{4.3}$$

where *depth* is the depth of the node in the tree and *score* is the current score of the node (defined in section "Overlap Score"). Because BeFS is applied locally at each level the score is optimized concurrently.

4.7 Pruning the Tree

Transitivity pruning. The potentially exponential size of the D-tree is controlled by exploiting certain specific structures of the assembly problem that permit a quick pruning of many redundant and uninformative branches of the tree — surprisingly, substantial pruning can be done only using local structures of the overlap relations among the reads. The core observation is that it is not prudent to spend time on expanding nodes that can create a suffix-path of a previously created path, as no information is lost by delaying the expansion of the *last* node/read involved in such a "transitivity" relation. This scenario can happen every time there is a transitivity edge between 3 consecutive reads (see figure 4.1), and it is further illustrated in Figure 4.3 with an example. Suppose that $\langle A, B_1, B_2, \ldots, B_n \rangle$ are $n+1$ reads with a layout shown in figure 4.3. The local structure of the D-tree will have node A with n children $B_1, B_2, \ldots, B_n$. However , since B_1 also overlaps $B_2, B_3, \ldots, B_n$ these nodes will appear as chil-

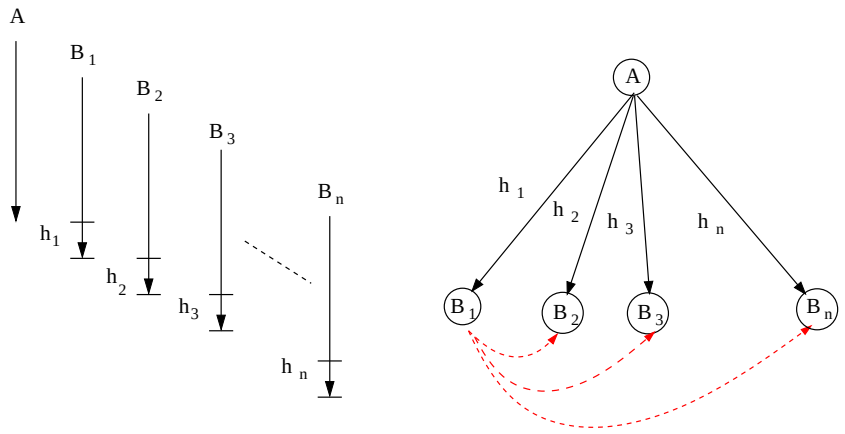


Figure 4.3: Example of transitivity pruning: expanding nodes $B_2, \ldots, B_n$ can be delayed because their overlap with read A is enforced by read B_1.

dren of B_1 at the next level in the tree. So the expansion of nodes $B_2, B_3, \ldots, B_n$ can be delayed because their overlap with read A is enforced by read B_1. Similarly arguments hold for nodes $B_2, B_3, \ldots, B_n$. In the best scenario, this kind of pruning can reduce a full tree structure into a linear chain of nodes. Additional optimization can be performed by evaluating the children according to the following order ($h_1 \leq h_2 \leq \cdots \leq h_n$), where h_i is the size of the hang[2] for read B_i. This ordering gives higher priority to reads with higher overlap score. This explains how the $Transitivity()$ procedure from Algorithm 3 is performed.

Zig-zag overlaps mapping. Although based on a simple principle, the time complexity of the transitivity pruning is a function of how quickly it is possible to check the existence of an overlap between two reads (corresponding to the red arrows of figure 4.3). The general problem is the following: given the set of overlaps O (computed in a preprocessing step) for a set of reads F, check the

[2]Size of the read portion that is not involved in the overlap.

existence of an overlap (or set of overlaps) for a pair of reads (r_1, r_2). The naive strategy that checks all the possible pairs takes time $O(n^2)$ where $n = |O|$. If a graph-theoretic approach is used, by building the overlap-graph information (adjacency list), this operation takes time $O(l)$ where l is the size of the longest adjacency list in the graph. However a much better (with $O(1)$ expected time) approach uses *hashing*. The idea is to build a hash-table, in which a pair of reads is uniquely encoded to a single location of the table by using the following hash-function:

$$H(a, b) = \frac{(a + b)(a + b - 1)}{2} + (1 - b), \tag{4.4}$$

where a and b are the unique identification numbers of the two reads. This is the well known *zig-zag* function which is the bijection often used in countability proofs. The number of possible overlaps $|H(a, b)|$ between two reads is always bounded by some constant c which is a function of the read length, genome structure (e.g., number of simple repeats) and the strategy adopted for the overlap computation (Smith-Waterman, exact match, etc.). In practice the constant c is never too large because even when multiple overlaps between two reads are available (typically 4), only a small subset with a reasonably good score (i.e. above a threshold) is examined by the algorithm.

4.8 Lookahead

Mate-pairs. Had the overlapping phase produced only true-positive overlaps, every overlapping pair of reads would have been correctly inferred to have orig-

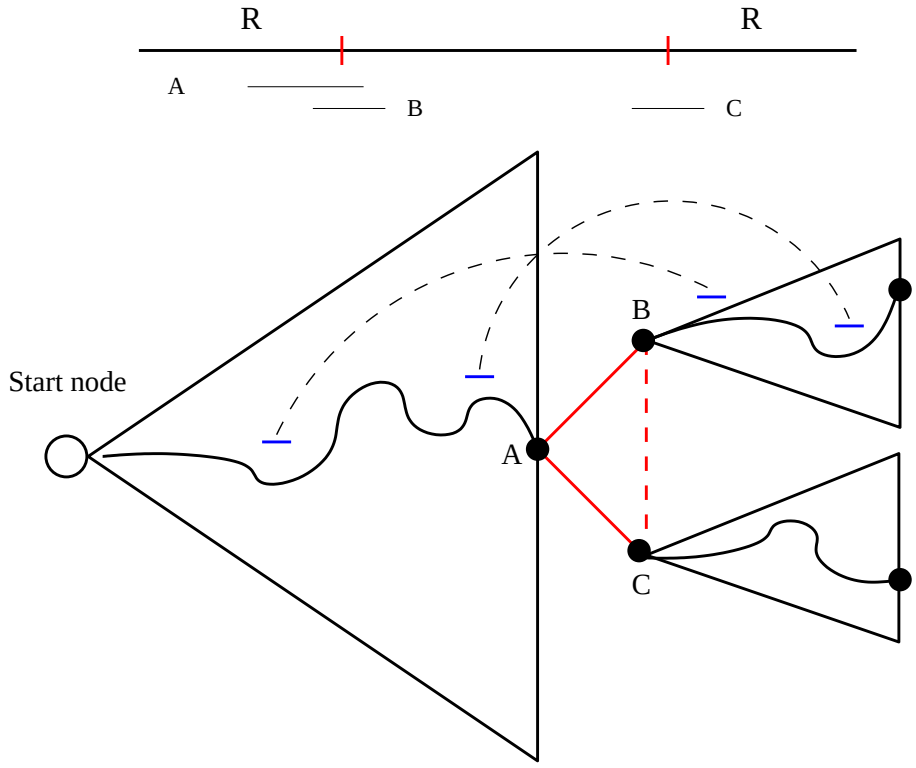


Figure 4.4: Lookahead: the repeat boundary between reads B and C is resolved looking ahead in the subtree of B and C, and checking how many and how well the mate-pair constrains are satisfied.

inated in the same genomic neighborhood, thus turning the assembly process to an almost trivial task. However, this is not the case — the overlap detection is not error-free and produces false-positive or ambiguous overlaps abundantly, especially when repeat regions are encountered. A potential repeat boundary between reads A, B and C is shown in figure 4.4. Read A overlaps both reads B and C, but B and C do not overlap each other. Thus, the missing overlap between B and C is the sign of a possible repeat-boundary location, making the pruning decisions impossible. However, SUTTA's framework makes it possible to resolve this scenario by *looking ahead* into the possible layouts generated by the

two reads, and keeping the node that generates the layout with the least number of unsatisfied constraints (i.e., consistent with mate-pair distances or restriction fragment lengths from optical maps).

SUTTA's implementation generates two subtrees: one for node B and the other for C (see figure 4.4). The size of each subtree is controlled by the parameter W_{mp}, the maximum height allowed for each node in the tree. The choice of W_{mp} is both a function of the size of the mate-pair library, local genome coverage and the genome structure. For genomes with short repeats a small value for W_{mp} is sufficient to resolve most of the repeat boundaries, and can be estimated from a k-mer analysis of the reads. However some genomes have much higher complexity (family of LINEs, SINEs and segmental duplications with varying homologies). In this case, a higher value of W_{mp} is necessary, but can be estimated adaptively. Once the two (or occasionally more) subtrees are constructed, the best path is selected based on the overlap score and the quality of each path is evaluated by a reward/penalty function corresponding to mate-pair constraints. For each node in the path, its pairing mate (if any) is searched to collect only those mate-pairs that cross the connection point between the subtree and the full tree, which are then scored by the following rule:

$$S_{MP}(r_1, r_2) = \begin{cases} 1, & \textit{iff} \;\; (l \in [\mu - \alpha\sigma, \mu + \alpha\sigma]) \wedge (r_1 \leftrightarrow r_2) \;\; ; \\ -1, & \textit{iff} \;\; (l \notin [\mu - \alpha\sigma, \mu + \alpha\sigma]) \wedge (r_1 \leftrightarrow r_2) \;\; ; \\ -1, & \textit{iff} \;\; \neg(r_1 \leftrightarrow r_2) \;\; ; \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

Here l is the distance between the two reads in the layout, μ and σ are the mean and standard deviation of the mate-pair library, α is a parameter that controls the relaxation of the mate-pair constrains (in the results, fixed at $\alpha = 6$), and $r_1 \leftrightarrow r_2$ denotes that the two reads are oriented towards each other. Such a score can be easily shown to give higher value to layouts with as few unsatisfied constraints as possible. Note that the mate-pair score is also dependent on local coverage of the reads, so its value should be adjusted/normalized to compensate for the variation in coverage. By penalizing the score negatively and positively according to the constraints, the current formulation assumes uniform coverage. However, more sophisticated score functions could be employed if it is necessary to precisely quantify the extent to which the score varies with coverage. The mate-pair score f of the full path P is given by the sum of the scores of each pair of reads with feasible constraints in P:

$$f(P) = \sum_{r_i, r_j \in P} S_{MP}(r_i, r_j) \tag{4.6}$$

Note that the current formulation of S_{MP} models only mate-pairs libraries whose reads face against each other. However, most current assemblies use a mixture of paired-end and mate-pair data sets that differ in insert size and read pair orientation. SUTTA's mate-pair score can be easily adapted to support any read pair orientation and insert size.

Memory management is very important during lookahead: the subtrees are dynamically constructed and their memory deallocated as soon as the repeat boundary is resolved. Also note that the lookahead procedure is performed ev-

ery time a repeat boundary is identified, so the extra work associated with the construction and scoring of the subtrees is performed only when repeated regions of the genome are assembled. Finally, note that the construction of each subtree follows the same strategy (from Algorithm 3) and uses the same overlap score (defined in section "Overlap Score"). However, recursive lookahead is not permitted. The mate-pair score introduced in (4.5) is used only to prune one of the two original nodes under consideration (or both, in the rare but possible scenarios, where neither of the subtrees satisfies the mate-pair constraints). This explains how the $MatePairs()$ procedure from Algorithm 3 is performed.

Dead-ends and Bubbles. Base pair errors in short reads from next generation sequencing produce an intuitively non-obvious set of erroneous paths in the graph and tree structures. Because perfect matching is used to compute the overlaps, these errors vary according to where the base error is located. Two possible ambiguities need to be resolved: *dead-ends* and *bubbles*. Dead-ends consist of short branches of overlaps that extend only for very few steps and they are typically associated with base errors located close to the read ends (see figure 4.5). Bubbles instead manifest themselves as false branches that reconnect to a single path after a small number of steps. They are typically caused by single nucleotide difference carried by a small subset of reads (see figure 4.6). Note that for human genomes bubbles might have been caused by either errors in the reads or haplotypic differences due to the structure of the human genome. In the second case both paths should be kept and given in output. The lookahead procedure is easily adapted to handle these kind of structures. Specifically for dead-ends, each

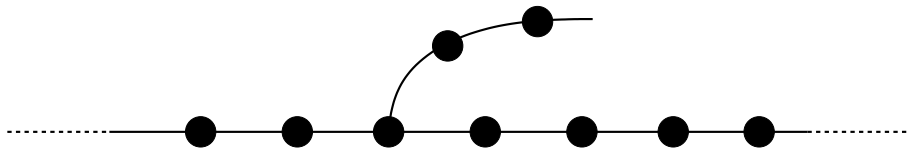


Figure 4.5: Dead-end: short branches of overlaps that extend only for very few steps. They typically associated with base errors located close to the read ends.

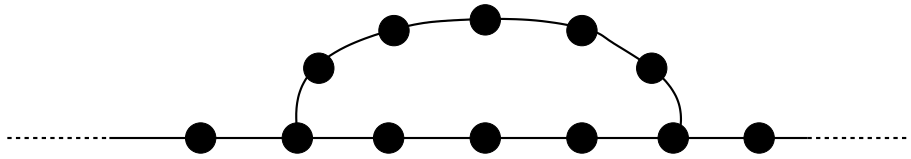


Figure 4.6: Bubble: false branches that reconnect to a single path after a small number of steps. They are typically caused by single nucleotide difference carried by a small subset of reads.

branch is explored up to depth W_{de} and all the branches that have shorter depth are pruned. In the case of bubbles, both branches are expanded up to depth W_{bb} and, if they converge, only the branch with higher coverage is kept and the other one is pruned.

4.9 Implementation details

SUTTA assembler is prototyped around the AMOS[3] assembly framework (A Modular Open-Source assembler). AMOS supports a central data repository of various genomic objects (reads, inserts, maps, overlaps, contigs, scaffolds, etc.) to be easily collected and indexed. In addition, the framework provides several algorithms to perform some of the standard steps in the assembly pipeline (e.g., Trimming, Overlapping, Error Correction, Scaffolding, Validation). SUTTA's

[3]http://amos.sourceforge.net

pipeline is composed of three modules: (1) overlapper, (2) contigger, and (3) multi-aligner. We developed our tools for the first two steps (described here). However, we relied on the "make-consensus" module available in AMOS for the computation of the final consensus sequence. Specifically, for the assembly of long reads, the UMD overlapper [84] has been used to compute the set of overlaps for the input reads. This overlapper keeps the number of repeat-induced spurious overlaps small and it builds the initial overlapping-phase of the algorithm with a reasonably small number of k-mers, whose cardinality is optimized by an order of magnitude through the use of minimizers. For the assembly of short reads, we instead developed our own short read overlapper (see next section).

4.10 Short-Read Overlapper

Current overlappers designed to deal with data generated by next-generation sequencing technologies (e.g., Illumina Inc. Genome Analyzer, Applied Biosystems SOLiD System and 454 Life Sciences) need to efficiently deal with an impressive amount of data (200X coverage or more in a single run for the bacteria E. coli). Allowing approximate matching (using dynamic programming) for the computation of the overlaps would not only be computationally intensive, but also significantly increases the number of nonspecific spurious overlaps due to sequencing errors. SUTTA's framework supports both Smith-Waterman alignments and perfect matching. However, for short read technologies, we have opted for an approach now popular among many short read assemblers: *exact string matching*. Because of the high coverage of next-generation sequencing data, this

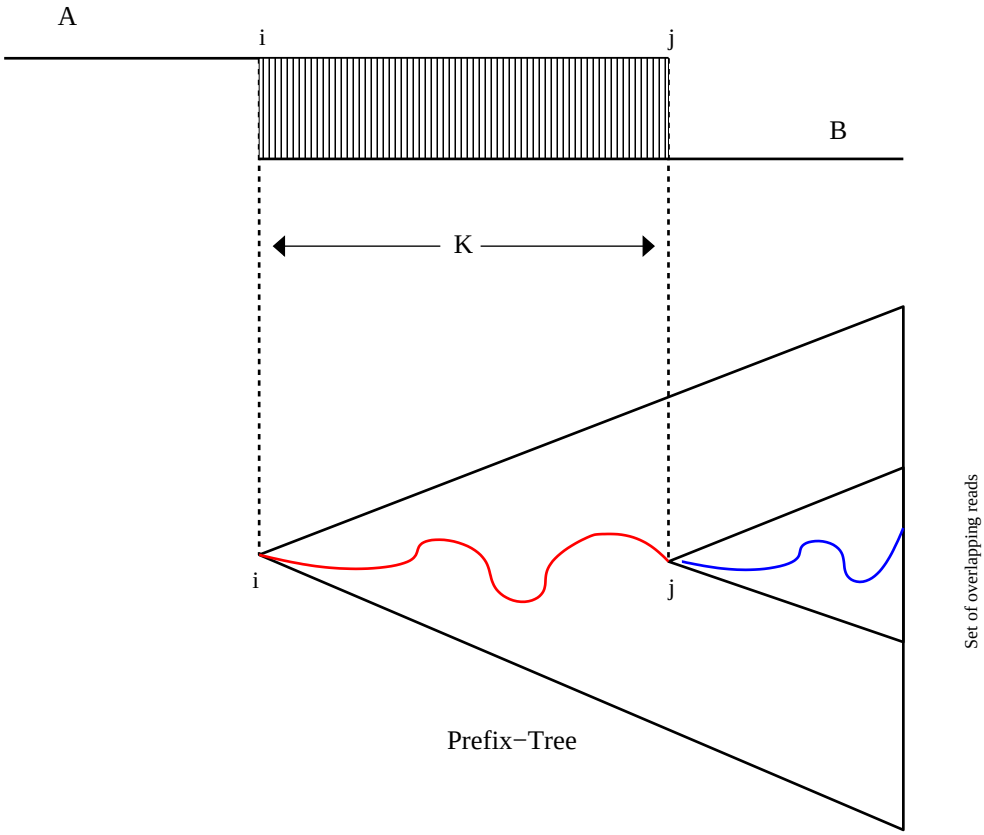


Figure 4.7: Overlap computation using the trie data structure.

technique produces reliable overlaps, while being drastically faster than approximate matching that requires dynamic programming. We first filter the data by removing redundant reads and those containing ambiguous bases. Next we index the remaining reads by a prefix-tree (trie) both in the forward and reverse complement direction. Using this data structure we can compute overlaps by a fast and simple traversal of the tree as illustrated in figure 4.7. Note that using this strategy for short reads enforces the overlaps to be only of type prefix or suffix.

Chapter 5

Feature-Response Curve

5.1 Introduction

As noted earlier, recent advances in DNA sequencing technology and their focal role in Genome Wide Association Studies (GWAS) have rekindled a growing interest in the whole-genome sequence assembly (WGSA) problem, thereby, inundating the field with a plethora of new formalizations, algorithms, heuristics and implementations. And yet, scant attention has been paid to comparative assessments of these assemblers quality and accuracy. No commonly accepted and standardized method for comparison exists as yet. Even worse, widely used metrics to compare the assembled sequences emphasize only size, while poorly capturing the contig quality and accuracy. This chapter addresses these concerns and introduces a novel metric that more satisfactorily captures the trade-offs between quality and contig size.

5.2 Assembly Comparison and Validation

Though validation and performance evaluation of an assembler are very important tasks, no commonly accepted and standardized method for this purpose exists as yet. The genome validation process appears to have remained a largely manual and expensive process, with most of the genomes simply accepted as draft assemblies. For instance, the initial "draft" sequence of the human genome [38] has been revised several times since its first publication, with each revision eliminating various classes of errors through successive algorithmic advances. Nevertheless, genome sequencing continues to be viewed as an inexact craft and inadequate in controlling the number of errors, which in the draft genomes are estimated to be up to hundred or even thousands [86]. The errors in such draft assemblies fall into several categories: collapsed repeats, rearrangements, inversions, etc., with their incidents varying from genome to genome.

It should be noted that the most popular metrics for evaluating an assembly (e.g., contig size and N50) only emphasize size and poorly capture the contig quality as they do not contain all the information needed to judge the correctness of the assembly. For example N50 is defined as the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs. In these scenarios, an assembler that sacrifices assembly quality in exchange for contig sizes, appears to outperform others, despite generating consensus sequences replete with rearrangement errors. For example, in the extreme case, an assembly consisting of one large contig of roughly the size of the genome is useless if mis-assembled. On the other extreme, an assembly

consisting of many short contigs covering only the inter-repeat regions of the genome could have very high accuracy although contigs might be too short to be used in, for example, gene-annotation efforts. Similarly, just a simple count of the number of mis-assembled contigs obtained by alignments to the reference genome (if available), a metric typically used to compare short-read assemblies, is also inadeguate, because it does not take into account the various structural properties of the contigs and of the reads contained in it. For example, one single mis-assembled contig could represent the longest contig in the set and it could include multiple types of errors (mate-pair orientation, depth of coverage, polymorphism, etc) which should be weighted differently. Although the evaluation of the tradeoff between contig length and errors is an important problem, there is very little in the literature to address this topic and help evaluate the assembly quality of different assemblers.

Consequently, we have developed a new metric, Feature-Response curve (FRC) [73], which captures the trade-offs between quality and contig size more accurately (see section 5.3). The FRC shares many similarities with classical ROC (receiver-operating characteristic) curves, which are commonly employed to compare the performance of statistical inference procedures. Analogous to ROC, FRC emphasizes how well an assembler exploits the relation between incorrectly-assembled contigs (false positives, contributing to "features") against gaps in assembly (false negatives, contributing to fraction of genome-coverage or "response"), when all other parameters (read-length, sequencing error, depth, etc.) are held constant.

5.3 Feature-Response Curve

Inspired by the standard receiver operating characteristic (ROC) curve, the Feature-Response curve characterizes the sensitivity (coverage) of the sequence assembler as a function of its discrimination threshold (number of features). The AMOS package provides an automated assembly validation pipeline called *amosvalidate* [80] that analyzes the output of an assembler using a variety of assembly quality metrics (or features). The features include:

- (M) mate-pair orientations and separations,
- (K) repeat content by k-mer analysis,
- (C) depth-of-coverage,
- (P) correlated polymorphism in the read alignments, and
- (B) read alignment breakpoints to identify structurally suspicious regions of the assembly.

After running amosvalidate on the output of the assembler, each contig is assigned a number of features that correspond to doubtful regions of the sequence. For example, in the case of mate-pairs checking (M), the tool flags regions where multiple matepairs are mis-oriented or the insert coverage is low. Given any such set of features, the response (quality) of the assembler output is then analyzed as a function of the maximum number of possible errors (features) allowed in the contigs. More specifically, for a fixed feature threshold ϕ, the contigs are sorted by size and, starting from the longest, only those contigs are tallied, if their sum

of features is $\leq \phi$. For this set of contigs, the corresponding approximate genome coverage is computed, leading to a single point of the Feature-Response curve. Algorithm 4 shows the pseudo-code of the FRC computation.

Algorithm 4: Feature-Response Curve - pseudocode

Input: Set of contigs $\mathcal{C}$ tagged with features/errors $\mathcal{F}$ computed using *amosvalidate*.

Output: Feature-Response Curve.

```
1  sort(C);        /* Sort contigs by length (longest to shortest) */
2  for (k=1 to 100) do
3      φ_k := |F| × k/100.0;                          /* Feature threshold */
4      sum := 0;                                        /* Sum of features */
5      tot_length := 0;                          /* Sum of contig lengths */
6      for (j=1 to |C|) do
7          f_j := c_j.getFeatures(); /* Num. of features for contig c_j */
8          sum := sum + f_j;              /* Update the sum of features */
9          tot_length := tot_length + c_j.getLength();
10         if (sum ≥ φ_k) then
               /* Exit if the sum of features is more than φ_k */
11             break;
12         end
13     end
14     cov_k := tot_length/genome_size × 100;        /* Approximate coverage for φ_k */
15 end
16 return (φ_k, cov_k), k ∈ {1, 2, ..., 100};
```

Note that no reference sequence is used to compute the FRC curve, which makes the FRC a useful tool in de novo sequencing project where a reference genome is not available to validate and guide the assembly process. In a scenario where the size of the genome is not available any reasonably good estimate of the reference genome size is adequate for the purpose of computing the FRC, since the genome size is simply used as a normalizing denominator across all

the assemblers to compare the contigs quality. For example, in the case of re-sequencing, a good estimate for the genome size can be obtained from genomes of the related species. In the case of the de novo sequencing projects the genome size can be judged from estimate of coverage (usually modeled as a dispersed Poisson) from a subsample of contigs (with some care to eliminate the outliers coming from repeats or difficult to sequence regions). The procedure described in Algorithm 4 can be applied to generate FRCs for each feature separately as shown in chapter 6 and appendix 7.6.1. The inspection of these separate FRCs enables to quantify comparisons of the relative strengths and weaknesses of each assembler. Finally note that the definition of coverage computed by the FRC is only an approximation of the standard one because the contigs are not aligned to the genome. However, it has the property of identifying assemblies where the genome length has been overestimated.

The current formulation of the FRC is exceedingly simple and yet natural. Thus, we hope that starting from here, more sophisticated versions of the FRC will be developed in the future. For example, the features could be weighted by contig length (density function); additional features may be included; features may be combined or transformed (e.g., eigen-features); the response, instead of coverage, could be another assembly quality metric of choice; etc. It must be emphasized that the features should not be interpreted directly as errors, since, as reported by the developers of *amosvalidate* [80], the method used to compute each feature may contain some false-positives. These false-positives frequently correspond to irresolvable inconsistencies in the assembly — and not mis-assembly

errors or incorrect consensus sequences. Consequently, the results could appear pessimistic for any one assembler, but are unlikely to be skewed in a comparative study, such as the one presented in chapter 6. The utility of Feature-Response curve is thus not diminished by the nature of the simple features, and it should be used in combination with other metrics and alignments to the reference genome (if available). Note further that since the reported sensitivity of *amosvalidate* is $\geq 92\%$, almost all the mis-assemblies are captured by one or more features, which point to possible sources of errors in a particular assembler.

5.4 Implementation details

As with SUTTA, the Feature-Response curve has been also developed as part of the AMOS[1] assembly framework (A Modular Open-Source assembler). Following the AMOS philosophy, the FRC is implemented as a pipeline that consists of two steps: 1) invocation to the *amosvalidate* tool to compute the features for the set of contigs; 2) invocation to the FRC module that implements Algorithm 4. FRCurve module is part of the AMOS distribution and its documentation is available at the AMOS wiki page[2].

[1] http://amos.sourceforge.net
[2] http://sourceforge.net/apps/mediawiki/amos/index.php?title=FRCurve

Chapter 6

Experimental Comparison of De Novo Genome Assembly

6.1 Introduction

Since the completion of the Herculean task of the Human Genome project (HGP) in 2003, the genomics community has witnessed a deluge of sequencing projects: They range from metagenomes, microbiomes, and genomes to transcriptomes; often, they focus on a multitude of organisms, populations and ecologies. In addition, the subsequent advent of high-throughput sequencing technologies – with their promise to considerably reduce the genome sequencing cost – now appear poised to usher in a personal genomics revolution [48, 90, 1].

However, in the ensuing euphoria, what seems to have been left neglected is a constructive and critical retrospection, namely:

1. to appraise the strengths and weaknesses of the schemes, protocols and

algorithms that now comprise a typical "sequencing pipeline";

2. to scrutinize the accuracy and usefulness of the assembled sequences by any standard pipeline;

3. and to build assembly algorithms that would easily adapt to the fast evolving biotechnologies.

This chapter addresses these issues by presenting a diverse set of experimental results to compare SUTTA's performance relative to many assembly algorithms in the literature with a targeted focus on both older Sanger technology and next-generation sequencing technology data. The analyses are performed under both standard metrics (N50, coverage, contig sizes, etc.) as well as the new more comprehensive metric (Feature-Response Curves, FRC) that has been introduced in chapter 5. Furthermore, visual inspection of the consistency of the assembled contigs is enabled by several graphic representations of the alignment against the reference genome, e.g., through dot-plots. Experimental analysis of the parametric complexity is also reported here showing that overlap graph complexity, assembly contiguity and assembly quality all strongly depend on the choice of the minimum overlap parameter k.

Specifically, the chapter is organized as follows: the experimental protocol adopted is first described; assembly results and quality analysis are then presented using standard paired and unpaired, low- and high-coverage, long and short reads from previously collected real and simulated data. Experimental analysis, showing the dependency of the assembly contiguity and quality on the choice of the minimum overlap size k, is presented next; and finally, the compu-

tational performance of SUTTA compared to several other assemblers concludes the chapter.

6.2 Experimental Protocol

In order to analyze the assembly quality of many different assemblers and to understand the inconsistencies that plague many traditional metrics, it was decided to collect a significant volume of comparative performance statistics using a large benchmark of both bacterial and human genome data. For all the genomes used in this study, the finished sequences are available, thus enabling direct validation of the assemblies. Before discussing the results, we present the benchmark and assemblers that we have selected for comparison and explain the design of the experimental protocol adopted here.

6.2.1 Benchmarks

In evaluating the assembly quality of different assemblers, several criteria were used in choosing bench-mark data sets, assembly-pipelines and comparison metrics: e.g., statistical significance, ease of reproducibility, accessibility in public domain etc. For example, by avoiding expensive studies with large sized genome-assembly and specialized (but not widely available) technologies, we wished to ensure that the reported results could be widely reproduced, revalidated and extended — even by moderate-sized biology laboratories or small teams of computer scientists. To the extent possible, we have favored the use of real data over synthetic data.

Consequently, we have not included large genomes (e.g., whole haplotypic human genomes) or single-molecule technologies (PacBiosciences or Optical Mapping), but ensured that all possible genome structures are modeled in the data (from available data or through simulation) as are the variations in coverage, read lengths and error rates. The only long range information included has come from mate-pairs. Note, however, that our analysis is completely general, as is the software used in this study, and can be used for broader studies in the future.

For these reasons, following datasets were selected (see table 6.1):

1. Sanger reads data, although now considered archaic, remain an important benchmark for the future. For instance, various technologies, promised by PacBiosciences, Life Technologies, and others, seek to match and exceed the read-lengths and accuracy of Sanger (hopefully, also inexpensively). Also Sanger-approach remains a statistically reliable source of data and implementations, since there continue to exist an active community of Sanger sequencers, a large amount of data and a variety of algorithmic frameworks dealing with Sanger data. As a result, they provide much more reliable statistics in the context of comparing so many different algorithmic frameworks (e.g., greedy, OLC and SBH). Such richness is not yet available from the current short-read assemblers, which have primarily focused on SBH (and de Bruijn graph representation).

2. Most recent Illumina machines can now generate reads of about 100 bps or more. However, our focus on 36 bps Illumina reads is based on the fact the these datasets have been extensively analyzed by previously published

short read assemblers. Since longer reads can only make the assembly process easier, these datasets still represent some of the hardest instances of the sequence assembly problem. We have also discovered that longer reads ($\sim$ 100 bps) from recent Illumina machines have higher error rates towards the ends of the reads, thus, limiting the apparent advantage of longer sequences.

3. By focusing on low-coverage long reads and short reads with high coverage, we stress-test all assemblers against the most extreme instances of the sequence assembly problem, especially, where assembly quality is of essence.

Genome	Length (bp)	# reads	Avg. read length (bp)	Std. (bp)	Cov.
Long reads:					
Brucella suus	3,315,173	36,276	895.8	44.1	9.8
Wolbachia sp.	1,267,782	26,817	981.9	50.6	20.7
Staphylococcus epidermidis	2,616,530	60,761	900.2	46.2	19.9
*Chromosome Y**	3,000,000	37,530	800	88	10
Short reads:					
Staphylococcus aureus	2,820,462	3,857,879	35	0	47.8
Helicobacter acinonychis	1,553,927	12,288,791	36	0	284.6
Escherichia coli	4,639,675	20,816,448	36	0	161.5

Table 6.1: Benchmark data. first and second columns report the genome name and length; columns 3 to 6 report the statistics of the shotgun projects: number of reads, average and standard deviation of the read length and genome coverage (*[35,000,001 - 38,000,000]).

Long reads. Starting with the pioneering DNA sequencing work of Frederick Sanger in 1975, every large-scale sequencing project has been organized around reads generated using the Sanger chemistry [87]. This technology could be typi-

cally characterized by reads of length up to 1000 bps and average coverage of 10×. Additional mate-pair constraints are typically available in the form of estimated distance between a pair of reads.

The first data set of Sanger reads consists of three bacterial genomes: *Brucella suis* [76], *Wolbachia sp.* [103] and *Staphylococcus epidermidis* RP62A [28]. These bacteria have been sequenced and fully finished at TIGR, and all the sequencing reads generated for these projects are publicly available at two sites: the NCBI Trace Archive[1], and the CBCB website[2]. Also included in the benchmark are sequence data from the human genome. Specifically, we selected a region of 3Mb from human *Chromosome Y*'s p11.2 region. These euchromatin regions of Y Chromosome are assumed to be particularly challenging for shotgun assembly as it is full of pathologically complex patterning of genome structures at multiple scales and resolutions — usually described as fractal-like motifs within motifs (repeats, duplications, indels, head-to-head copies, etc.). For this region of the Y Chromosome we generated simulated shotgun reads as described in Table 6.1. We created two mate-pair libraries of size ($\mu = 2,500, \sigma = 166$) and ($\mu = 10,000, \sigma = 1,300$) respectively; 90% of the reads have mates (45% from the first library and 45% from the second library), the rest of the reads are unmated; finally, we introduced errors in each read at a rate of 1%.

[1]http://www.ncbi.nlm.nih.gov/sra/
[2]www.cbcb.umd.edu/research/benchmark.shtml

Short reads. More recent advances in sequencing technology have produced a new class of massively parallel next-generation sequencing platforms such as: Illumina, Inc. Genome Analyzer, Applied Biosystems SOLiD System, and 454 Life Sciences (Roche) GS FLX. Although they have orders of magnitude higher throughput per single run (up to 200× coverage) than older Sanger technology, the reads produced by these machines are typically shorter (35–500 bps). As a result they have introduced a succession of new computational challenges, for instance, the need to assemble millions of reads even for bacterial genomes.

For the short reads technology, we used three different data sets, which have been extensively analyzed by previously published short read assemblers. The first data set consists of 3.86 million 35-bp unmated reads from the *Staphylococcus aureus* strain MW2 [10]. The set of reads for this genome are freely available from the Edena assembler website[3]. The second dataset consists of 12.3 million 36-bp unmated reads for a raw coverage of 284×. This second dataset is for the *Helicobacter acinonychis* strain Sheeba genome [23], which was presented in the SHARCGS [22] paper and is available for download at `sharcgs.molgen.mpg.de`. The third data set instead is made up of 20.8 million paired-end 36 bp Illumina reads from a 200 bp insert *Escherichia coli* strain K12 MG1655 [13] library (NCBI Short Read Archive, accession no. SRX000429).

[3] www.genomic.ch/edena.php

6.2.2 Assemblers

The following assemblers have been selected for comparison of long read pipelines: ARACHNE [11], CABOG [64], Euler [79], Minimus [95], PCAP [35], Phrap [30], SUTTA [74], and TIGR [96]. Simiarly, the following assembler were selected for comparison of short read pipelines: ABySS [92], Edena [32], Euler-SR [19], SOAPdenovo [60], SSAKE [101], SUTTA [74], Taipan [88], and Velvet [104]. Note that, although the most recent release of CABOG supports short reads from Illumina technology, it cannot be run on reads shorter than 64bp.

This list is meant to be representative (see [49] for a survey), as it includes assemblers satisfying the following two criteria: (*i*) they have been used in large sequencing projects with some success, (*ii*) together they represent all the generally accepted assembly paradigms (e.g., greedy, OLC, SBH and B&B; see the discussion in chapter 3) and (*iii*) the source code or binaries for these assemblers is publicly available on-line, thus enabling one to download and run each of them on the benchmark genomes. In order to interpret the variability in assembler performance under different scenarios, both paired and unpaired data were analyzed separately. All the long-read assemblers were run with their default parameters, while parameters for the short-read assemblers were optimized according to recent studies [74] (see table 7 in 7.6.1).

6.3 Long reads results

Analysis without mate-pair constraints. Table 6.2 presents the contig size analysis while excluding mate-pair data (thus, ignoring clone sizes and forward-reverse constraints). Since not all next-generation sequencing technologies are likely to produce mate-pair data, it is informative to calibrate to what extent an assembler's performance is determined by such auxiliary information. Note that ARACHNE had to be omitted from this comparison (and the associated table 6.2), since its use of mate-pairs is tightly integrated into its assembly process and cannot be decoupled from it. Similarly CABOG does not support data that is totally lacking in paired ends.

A caveat with the preceding analysis needs to be addressed: if one wishing to select an assembler of the highest quality were to base one's judgement solely on the standard and popular metrics (as in this table), the result would be somewhat uncanny and unsatisfying. For instance, Pharp would appear to be a particularly good choice, since it seems to typically produce an almost complete genome coverage with fewer contigs and each of sizable length (as confirmed by the N50 values). More specifically, except for *Wolbachia*, the N50 value of Phrap is the highest, yielding a respectable genome coverage of the big contigs ($>$ 10 kbp). Unfortunately, a closer scrutiny of the Phrap-generated assembly (e.g., the dot plots of the contigs' alignment) reveals that Phrap's apparent superiority is without much substance — Phrap's weaknesses, as evidenced by its mis-assemblies within long contigs (see alignments in the Appendix 7.6.1), are not captured by the N50-like performance parameters. Phrap's greedy strategy

Genome	Assembler	# ctgs	# big ctgs (>10 kbp)	Max ctg size (kbp)	Mean big ctg size (kbp)	N50 (kbp)	Big ctg cov. (%)
Brucella suis	Euler	280	118	82	22	19	78.4
	Minimus	203	101	89	30	32	93.1
	PCAP	88	62	198	53	80	100.7
	PHRAP	54	23	434	126	199	103.2
	SUTTA	73	53	268	62	79	99.2
	TIGR	108	67	182	48	57	98.8
Staphylococcus epidermidis	Euler	192	75	78	29	32	85.6
	Minimus	425	86	119	10	19	80.7
	PCAP	109	36	179	72	114	100.1
	PHRAP	86	22	357	123	183	103.9
	SUTTA	65	31	249	83	116	99.3
	TIGR	94	38	230	68	100	99.8
Wolbachia sp.	Euler	604	0	6	0	1	0
	Minimus	1545	37	16	13	2	40.7
	PCAP	1241	41	64	23	3	77.2
	PHRAP	2253	55	64	22	1.8	98.5
	SUTTA	1089	39	87	26	6	80.8
	TIGR	1080	46	46	20	5	73.6
Human Chromosome Y	Euler	60	27	403	107	266	96.7
	Minimus	850	104	48	18	11	63.1
	PCAP	140	38	239	77	112	98.2
	PHRAP	4	4	1869	764	1869	101.9
	SUTTA	15	10	1020	301	712	100.5
	TIGR	1103	108	51	10	8	63.7

Table 6.2: Long reads assembly comparison without mate-pair information (clone sizes and forward-reverse constraints). First and second columns report the genome and assembler names; columns 3 to 7 report the contig size statistics, specifically: number of contigs, number of contigs with size $\geq 10kbp$, max contig size, mean contig size, and N50 size (N50 is the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs). Finally column 8 reports the coverage achieved by the large contigs ($\geq 10kbp$). Coverage is computed by double-counting overlapping regions of the contigs, when aligned to the genome.

cannot always handle long-range genome structures and when a repeat boundary is found it can be fooled by false positive overlaps. In contrast TIGR, PCAP and SUTTA have similar performance in terms of N50; however, SUTTA produces a smaller number of big contigs (>10 kbp) compared to TIGR and PCAP, and higher genome coverage (except for the *S. epidermidis*, where they have similar coverage). All the assemblers encounter various difficulties in assembling the *Wolbachia sp.* dataset into long contigs, which is probably due to a higher error

rate in the reads. These difficulties are especially noticeable for Euler assembler; in fact, its big contigs coverage comes very close to zero. Minimus instead uses a very conservative approach where, if a repeat boundary is encountered, it stops extending the contig. Such a strategy reduces the possible mis-assembly errors, but causes a considerable pdecrease in contig size.

The results for the 3Mb region of *Chromosome Y* (from p11.2, a euchromatin region) paint a somewhat different picture. TIGR's and PCAP's performances are now inferior, with lower coverage and a higher number of contigs generated. In particular, PCAP performance was obtained by reducing the stringency in overlap detection to tolerate more overlaps (using parameter `-d 500`), this parameter setting was necessary in order to generate reasonably long contigs. Phrap still has the best performances in terms of contig size and N50, followed by SUTTA, but now its alignment results do not show mis-assembled contigs (see Appendix 7.6.1). Surprisingly, Euler now improves the genome coverage for simulated assembly. Note that for *Chromosome Y*, simulated reads were generated using fairly realistic error distributions, but still raise questions about the simulation's fidelity (e.g., ability to capture the non uniform coverage pattern, potential cloning bias, etc., that would be inevitable in any real large scale genomic project). Various simplifying assumptions used by the simulators may explain why simulated data appear somewhat easier to assemble.

Since the contig size analysis gives only an incomplete and often misleading view of the real performance of the assemblers, a more principled and informative approach needs to be devised. As described in chapter 5, a new metric, called

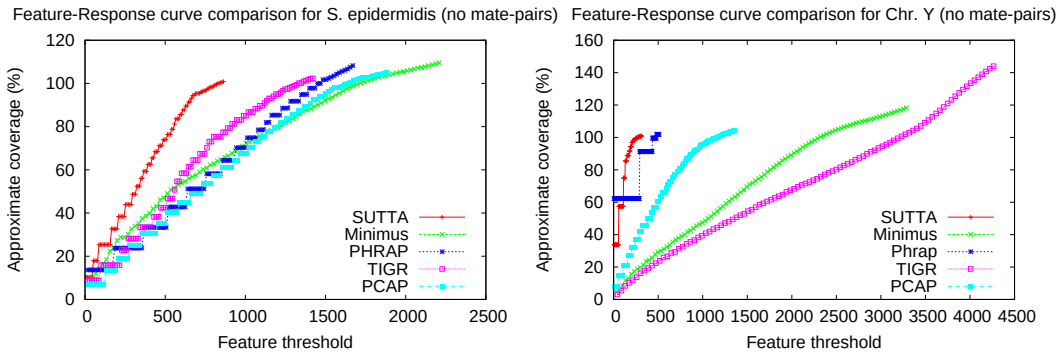


Figure 6.1: Feature-Response curve comparison for *S. epidermidis* and *Chromosome Y* (3Mbp of p11.2 region) genomes when no mate-pairs information is used in the assembly.

"Feature-Response curve" (FRC), is proposed and evaluated to see how well it can check the quality of the contigs and validate the assembly output. Figure 6.1 shows the Feature-Response curves for the *S. epidermidis* and *Chromosome Y* (p11.2 region) genomes when mate-pairs are not used in the assembly. The x-axis is the maximum number ϕ of errors/features allowed in the contigs and the y-axis reports the approximate genome coverage achieved by all the contigs (sorted in decreasing order by size) such that the sum of their features is $\leq \phi$ (see section 5.3 for more details). Note that the definition of coverage used in this plot is not the conventional one since we double-count overlapping regions of contigs, when aligned to the genome. We decided to employ such a definition because it highlights assemblies that over-estimate the genome size (coverage greater than 100%). Based on this analysis, SUTTA seems to be performing better than all the other assemblers in terms of assembly quality, however it is important to mention that the current version of the FRC includes several types

of assembly errors with a uniform weighting, chosen arbitrarily. For example, a mis-join is generally considered the most severe type of mis-assembly, but this is not currently captured by the FRC. In fact SUTTA clearly creates mis-joined contigs in the absence of paired reads (see dot plots in the Appendix 7.6.1). However, this problem is alleviated by the addition of paired reads as shown next in the analysis with mate-pair constraints. Finally, note that Euler and Minimus go to extreme short lengths to avoid mis-joins in the absence of paired reads.

Analysis with mate-pair constraints. Table 6.3 presents the results with mate-pairs data, restricting the analysis only to assemblers (ARACHNE, CABOG, Euler, PCAP, SUTTA and TIGR) that use mate-pair constraints effectively during the assembly process. Obviously, the use of mate-pair-constraints improves the performance and quality of all four assemblers; however, they do so to varying degrees. For example TIGR's N50 values are now typically twice as large as those without mate-pairs. In contrast, Euler's results only improve marginally with mate-pair constraints, and it is still unable to produce contigs larger than 10 kbp for the *Wolbachia sp.* genome. Note that Euler shows weaker performance in comparison to the results reported on its home-page[4] for the bacterial genomes. Although the exact explanation of this discrepancy is not obvious, it could be due to an additional screening (preprocessing) of the reads that removes low quality regions (note that, here, the analysis of all assemblers assumes no preprocessing.). ARACHNE and CABOG shows the highest N50 values for all datasets.

As earlier, whereas the contig size analysis indicates all of the following assem-

[4]http://nbcr.sdsc.edu/euler/benchmarking/bact.html

Genome	Assembler	# ctgs	# big ctgs (>10 kbp)	Max ctg size (kbp)	Mean big ctg size (kbp)	N50 (kbp)	Big ctg cov. (%)
Brucella suis	ARACHNE	33	28	463	119	161	101.2
	CABOG	30	19	775	175	268	101.5
	Euler	258	118	82	22	20	80.1
	PCAP	81	34	416	98	131	100.5
	SUTTA	72	58	269	56	74	99.2
	TIGR	69	43	361	77	112	99.9
Staphylococcus epidermidis	ARACHNE	27	17	565	156	294	101.7
	CABOG	41	8	655	330	483	103.1
	Euler	131	60	123	38	49	89.1
	PCAP	103	27	362	98	153	101.5
	SUTTA	89	63	244	48	63	97.6
	TIGR	51	12	545	220	389	101.1
Wolbachia sp.	ARACHNE	100	41	71	26	27	84.7
	CABOG	1035	26	181	47	26	178.7
	Euler	604	0	6	0	1	0
	PCAP	1263	54	42	18	3	79.1
	SUTTA	1132	46	104	22	6	86.7
	TIGR	1131	29	136	40	6	91.8
Human Chromosome Y	ARACHNE	5	4	1869	763	1869	101.7
	CABOG	5	4	1851	756	1851	100.9
	Euler	37	21	585	139	268	97.9
	PCAP	135	33	239	89	130	98.9
	SUTTA	17	8	1020	377	737	100.7
	TIGR	1030	116	48	18	10	72.6

Table 6.3: Long reads assembly comparison using mate-pair information. First and second columns report the genome and assembler names; columns 3 to 7 report the contig size statistics, specifically: number of contigs, number of contigs with size $\geq 10kbp$, max contig size, mean contig size, and N50 size (N50 is the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs). Finally column 8 reports the coverage achieved by the large contigs ($\geq 10kbp$). Coverage is computed by double-counting overlapping regions of the contigs, when aligned to the genome.

blers, ARACHNE, CABOG, PCAP and TIGR producing better performance, a cursory inspection of the Feature-Response curve points to a different conclusion. Figure 6.2 shows the FRCs of the assemblers for *S. epidermidis* and *Chromosome Y* (p11.2 region) genomes when mate-pairs are used in the assembly. Because Euler assembly output could not be converted into an AMOS bank for validation, it is excluded from the plot.

An intuitive understanding of the different assembly quality can be gleaned

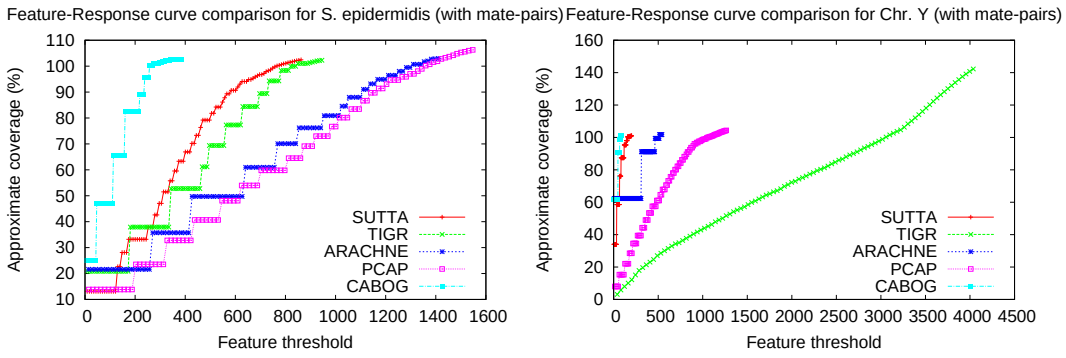


Figure 6.2: Feature-Response curve comparison for *S. epidermidis* and *Chromosome Y* (3Mbp of p11.2 region) genomes when mate-pairs information is used in the assembly.

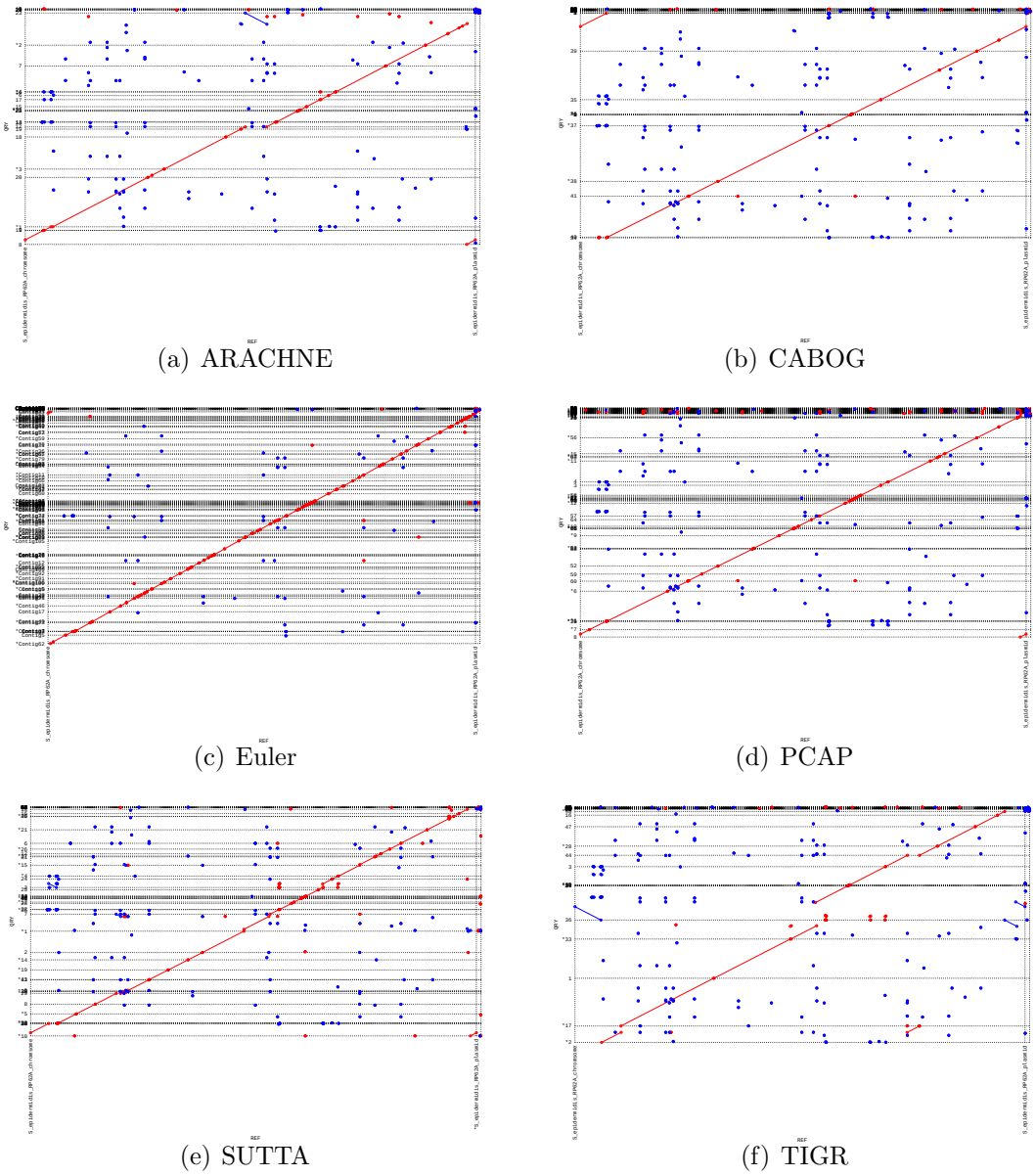
from Figure 6.3, which shows the dot plots of comparison of assemblies produced by the various assemblers aligned to the completed *S. epidermidis* genome. The dot plot alignments were generated using the MUMmer[5] package [53]. Assemblies generated by CABOG, Euler, PCAP and SUTTA are seen to match quite well with the reference sequence, as suggested by the fraction of matches lying along the main diagonal[6]. TIGR instead shows many large assembly errors, mostly due to chimeric joining of segments from two distinct non-adjacent regions of the genome. Further, note that two perfect dot plot alignments can still have different quality when analyzed with the FRC. An example is given by comparing the contigs generated by ARACHNE and SUTTA for the 3Mb segment of *Chromosome Y*'s p11.2. Despite the dot plots showing high alignment quality for both, the FRC scores them very differently (see Appendix 7.6.1). Appendix 7.6.1

[5]http://mummer.sourceforge.net/

[6]Note that since *S. epidermidis* has a circular genome, the small contigs aligned at the bottom right or top left are not mis-assembled.

contains the dot plots for the other genomes and the associated FRCs.

To further analyze the relative strengths and weaknesses of each assembler, Figure 6.4 shows separate FRCs for each feature type when assembling the *S. epidermidis* genome using mate-pairs. By inspecting these plots it is clear that each assembler behaves differently according to each feature type. For example, CABOG outperforms the other assemblers when mate-pair constraints are considered. TIGR and SUTTA outperform the other assemblers in the number of correlated polymorphism in the read alignments. The FRC that analyzes the depth of coverage shows ARACHNE, CABOG and PCAP to be winners in the comparison. Moving to the FRC that analyzes the k-mer frequencies, which can be used to detect the presence of mis-assemblies due to repeats, SUTTA and TIGR outperform ARACHNE and PCAP, while CABOG performs somewhere in between. The breakpoint-FRC examines the presence of multiple reads that share a common breakpoint, which often indicates assembly problems. PCAP and ARACHNE seem to suffer more from this problems, while the other assemblers are not affected (the FRCs reduces to a single point). Finally the mis-assembly FRC is computed using the mis-assembly feature which is obtained applying a feature combiner to collect a diverse set of evidence for a mis-assembly and output regions with multiple mis-assembly features present at the same region (see [80] for more details). CABOG in this case again achieves a superior rank over the other assemblers.

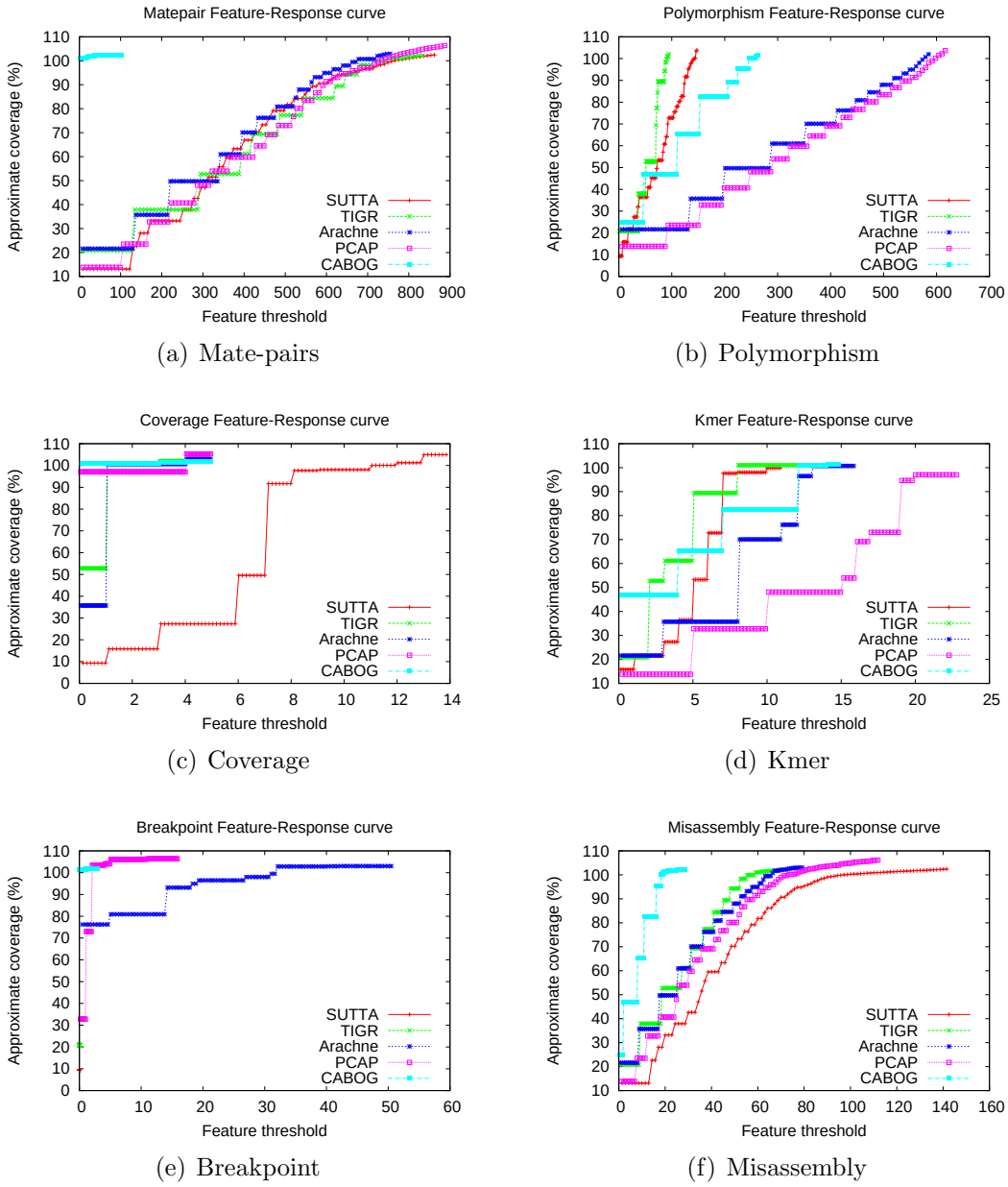


(a) ARACHNE (b) CABOG

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 6.3: Dot plots for the *Staphylococcus epidermidis*. Assemblies produced by ARACHNE, CABOG, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis. Note that number of single dots are an artifact of the sensitivity of the MUMmer alignment tool; they can be reduced or removed by using a larger value for the minimum cluster length parameter *–mincluster* (default 65).



(a) Mate-pairs (b) Polymorphism

(c) Coverage (d) Kmer

(e) Breakpoint (f) Misassembly

Figure 6.4: Separate FR-curve comparison for each feature type for the *S. epidermidis* genome using mate-pairs.

6.4 Short reads results

In case of short reads, interpretations of the contigs data, e.g., ones based purely on contig sizes and N50, etc. are complicated by the following facts: (1) for short-reads, the required threshold ratio $\frac{K}{L}$ is only slightly less than 1, where K is the required minimum overlap length and L is the length of the reads; therefore the effective coverage is significantly small, thus making all the statistics rather non-robust and highly sensitive to choice of the parameters; (2) there is no consensus definition of correctness of a contig – the required similarity varying from 98% down to 90% and the allowed end-trimming of each contig being idiosyncratic; and (3) many algorithms have specific error-correction routines that are embedded in a pre-processing or post-processing steps and that cull or correct bad reads and contigs in a highly technology-specific manner.

Analysis without mate-pair constraints. Returning to an analysis based on contig size, in Table 6.4, we show a comparison of the assembly results for the *S. aureus* and *H. acinonychis* genomes. The values reported for all the assemblers, are based on the tables presented in the recent SUTTA paper [74]. Only contigs of size ≥ 100 are used in the statistics. Without mate-pair information, as in here, it is inevitable that all assembly approaches (especially, if they are not conservative enough) could produce some mis-assemblies. As described earlier, what constitutes a correct contig is defined idiosyncratically (align along the whole length with at least 98% base similarity [32]), making it very sensitive to small errors which typically occur in short distal regions of the contigs. For

Genome	Assembler	# correct	# mis-assembled	N50 (kbp)	Mean (kbp)	Max (kbp)	Coverage (%)
S. aureus	ABySS	928	6	7.8	2.9	32.7	98
(strain MW2)	Edena (strict)	1124	0	5.9	2.4	25.7	98
	Edena (nonstrict)	740	16	9.0	3.7	51.8	97
	EULER-SR	669	33	10.1	4.0	37.9	99
	SOAPdenovo	867	25	8.1	3.1	30.8	97
	SSAKE	2073	378	2.0	1.1	9.7	99
	SUTTA	998	11	6.0	2.6	22.8	97
	Taipan	692	16	11.1	3.9	44.6	98
	Velvet	945	5	7.4	2.8	32.7	97
H. acininychis	ABySS	270	8	13.9	5.4	54.7	98
(strain Sheeba)	Edena (strict)	336	0	10.1	4.5	36.9	98
	Edena (nonstrict)	302	1	13.2	4.9	35.0	97
	EULER-SR	730	21	4.3	2.1	18.8	98
	SOAPdenovo	479	21	7.3	3.3	29.8	98
	SSAKE	675	156	3.2	1.8	14.6	99
	SUTTA	313	9	9.6	4.5	41.3	98
	Taipan	271	0	13.3	5.6	48.6	98
	Velvet	278	2	12.8	5.4	49.5	98

Table 6.4: Short reads assembly comparison without mate-pair information. First and second columns report the genome and assembler names; columns 3 to 7 report the contig size statistics, specifically: number of contigs, number of contigs with size $\geq 10kbp$, max contig size, mean contig size, and N50 size (N50 is the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs). Finally column 8 reports the coverage achieved by all the contigs.

example, contigs ending in gaps accumulate errors, as coverage gets lower towards the ends. To overcome such errors some assemblers perform a few correction steps. For example, Edena exercises an option to trim a few bases from these ends until a minimum coverage is reached; Euler-SR performs a preprocessing error correction step where errors in reads are corrected based on k-mer coverage analysis. From the table 6.4 it is clear that SSAKE has the worst performance in terms of contig size and quality, while the rest of the assemblers have relatively small errors and they all achieve high genome coverage ($\geq 97\%$).

Analysis with mate-pair constraints. Table 6.5 shows the assembly comparison using mate-pair information on the read set for the *E. coli* genome. The

comparison is based on the results from table 2 in [74]. In accordance with this analysis, statistics are computed only for contigs whose length is greater than 100 bps. A contig is defined to be correct if it aligns to the reference genome with fewer than five consecutive base mismatches at the termini and has at least 95% base similarity. By inspecting the column with the number of errors, one might conclude that the lower the number of errors the better the overall assembly quality. As explained earlier, a simple count of the number of total mis-assembled contigs is not informative enough. For example, ABySS and SOAPdenovo have the highest N50 values and a low number of mis-assembled contigs. However, such misassembled contigs are on average longer than those from other assemblers like SUTTA and Edena. This is evident in the table from the analysis of the mean length of the mis-assembled contigs. This analysis also shows that Edena and SUTTA behave more conservatively than Velvet, ABySS and SOAPdenovo, as they trade contig length in favor of shorter correctly assembled contigs. Interestingly Taipan's number of errors for *E .coli* increases compared to the results in table 6.4. Instead SOAPdenovo's performance improves for *E.coli* thanks to the availability of mate-pair information.

Note that the N50 statistic does not give any information about the reason why the contigs are mis-assembled: the contigs could contain an error due to accumulated errors close to the contigs' ends or it could contain rearrangements due to repeated sequences. Of course, these two error types have very different importance in terms of quality. In this scenario, the FRC analysis can give a deeper understanding of the assembly quality, as shown in Figure 6.5 where the

Genome	Assembler	# correct	# mis-assembled (mean kbp)	N50 (kbp)	Mean (kbp)	Max (kbp)	Coverage (%)
E. coli	ABySS	114	10 (49.5)	87.4	37.3	210.7	99
(K12 MG1655)	Edena	674	6 (13.2)	16.4	6.6	67.1	99
	EULER-SR	190	26 (37.8)	57.4	21.1	174.0	99
	SOAPdenovo	200	9 (71.8)	76.6	21.7	173.9	98
	SSAKE	407	66 (15.3)	31.2	9.6	105.9	98
	SUTTA	423	7 (18.8)	22.7	10.2	84.5	98
	Taipan	742	62 (5.2)	12.2	5.6	56.5	97
	Velvet	275	9 (52.9)	54.3	15.9	166.0	98

Table 6.5: Short reads assembly comparison using mate-pair information. First and second columns report the genome and assembler names; columns 3 to 7 report the contig size statistics, specifically: number of contigs, number of contigs with size $\geq 10kbp$, max contig size, mean contig size, and N50 size (N50 is the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs). Finally column 8 reports the coverage achieved by all the contigs.

contigs produced by SUTTA and Velvet are compared. Although SUTTA has a higher number of mis-assembled contigs (see table 6.5), the FRC presents a different scenario. By inspecting the feature information of the contigs produced by SUTTA and Velvet, it is seen that SUTTA's contigs have a lower number of unsatisfied mate-pair constraints, which leads to fewer large mis-assembly errors. This result is primarily due to SUTTA's optimization scheme, which allows it to concurrently optimize both overlap and mate-pair scores while searching for the best layout. Unfortunately we were unable to generate FRCs for each short-read assembler because their output could not be converted into an AMOS bank for validation. The ones that could be analyzed with FRC include SUTTA and Velvet.

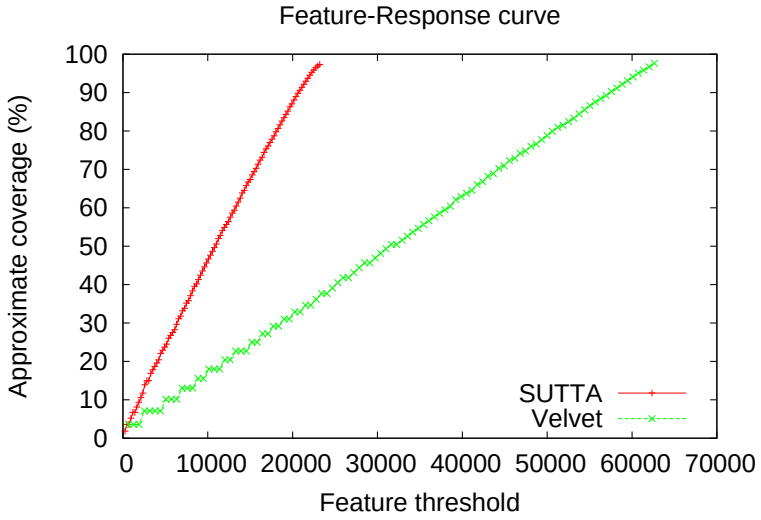


Figure 6.5: Feature-Response curve comparison for the *E. coli* genome using mate-pair short read data.

6.5 Parametric complexity experiments

In order to design better assembly algorithms and exploit the characteristics of sequence data from new technologies, it is important to have a deep understanding of the parametric complexity of the assembly problem. This is especially true for short reads from next-generation technologies since typically the required overlapping length represents a significant part of the read length. In fact, the min overlap length k is a determinant parameter, and its optimal setting strongly depends on the data (coverage). Therefore the effective coverage [55]:

$$E_{cov} = \frac{N(l-k)}{G} \tag{6.1}$$

is very sensitive to the choice of the minimum overlap parameter k, where N is the number of reads, l is the read length and G is the genome size. For example, in the case of the real dataset for *S. aureus* from table 6.1 ($l = 35$, $G = 2.82$ Mbp, $N = 3.86$ Millions) the raw coverage and effective coverage are respectively:

$$c = \frac{lN}{G} = 48X, \quad c_E = \frac{N(l-k)}{G} = 14X(k = 21) \tag{6.2}$$

This section illustrates the the strong dependency of assembly contiguity and quality on the choice of the minimum overlap parameter k.

6.5.1 Overlap graph complexity

As described in chapter 2, sequence assembly can be formulated as solving specific problems for a general graph constructed from the overlap information of the reads. The size and complexity of this graph is clearly a function of the minimum overlap size k. Figure 6.6 shows the overlap and read count distributions as a function of the minimum overlap parameter k for the *E. coli* dataset from table 6.1. The Y axis of the main plot shows how many reads have the number of overlaps in the X axis. The overlap graph seems to closely follow a *power law* distribution in accordance with random graph models and scale-free networks, where it is common to have vertices with a degree that greatly exceeds the average. In the context of genome sequences this is expected because of the presence of repeat regions: reads that have been sampled from such region are more likely to have a higher number of overlaps. These reads appear in the tail of the distribution in figure 6.6 ($x \geq 400$). The majority of the sequences have

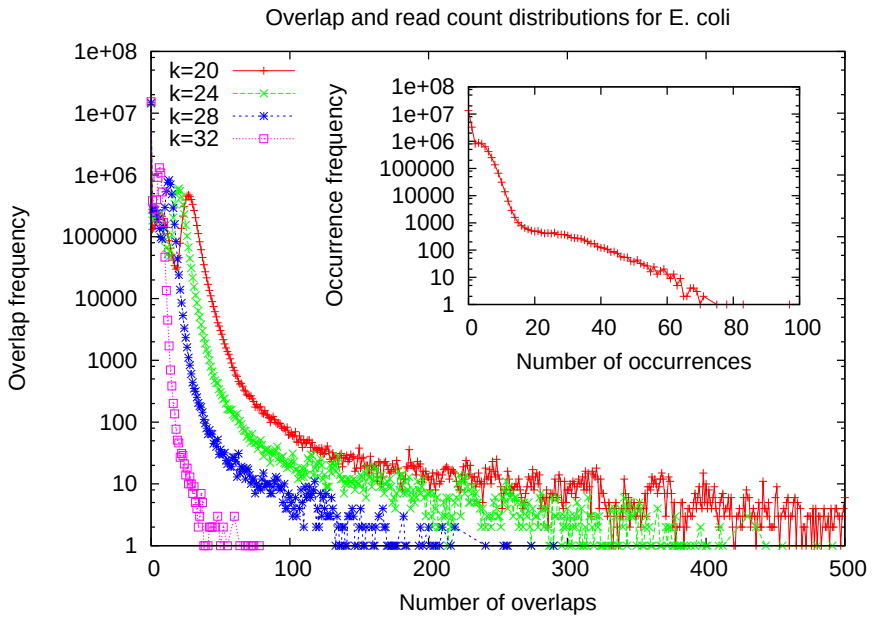


Figure 6.6: Overlap and read count distribution for *E. coli*. Main plot: point (x, y) represents that y number of reads have x number of overlaps. Inset plot: point (x, y) represents that y number of reads occur x number of times in the dataset (duplicates). For both plots the Y axis is in logarithmic scale.

a much lower number of overlaps ($10 \leq x \leq 70$) and they probably correspond to the inter repeat regions. Because the sequencing machines are not error free, there is another class of sequences that contains error and have very few number of overlaps ($x \leq 10$). These reads are responsible for the presence of dead-ends and bubbles in the overlap graph, which represent a big portion of the graph size. From this analysis it is clear that the extreme situations where sequences have very few or high number of overlaps make the overlap graph particularly complex to analyze and sequence assemblers must be carefully designed to explore and disambiguate these graph structures.

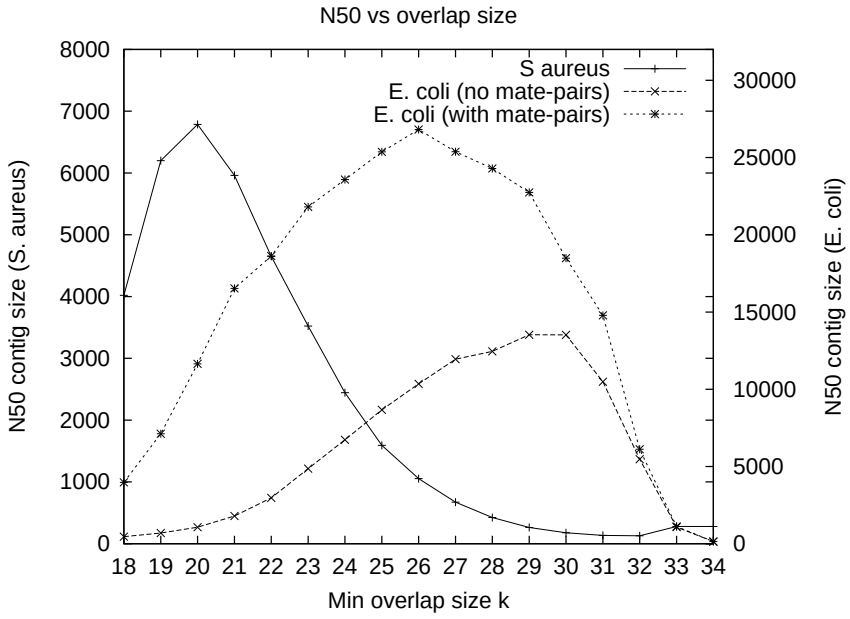


Figure 6.7: Relation between the min overlap parameter k and the N50 contig size for *S. aureus* and *E. coli* (contigs have been assembled with SUTTA).

6.5.2 Trade-off between N50 and Overlap size k

Figure 6.7 shows the relation between the N50 size versus the minimum overlap parameter k for two of the genomic data sets analyzed in this chapter. Clearly there is a trade-off between the number of spurious overlaps and lack of overlaps as the values for k move from small to larger numbers. Increasing the overlap size allows the resolution of more ambiguities, but in turn requires a higher coverage depth to achieve the same N50 value. It is important to emphasize that the optimal value for k depends on the genome structure and coverage (*S. aureus* and *E. coli* have different optimal values) and so it needs to be tuned accordingly. Finally, the availability of mate pairs definitely improves the results and enables assembly of longer contigs for the *E. coli* genome.

6.5.3 Feature-Response curve dynamics

The choice of the minimum overlap parameter k not only affects the estimated length of the assembled contigs but also changes the overall quality of the assembled sequences. In order to show this phenomenon, we have examined the assembly quality as a function of k using the Feature-Response curve. Figure 6.8 shows the dynamics of the FR curve for *E. coli* as a function of the minimum overlap parameter k. Like the plots in figure 6.7, both small and large values of k produce more assembly errors, while the best value lies in the middle range 25-29. There seems to be a phase transition for $k = 33$ and $k = 34$; this is due to the fact that the probability to detect a perfect match overlap of higher size ($k > 32$) becomes more unlikely without increasing the coverage. Both average contig length and N50 value decrease such that more contigs of size smaller than the insert size are created. All these contigs then violate the mate-pair constraints and result in a high number of features/errors.

6.6 Computational performance

Because of the theoretical intractability of the sequence assembly problem and because, in principle, SUTTA's exploration scheme could make it generate an exponentially large number of layouts, SUTTA could be expected to suffer from long running times and high memory requirements. However, our empirical analysis shows that SUTTA has a competitively good performance — thanks to the branch-and-bound strategy, well defined scoring and pruning schemes, and a care-

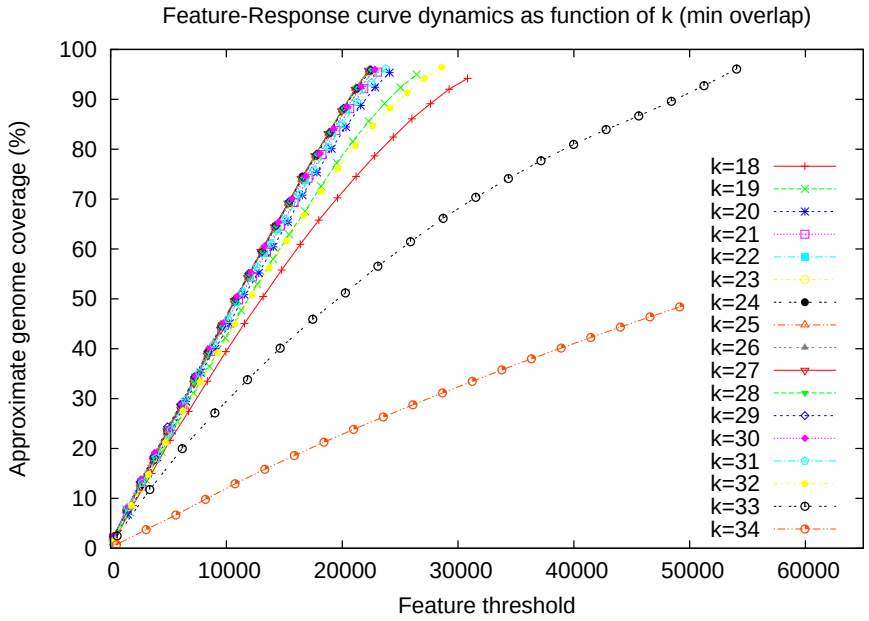


Figure 6.8: Feature-Response curve dynamics as a function of the minimum overlap parameter k for *E. coli* using mate pair data.

ful implementation. SUTTA's computational performance was compared to Velvet, ABySS, EULER-SR, Edena and SSAKE on the *S. aureus* genome using a four quad core processor machine, Opteron CPU 2.5GHz (see table 6.6). SUTTA has an assembly time complexity similar to Edena, SSAKE and EULER-SR. Velvet and ABySS have the best computational performance. Velvet, ABySS and Edena consume less memory than SUTTA. However, SUTTA relies on AMOS to maintain various genomic objects (reads, inserts, maps, overlaps, contigs, etc.), which are not optimized for short reads. At the current stage of development, SUTTA's time complexity increases with mate-pairs constrain computation, but is expected to improve with reengineering planned for the next versions. Finally, we note that typically 2/3 of total SUTTA's running time is dedicated to the

computation of overlaps, leaving only a 1/3 of the total time to assemble the contigs.

	SUTTA	Edena	Velvet	ABySS	SSAKE	EULER
Time (min.)	18m	10m	2m	3m	30m	13m
Memory	3.4Gb	800Mb	600Mb	600Mb	4.2Gb	800Mb

Table 6.6: Assemblers computational performance for *Staphylococcus aureus strain MW2*. Time and memory requirements reported here include both overlapping and assembly steps.

Chapter 7

Integrating Base-Calling, Error Correction and Assembly

7.1 Introduction

Although algorithmic improvements play an important role in sequence assembly, the complexity of the problem is strongly reduced if high quality (low error rate) sequences can be generated. For this purpose, Base-Calling and error correction tools play a significant role in generating highly accurate sequence assemblies.

This chapter starts by presenting a novel DNA base-caller, TotalReCaller [62], designed to interpret analog signals from sequencing machines in terms of a sequence of bases ($\{A, C, G, T\}$), while simultaneously aligning the sequence reads to a source reference (draft) genome, whenever available as it can reduce the error rate. By merging genomic information with raw intensity data TotalReCaller produces high quality sequence reads as well as an alignment at the possible location

in the reference genome. To achieve these objectives, TotalReCaller combines a linear error model for the raw intensity data and Burrows-Wheeler transform (BWT) based alignment into a Bayesian score function, which is then globally optimized over all possible genomic locations using an efficient branch-and-bound (specifically beam search) approach.

This chapter also demonstrates the advantages of a complete de novo pipeline integrating TotalReCaller (base-calling) with SUTTA (sequence assembly) in a Bayesian manner. We have found that this pipeline significantly improves the assembly quality and performance compared to the standard SUTTA pipeline (without error correction) described in chapter 4. Comparison results from the best state-of-the-art assembly algorithms (e.g., SOAPDenovo and ABySS) for short read next-generation technology demonstrate the competitive performance improvement with this new pipeline.

7.2 Base-Calling Challanges

Base-calling takes as input the vector analog time series of signals generated by the sequencing machines, and produces a base-by-base digitized estimate of the underlying DNA sequence, which is most likely to have given rise to that signal. As explained in chapter 1, although next-generation sequencing technologies have reduced the cost and increased the throughput, they pose new challenges in base-calling. In fact, these platforms are error-prone, corrupt signals in the data by non-stationary errors, and generate much shorter reads than those needed for both proper alignment and sequence assembly as well as what used to be routinely

possible with the traditional Sanger sequencers.

Motivated by such challenges, novel base-calling frameworks have been proposed to deal with many unknown sources of noise in the data (see chapter 1 section 1.7). However, these methods have also exposed additional difficulties that must be resolved by better Base-Calling software:

- Over-fitting: Parametric models (e.g., Alta-Cyclic, Ibis, BayesCall, and Rolexa) seem to suffer from over-fitting to the in-sample data and thus are unlikely to be very robust in dealing with varying out-of-sample datasets, even from the same sequencing platforms.

- Computational Cost: All base-callers require preprocessing to learn the error model from training data in order to build a classifier that can then correct the errors in the signal. This preprocessing can be very time consuming (as in the case of BayesCall, Alta-Cyclic and Ibis) and may require a cluster computer facility (as in the case of Alta-Cyclic), thus preventing them from real time base-calling, which would be needed in many clinical applications.

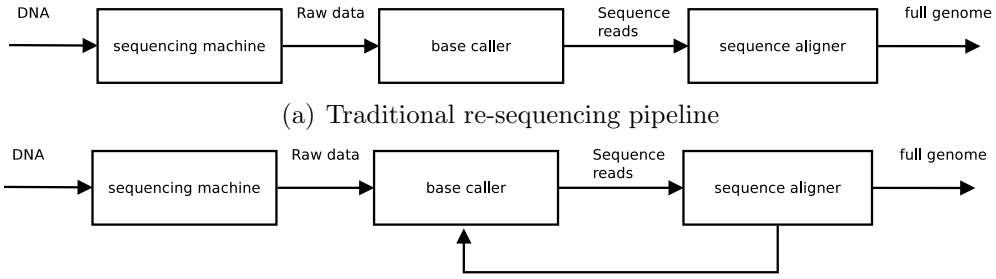
- Training: Since a sub-class of these base-callers (e.g., Alta-Cyclic, Ibis) needs a training library of correct reads, they require both a secondary base caller as well as a sequence aligner to build the training library. However, this concern has been somewhat alleviated in the newer base-callers (e.g., Bustard, BayesCall), which estimate their parameters solely from intensity files.

- Technology Dependence: Many base-callers (e.g., Alta-Cyclic, Ibis and BayesCall) use a detailed parametric model to describe the signal distortion as a function of successive cycles. Such models require hard-wiring specific knowledge of the underlying sequencing technology into the algorithm, thus making it harder to customize the base-caller to support other platforms.

Currently, the major application of the next-generation sequencing technologies is thought to be in re-sequencing. Despite the obvious centrality of alignment in these applications, traditional base-callers have avoided performing alignment until the end of the base-calling process. In fact, the typical pipeline for a re-sequencing process traditionally consists of two sequential steps (see figure 7.1(a)): (1) Base-Calling: each single base of the read is called according to the intensity signal and error profiles; (2) Alignment: sequence reads are aligned to a reference genome. Because the base calling process is error-prone, and because correct alignment to the reference genome is non-trivial, high coverage is required in order to reduce the errors in re-sequencing and recover the true full DNA sequence.

Motivated by the limitations of current base-callers and the challenges of re-sequencing, we have designed a new base-caller, TotalReCaller, which substantially ameliorates the problems discussed above and significantly improves the quality of reads by injecting knowledge of the reference genome into the base-calling step. TotalReCaller replaces the typical sequential re-sequencing protocol into a combined pipeline (see figure 7.1(b)) that has the ability to concurrently perform base-calling, alignment, error correction and SNP detection. Although in

this chapter we address base-calling for the Illumina platform, the method is, in principle, applicable to any other sequencing technology. Adaptation to a different sequencing platform only requires re-designed score functions to encapsulate error correction and alignment – and, thus, accommodate the different features and error profiles of the variant system in a technologically extensible manner.



(a) Traditional re-sequencing pipeline

(b) Proposed re-sequencing pipeline

Figure 7.1: Block diagrams for re-sequencing pipelines: (a) open-loop (no feedback) and (b) closed-loop with feedback.

7.3 Source of Errors in Illumina Raw Sequencing Data

As reported previously in [24], there are four dominant sources of noise affecting the intensities generated by Illumina:

1. *Crosstalk:* The intensity channels are not independent. This interdependence is due to the fact that the fluorescent markers for A, C and G, T emit photons with similar wavelength, get excited by the same lasers and fluorescent markers from one cycle can only be chemically partially removed

("washed") before the cycles for the next nucleotides (all performed in the same flow cell). Figure 7.2 clearly shows this effect: calling the bases A and G from their intensity channels does not pose a severe problem at least for the first 40 cycles. In contrast, channels C and T appear hopelessly disrupted soon after the start and within the first few cycles.

2. *Fading:* With successive cycles, the absolute intensity of light emitted from the cluster of DNA strands diminishes because fluorescent markers are only able to bind to fewer and fewer strands within the clusters. This produces a low signal-to-noise ratio (SNR) as shown in figure 7.2. which worsens precipitously with the cycle numbers – attributed primarily to polymerase desynchronization and its low processivity.

3. *Lagging (Phasing):* Some strands in the clusters start to lag behind the population, as in each cycle some of the polymerases fail to operate synchronously, but then rejoin the other strands in subsequent cycles, whence producing ambiguous intensity values. Eventually, the correct channel gets obscured by the other wrong channel intensities, leading to wrong base-calls.

4. *Leading (Pre-phasing):* Some strands in the clusters start to lead ahead of the population, which also causes ambiguous and incorrect channel intensity values in a fashion analogous to lagging.

These noise factors dominate and affect the signal differently in different cycles. In the first few cycles, crosstalk is the major source of base-call errors. However, in later cycles fading, lagging and leading prevail. We have observed

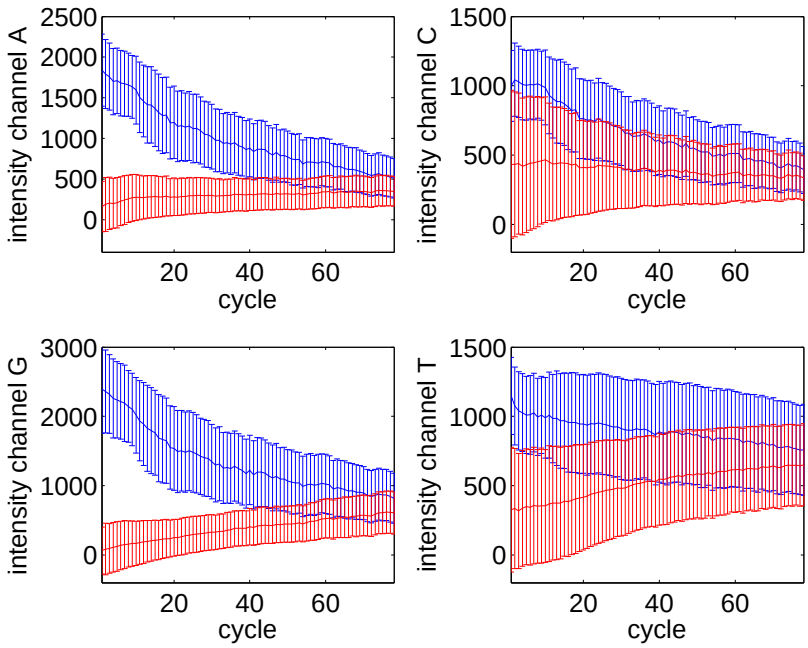


Figure 7.2: Statistics for high and low intensity levels depicted with their means and standard deviations for four channels – one for each base $B \in \{A, C, G, T\}$. A high intensity level [blue] (with a value above a threshold) in one of the channels indicates that this base should be called at a given cycle. A low intensity level [red] (below threshold) in a channel means that this base should not be called at a given cycle. In a "good" set of intensities, it is expected that one channel is higher than a threshold, while all other three are lower than it. The panels depict that in later cycles the low and high intensities become increasingly indistinguishable, which causes erroneous base-calling for distal positions.

that lagging often causes many false-positive insertions in the distal extending-end of sequence reads. In later cycles, intensities measured in cycle k reflect more and more what would have been the value in cycle $(k-1)$. This process leads directly to "base-stuttering," occurring much more frequently after some threshold value for k, the cycle number. This dynamic can be modeled by a step

function appearing randomly but more frequently in later cycles, thus making it extremely difficult to analyze. This effect has important implications for the succeeding alignment step, since many popular short-read sequence-aligners cannot align gapped sequence reads [56], [58]. We have also observed the effects of leading on signal to noise ratio to be negligible in comparison to the other three (crosstalk, fading and lagging).

7.4 TotalReCaller

TotalReCaller combines the knowledge from sequencers' raw intensity data with information from a reference genome (when available). In other words, it generates the most plausible m-base string (out of 4^m possibilities) that is most likely to have generated the channel intensity data, and also most likely to have originated at some location of the reference genome. Like SUTTA, TotalReCaller tames the worst-case exponential complexity of the implementation by using a beam search strategy strategy (an adaptation of the branch-and-bound method). Specifically, this strategy is used to concurrently extend multiple high quality reads that are immediately validated not only by the intensity signals but also by the likely alignments to a reference genome (thus the genome providing a weak prior to a Bayesian inference). This scheme builds on a rigorously defined Bayesian score function that accounts for both — thereby, resulting in a single score to quantify the quality of a given sequence read. In order to execute this task, TotalReCaller implements four different components that are described in detail in the following sections: (1) linear error model; (2) base-by-base sequence

alignment; (3) beam search read extension; and, finally, (4) score function.

7.4.1 Linear error model and filter

A simple linear model has been devised to correct errors resulting from crosstalk, fading and cycle-dependent synchronous-lagging. The model is based on a cycle-dependent transition matrix (thus dynamic) in order to filter the raw intensity channels. The linear-algebraic model for crosstalk and fading is first derived and then extended to include lagging. Let $\boldsymbol{I}_k = (I_A^k \; I_C^k \; I_G^k \; I_T^k)^\top$ be the vector of the four raw intensity channels. In order to model crosstalk in cycle $k \in \mathbb{N}$, we introduce the crosstalk matrix $\boldsymbol{A}_k \in \mathbb{R}^{4\times 4}$ and the crosstalk-free channels $\boldsymbol{X}_k = (X_A^k \; X_C^k \; X_G^k \; X_T^k)^\top \in \mathbb{R}^4$. We model their relationship simply by the following formula:

$$\boldsymbol{I}_k = \boldsymbol{A}_k \cdot \boldsymbol{X}_k. \tag{7.1}$$

Since a separate crosstalk matrix is computed for every cycle k, the intensities are implicitly normalized, thus additionally accounting for fading.

Lagging is then modeled by introducing a dependency between the current cycle and the previous cycle, resulting in:

$$\begin{pmatrix} \boldsymbol{I}_{k-1} \\ \boldsymbol{I}_k \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{k-1} & 0 \\ \boldsymbol{\Upsilon}_k & \boldsymbol{A}_k \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{X}_{k-1} \\ \boldsymbol{X}_k \end{pmatrix} \tag{7.2}$$

$$\Rightarrow \begin{pmatrix} \boldsymbol{X}_{k-1} \\ \boldsymbol{X}_k \end{pmatrix} = \underbrace{\begin{pmatrix} \boldsymbol{A}_{k-1} & 0 \\ \boldsymbol{\Upsilon}_k & \boldsymbol{A}_k \end{pmatrix}^{-1}}_{\boldsymbol{G}_k} \cdot \begin{pmatrix} \boldsymbol{I}_{k-1} \\ \boldsymbol{I}_k \end{pmatrix}, \tag{7.3}$$

where $\boldsymbol{\Upsilon}_k \in \mathbb{R}^{4\times4}$ describes the coupling between $\boldsymbol{I}_k$ and $\boldsymbol{I}_{k-1}$. A matrix inversion results in a simple transition matrix $\boldsymbol{G}_k$, which is then used to filter the raw intensity channels. The elements of the matrices $\boldsymbol{\Upsilon}_k$ and $\boldsymbol{A}_k$ are obtained by statistical analysis of the intensities, using a library of correct reads similar to the training sets used for the parameter estimation of the support vector machines in Alta-Cyclic and Ibis. However, notice that for TotalReCaller this is not a computationally expensive task since it only solves a simple linear system. Also, notice that, while training data set is used here to simplify the learning phase of TotalReCaller, it is not absolutely necessary for its operation, as TotalReCaller can adaptively tune these parameters.

After applying the filter to the set of raw intensities (see figure 7.2), we were able to significantly improve the quality of the intensity channels, as shown in figure 7.3. The error model and filter could easily be extended to include leading (pre-phasing). It was decided to refrain from including it in the current implementation, since it appeared to increase the computational costs without balancing it by a further proportional improvement in the quality of intensity information.

With the intensity channels suitably filtered, we needed a metric to compare the intensity channels among one another. For this purpose we focused on the conditional probabilities $P_k(X_B \mid B)$ and $P_k(X_B \mid \neg B)$ with $B \in \{A, C, G, T\}$. $P_k(X_B \mid B)$ denotes the conditional probability of the filtered intensity X_B of channel B, given that the correct base to call is base B, whereas $P_k(X_B \mid \neg B)$ denotes the conditional probability of the filtered intensity X_B, given that the correct base to call is not base B. Since the filtered channels X_B are independent

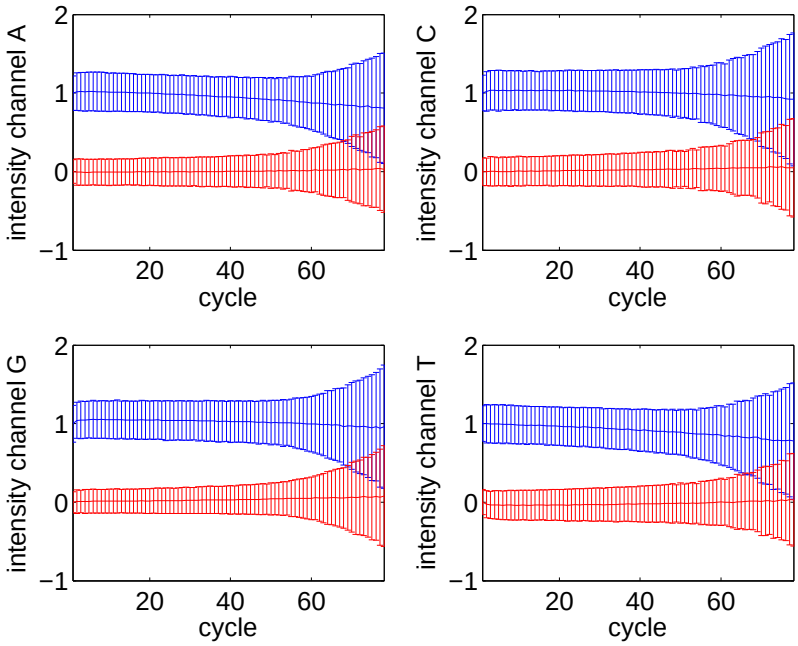


Figure 7.3: Filtered intensity channels and separation. Crosstalk and lagging are corrected using a linear filter, developed here. The high and low intensity levels are now cleanly separated for the first 60 cycles.

of each other, we can approximate these probabilities assuming that they are normally distributed (with subscript k suppressed to avoid clutter),

$$X_B \mid B \sim \mathcal{N}(\mu_B, \sigma_B) \quad \text{and} \quad X_B \mid \neg B \sim \mathcal{N}(\mu_{\neg B}, \sigma_{\neg B}). \tag{7.4}$$

That is:

$$P_k(X_B \mid B) = \frac{1}{\sqrt{2\pi}\sigma_B} \exp\left(-\frac{(X_B - \mu_B)^2}{2\sigma_B^2}\right), \tag{7.5}$$

$$P_k(X_B \mid \neg B) = \frac{1}{\sqrt{2\pi}\sigma_{\neg B}} \exp\left(-\frac{(X_B - \mu_{\neg B})^2}{2\sigma_{\neg B}^2}\right). \tag{7.6}$$

The means together with their standard deviations have already been presented in figure 7.3, in order to show the workings of the linear error model and filter. Note that although it is reasonable to assume $X_B \mid B$ to be normally distributed, it seems less justifiable to take $X_B \mid \neg B$ to be normally distributed as well, because the condition in this case is the disjunction of the three other bases. However, we have experimentally computed the distribution for $X_B \mid \neg B$ and found out that it follows a narrower distribution than the normal distribution. Because of that, forcing $X_B \mid \neg B$ to be also normally distributed is actually less accurate. Nevertheless, we decided to keep that assumption to design a score function that would be simpler and faster to compute.

7.4.2 Base-by-base sequence alignment

The key idea of TotalReCaller is to perform alignment while the sequence is being base-called. The partially generated sequences, which are grown one base at a time, must be aligned back to the reference genome. To account for this computationally intensive task, we designed an efficient base-by-base aligner that is based on a suffix tree search algorithm. Inspired by the many Burrows-Wheeler based short read sequence aligners (Bowtie [56], SOAP2 [59], BWA [58]), we constructed our base-by-base aligner essentially on the same principles, specifically the Ferrangina-Manzini search algorithm [26] and the Burrows-Wheeler transformation [17]. Ferrangina and Manzini showed how the suffix tree of a reference genome can be accessed through its Burrows-Wheeler transformation (BWT), which does not require more memory than the reference genome itself. Thus

searching for a (partial) sequence read in a BWT reference can be performed very efficiently. In addition, not only information about the existence but also the number of occurrences in the given reference of the (partial) sequence read can be computed concurrently.

Sequence	Frequency f_B	$P(B)$	$P(\neg B)$
ACGAC**A**	100	0.10	0.90
ACGAC**C**	20	0.02	0.98
ACGAC**G**	500	0.50	0.50
ACGAC**T**	380	0.38	0.62

Table 7.1: Probabilities from FM search for each base preceded by "ACGAC".

Note that sequence aligners usually only estimate whether and where a sequence read is located in a given reference. With alignment information for base-calling, we are also able to use the alignment frequency of a partial sequence read. For the example, suppose that the (partial) sequence "ACGAC" is contained in a reference 1000 times. We can use the FM search to count how often the sequences "ACGAC**B**" with $B \in \{A, C, G, T\}$ are contained in the reference, from which we can then compute the probability $P_k(B)$, that the next base (at cycle k) in the sequence is B (similarly to the k-gram model used in natural language analysis):

$$P_k(B) = \frac{f_B}{f_A + f_C + f_G + f_T}, \quad P_k(\neg B) = 1 - P(B) \tag{7.7}$$

Table 7.1 shows a complete example how the base probabilities are computed using the FM search.

So far we have introduced the intensity filter and base-by-base alignment components, what remains to be shown (in the next section) is how to combine them in order to score and prune the candidate solutions.

7.4.3 Beam search read extension

Like SUTTA, TotalReCaller uses a Branch-and-Bound (specifically a variant called beam search) strategy to combine intensity and alignment information by sequentially constructing a tree of hypothetical sequences. In order to reduce the computational, complexity the tree is only partially constructed and repeatedly evaluated. At each cycle the tree grows in depth, with each node in the tree (not just the leaves) representing a possible sequence. In order to be able to recover the best sequence out of the N_k possible sequences, every node is scored according to a Bayesian score function immediately upon creation. This score function combines terms for intensity and alignment information and is described in the next section. The maximally likely estimate for the correct sequence read(s) is thus obtained by simply choosing the node with the (globally) highest score. Without any pruning, the tree could grow exponentially in the number of cycles: at cycle k, $|N_k| = 4^k$ sequence reads must be evaluated. Moreover, since asynchronous lagging causes incorrect insertions into the sequence read, we need to consider deletions as a 5th child, which means that a tree $\overline{N}_k$, with $|\overline{N}_k| = 5^k$ sequence reads must be created and evaluated. Since TotalReCaller dynamically prunes unpromising sequences based on the evaluation of the score function in a beam search scheme [12], the worst-case complexity is rarely encountered in practice.

Note that the interesting special situations, where the exponential worst-case behavior would be exhibited, occur when the sequencer is extremely noisy and/or when the reference is incorrect (or highly error-ridden), thus producing exponentially many plausible hypothetical sequence reads – a judicious solution in these cases would then involve terminating the sequence read at a smaller read-length or rejecting it outright. The algorithm is described as a sequence of three consecutive steps that are repeated once for each cycle, as described in Algorithm 5.

Algorithm 5: TotalReCaller - beam-search pseudo code

Input: Set of intensities I_k and reference genome G
Output: Read sequence

1 **repeat**
2 **Branching:** For each sequence in the solution space N_{k-1} all four possible successor sequences are generated, resulting in the solution space N_k (note that at this point $N_{k-1} \subset N_k$);
3 **Bounding:** Each sequence in N_k is evaluated according to the score function g (combining intensity and alignment information);
4 **Pruning:** All but the best (highest score) $l \in \mathbb{N}$ sequences are pruned, thus reducing the size of N_k to $|N_k| = l$;
5 **until** *no more cycles* ;
6 **return** *Best sequence read according to g*;

Note that by reducing the computational complexity through bounding the solution space, we are no longer guaranteed to generate the optimal solution. However, in practice, the accuracy of the algorithm's outputs seem to be only slightly affected. Wherever necessary, the computational cost can be traded off for higher accuracy by setting a parameter that controls the width of the beam search.

7.4.4 Score functions

To complete the description of TotalReCaller, we need to define the score function used to evaluate the quality of the candidate sequences in the tree. From Bayes' theorem, it is possible to estimate the probability P_k that a specific base $B \in \{A, C, G, T\}$ is indeed the correct base to call at cycle k, given the filtered intensity vector $\boldsymbol{X}_k$:

$$P_k(B \mid \boldsymbol{X}_k) = \frac{P_k(\boldsymbol{X}_k \mid B)P_k(B)}{P_k(\boldsymbol{X}_k)} \qquad \text{with } B \in \{A, C, G, T\} \tag{7.8}$$

$$= \frac{P_k(\boldsymbol{X}_k \mid B)P_k(B)}{P_k(\boldsymbol{X}_k \mid B)P_k(B) + P_k(\boldsymbol{X}_k \mid \neg B)P_k(\neg B)} \tag{7.9}$$

$$= \frac{1}{1 + \frac{P_k(\boldsymbol{X}_k|\neg B)}{P_k(\boldsymbol{X}_k|B)} \cdot \frac{P_k(\neg B)}{P_k(B)}} \tag{7.10}$$

Since for our purposes it is sufficient to have a quantitative measurement (not a probability) to compare all different solutions to one another, we introduce a simplified score function f_{score} which is based on $P_k(B \mid \boldsymbol{X}_k)$:

$$f_{score} = \underbrace{\log\left(\frac{P_k(\boldsymbol{X}_k \mid B)}{P_k(\boldsymbol{X}_k \mid \neg B)}\right)}_{\text{intensities (eqn. 7.5)}} + w_{align} \cdot \underbrace{\log\left(\frac{P_k(B)}{P_k(\neg B)}\right)}_{\text{alignment (eqn. 7.7)}} \tag{7.11}$$

The score function consists therefore of two parts, both of which can be computed independently according to the sections discussing the intensity filter and the base-by-base alignment algorithm. The weight $w_{align} \in [0, 1]$ permits a user-defined control over the impact of the alignment on the overall score, thus enabling the user to adjust the Bayesian bias for a particular application.

7.5 Base-Calling Results

Several approaches have been developed to improve the read quality of Illumina reads. They use a variety of error models, statistical inference algorithms and supervised learning methods. Currently there are six major base-callers for Illumina sequencing machines (including TotalReCaller), which are presented in table 7.2. In the following we compare the performance of these base-callers[1] according to the standard metrics that have been used in the past: namely, base-calling error rate, alignment rate and base-calling speed. Base-calling results for TotalReCaller are presented with different weights on reference-alignment with respect to intensity information. Three different datasets have been used for comparison (listed in figure 7.3).

Name	Institute	Reference	Technology
TotalReCaller	New York University	[62]	Beam Search
Bustard	Illumina	n.a.	Linear Model
Alta-Cyclic	CSHL	[24]	SVM
BayesCall	UC Berkley	[42]	Graphical model
Ibis	Max Planck	[51]	SVM
Rolexa	Universit de Lausanne	[85]	Probabilistic model

Table 7.2: List of available Base-callers for Illumina sequencing technology including TotalreCaller.

7.5.1 Error rates

The base-calling error rate measures the quality per cycle (*bp* position) of the sequence reads, produced by a given base-caller (see figure 7.4). In order to

[1] As we lacked the required hardware and software, we were unable to compare against Alta-Cyclic.

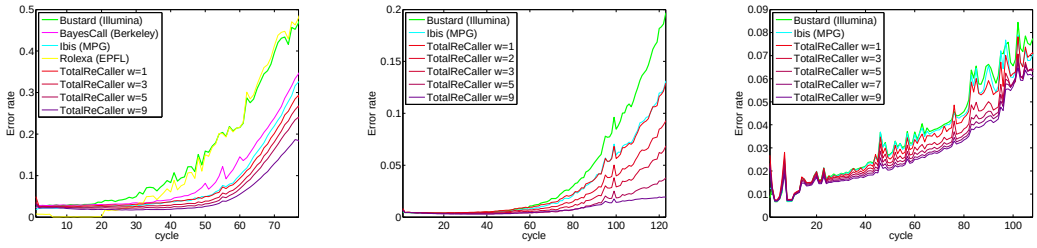
Genome	Genome size	#Clusters	#Cycles	#Aligned	Aligner	Description
phiX	$\sim 5.4kBp$	$11,803,171$	78	87%	BLAT	One lane of a *Genome Analyzer I* run.
E. coli	$\sim 4.5MBp$	$35,027,442$	125	65%	BLAT	The 2nd pair of a paired-end *Genome Analyzer II* run (one lane).
poplar	$\sim 420MBp$	$31,445,866$	109	32%	Bowtie	The 1st pair of a paired-end *Genome Analyzer II* run (one lane).

Table 7.3: Data sets used to evaluate and compare TotalReCaller's performance to its peers: *phiX*, *E. coli* and *poplar*.

generate error rates based on the same set of reads for each of the base-callers, we aligned all Bustard reads to the respective reference genome in order to create a set of "correct reads". We then perform a base by base comparison between this set of "correct reads" and the sequence reads created by each of the base-callers, resulting in an error rate for each position (cycle) in a sequence read. Since we used the output of Bustard to create the set of correct reads we introduced a bias, favoring Bustard. For the small genomes, *phiX* and *E. coli*, we used the aligner BLAT [46], which allows accurate, gapped alignment to create the set of "correct reads" (see table 7.3). For the poplar dataset we used Bowtie [56] to create the set of "correct reads". We had to use Bowtie instead of BLAT in order to properly handle the current draft of the poplar [99] genome[2], which is of relatively lower quality in comparison to *E. coli* and *phiX*, e.g. poplar consists of many contigs (out of 2518) that have not yet been phased to a scaffold. The low quality, in conjunction with the length and complex structure, of the poplar genome results in an unusually large number of false positive alignments, which,

[2]Populus trichocarpa v2: `http://www.phytozome.net/poplar` Feb. 2011

when analyzed by a sensitive aligner, makes it impossible to create a valid set of "correct reads." Since Bowtie and other suffix tree based algorithms are in general less sensitive than BLAST-like [2] alignment algorithms (e.g., they do not allow gapped alignment), they produce fewer but, especially in case of a "bad" reference genome, many more accurate sets of "correct reads." The base-calling error rates based on the reads produced by the base-callers and the set of "correct reads" can be found in figure 7.4. Based on the previous discussion, it is safe to conclude that the error rates for poplar dataset may be used only for a qualitative (and not a quantitative) comparison.



(a) *PhiX* (b) *E. coli* (c) *Poplar*

Figure 7.4: Sequence read error rates per cycle for each of the three datasets (see table 7.3). In order to compute the error rates, the sequence reads, generated by each of the base-callers, were compared base-by-base (cycle-by-cycle) to a corresponding "correct read". As can be seen, sequence reads generated by TotalReCaller have a significantly lower base calling error rate in comparison to all other base callers. Furthermore, the error rate can be controlled by choosing an alignment weight w_{align}. Note also that reads produced by the GAIIx Illumina machines are in general of higher quality than those generated by the older GAI machines. The fluctuations in the poplar error rates are primarily due to the poor quality of the poplar reference genome which led to a limited set of "correct reads". It was not possible to present statistics for BayesCall and Rolex for *E. coli* and *poplar*, since these base-callers do not support the newer Illumina file formats (RTA pipeline).

Genome	Base-caller	$t_{training}$	$t_{calling}$	alignment rate
	Bustard	-	-	15.29%
	Ibis	$\sim 8h$	$\sim 2h$	31.80%
	BayesCall	$\sim 50h$	$\sim 32h$	32.66%
	Rolexa	$\sim 2h$	$\sim 90h$	11.40%
phiX	TotalReCaller ($w = 0.1$)			40.50%
	TotalReCaller ($w = 0.3$)	$\sim 1h$	$\sim 17h$	45.45%
	TotalReCaller ($w = 0.5$)			49.57%
	TotalReCaller ($w = 0.9$)			57.94%
	Bustard	-	-	28.70%
	Ibis	$\sim 20h$	$\sim 10h$	36.31%
	TotalReCaller ($w = 0.1$)			46.45%
E. coli	TotalReCaller ($w = 0.2$)			55.77%
	TotalReCaller ($w = 0.3$)	$\sim 1h$	$\sim 28h$	64.37%
	TotalReCaller ($w = 0.5$)			77.19%
	TotalReCaller ($w = 0.9$)			87.47%
	Bustard	-	-	25.55%
	Ibis	$\sim 16h$	$\sim 9h$	25.97%
	TotalReCaller ($w = 0.1$)			25.60%
poplar	TotalReCaller ($w = 0.3$)			27.74%
	TotalReCaller ($w = 0.5$)	$\sim 1.5h$	$\sim 23h$	29.83%
	TotalReCaller ($w = 0.7$)			31.84%
	TotalReCaller ($w = 0.9$)			33.62%

Table 7.4: Speed and alignment comparison. In the third column, "$t_{training}$" tabulates the approximate duration of the training phase, the parameter estimation, for each of the base-callers. In the fourth column, "$t_{calling}$" tabulates the duration of the actual base calling. In the last column, "alignment rate" shows the percentage of how many of the reads, called by a specific base-caller, could be aligned back to the reference genome using Bowtie [56]. As an example, for *E. coli* after $1.5h$ of training TotalReCaller, with an alignment weight of $w_{align} = 0.5$, calls $35,027,442$ reads with a length of $125BP$ in $28h$ hours, which corresponds to $43\frac{BP}{ms}$. Out of these $3.5 \cdot 10^7$ reads, 77.19% could be aligned back to the *E. coli* reference genome. Base-calling was performed on the datasets presented in table 7.3 utilizing a single CPU thread. No precise times are given, since they vary depending on runtime parameters. In comparison to Ibis, Rolexa and BayesCall, TotalReCaller uses a faster training phase. The relatively long base-calling time for TotalReCaller can be accounted for by the time TotalReCaller implicitly spends on genome-alignment while base-calling. For all three datasets, it is shown that a bigger percentage of reads can be aligned to the reference, if TotalReCaller's strategy is used. Note also that the higher the alignment weight w_{align}, the more reads can be aligned.

7.5.2 Alignment rate and base-calling speed

The alignment rate (or mapping rate) describes how many reads produced by a specific base-caller can be aligned back to the source reference genome. This rate provides an important metric, because it quantifies how many of the estimated reads possess good enough quality to permit high level genome analysis (such as SNP detection and assembly). Of course, the alignment rate, similarly to the base calling error rate, depends to large extent on the specific sequence alignment tool that is used. In table 7.4 we present the alignment rates for the sequence reads produced by the different base-callers for each of the three datasets. The reads were aligned using Bowtie with conservative parameters (low sensitivity). Table 7.4 also shows the approximate base-calling speed for each base-caller.

7.5.3 SNPs specificity and sensitivity

Notwithstanding TotalReCaller's relative performance advantage in terms of error and alignment rates, it may be questioned whether TotalReCaller's bias due to reference-based Bayesian prior is the source of this advantage, and could affect (perhaps, adversely) its single nucleotide polymorphism (SNP) sensitivity and specificity. Specifically, since TotalReCaller injects knowledge from a reference genome into the base-calling process, it is possible that sequence reads at true SNP-positions (containing information from positions where the reference genome differs from the genome that is sequenced) are called incorrectly. Thus, it is vital to examine what happens when a sequence read is called, if that sequence contains one or more SNPs with respect to the reference genome. In order to assess

TotalReCaller's bias toward the reference genome, particularly with respect to reference-independent basecallers, we define two important statistics: *Specificity* (SPC_k) (also known as true negative rate, TNR) and *Sensitivity* (SNS_k) (also known as true positive rate, TPR) for each cycle k:

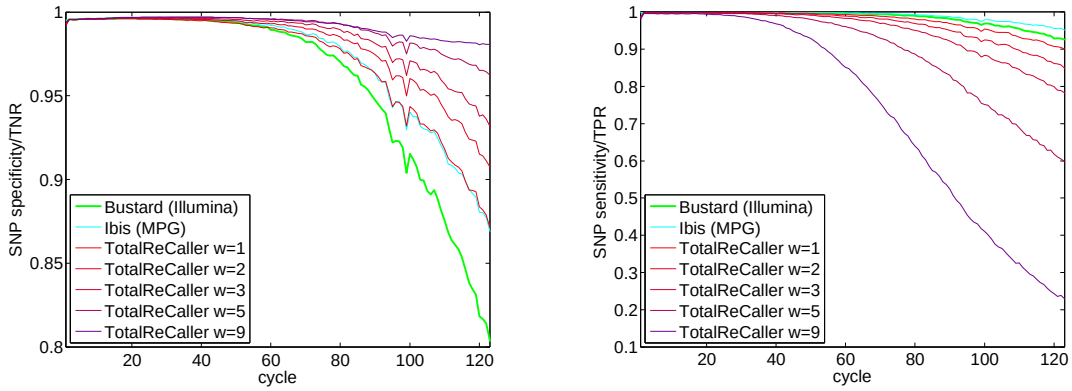
$$\mathrm{SPC}_k = \frac{\text{true negatives}_k}{\text{true negatives}_k + \text{false positives}_k}, \tag{7.12}$$

$$\mathrm{SNS}_k = \frac{\text{true positives}_k}{\text{true positives}_k + \text{false negatives}_k} \tag{7.13}$$

These statistics are based on artificially SNP-inserted reference genomes. On average we inserted one SNP every n bases randomly into each of the reference genomes (see table 7.3), where n was chosen to be equal to the number of cycles available for the given intensity data. Then the SNP-inserted genome was used as a reference for TotalReCaller. Although all other base callers ignore side-information, e.g., information in a reference genome, these same statistics can be computed for all of them for comparison purposes. For those basecallers, sensitivity and specificity depend only on their raw error rates.

Figure 7.5 shows the effect of the alignment on base-calling, as the weights w_{align} are varied for the *E. Coli* dataset. TotalReCaller's specificity is higher in comparison to all other base-callers for each of the presented alignment weights. TotalReCaller's sensitivity for a low alignment weight is however surpassed by Ibis for the *E. coli* dataset. Increasing TotalReCaller's alignment weight increases the specificity and reduces the sensitivity. Considering the significant higher align-

ment rate of TotalReCaller (see table 7.4), the loss of sensitivity with increasing w_{aling} is more than compensated.



(a) *E. coli*-SPC (b) *E. coli*-SNS

Figure 7.5: SNP specificity (SPC) and sensitivity (SNS) for *E. coli*. These figures show the effect of the alignment on base-calling, as the weights w_{align} are varied. SNP specificity (SPC) measures the rate at which a difference between a read and its reference represents a SNP and not a base calling error. SNP sensitivity (SNS) measures the rate of called SNPs with respect to all SNPs that should be called.

7.6 Error Correction during Base-Calling

The general technique adopted for error correction by next-generation sequence assemblers is based on k-mer frequency analysis. The basic idea is that, for deep sequencing, the correct k-mers appear multiple times in the reads set, while random sequencing errors produce k-mers with low frequency. For example, using such frequencies, SOAPdenovo analyzes each read in the dataset to infer potential erroneous sites of low-frequency k-mers. The impact of changing each erroneous site to the other three allele types is then tested and changes are allowed only

if the new k-mer results in higher frequencies. Other well-known assemblers, such as ALLPATH-LG and EULER-SR, use similar strategies. Since correcting errors can significantly reduce the complexity of assembling the reads, standalone software programs have been recently released that focus only on this task, e.g., Quake [45] and SHREC [89]. Although TotalReCaller was initially designed to perform base-calling and alignment in a combined re-sequencing framework, its range of applications goes beyond that. In this section we will show how it can be used to correct sequencing errors even in de novo sequencing projects.

In addition to the intensities, TotalReCaller requires an input reference genome to perform the base-calling. However, the reference genome does not need to be the correct one or complete (gap-free). Even with a low quality draft genome, TotalReCaller can significantly improve the quality of the reads during base-calling. The draft genome plays the role of a prior knowledge in Bayesian inference. In the specific case of genome sequences, the draft genome represents a more reliable vocabulary of longer words that can be used to correct the errors in the raw intensities. Table 7.5 shows the number of perfect reads (no errors in alignment) base-called by Bustard and TotalReCaller on the full *E. coli* dataset. Both the absolute number and percentage of perfect reads generated by TotalReCaller is higher then Bustard. Also note that this is true even when TotalReCaller uses the draft genome created by SUTTA. Although the alignment rate for the first mate improves only slightly, the number of perfect second mate is significantly higher. This has a strong implication during the assembly process, since more mates can be used to resolve repeat sequences in the genome.

Base-caller	# perfect reads	alignment rate
Bustard (1st mate)	13443953	54.82%
Bustard (2nd mate)	4855602	19.80%
Draft genome:		
TotalReCaller (1st mate)	19412596	58.17%
TotalReCaller (2nd mate)	11226130	33.64%
Real genome:		
TotalReCaller (1st mate)	19749991	59.18%
TotalReCaller (2nd mate)	11335611	33.97%

Table 7.5: Alignment rate for Bustard and TotalReCaller.

Unlike other sequence assemblers, SUTTA does not include any error correction preprocessing step. So we have designed the following pipeline to take advantage of both SUTTA and TotalReCaller capabilities:

1. Draft Assembly: Using SUTTA (or any other sequence assembler) generate a draft assembly using the available reads.

2. Base-calling & Error correction: given the reads intensity files and the draft assembly (generated in step 1), run TotalReCaller to generate a new set of reads with higher accuracy.

3. Sequence Assembly: Run SUTTA on the new set of reads generated in step 2 to create an improved assembly.

7.6.1 Assembly results

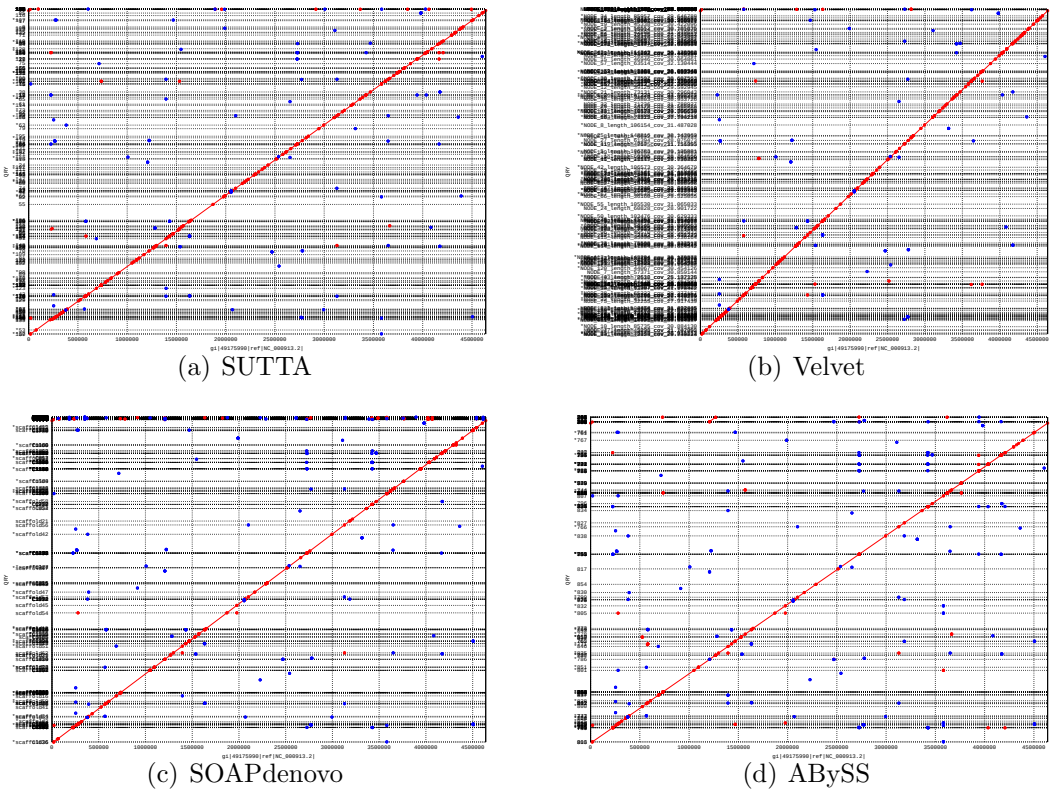
We have tested this pipeline on the Illumina *E. coli* dataset presented in table 7.3. Note that current Illumina software can filter the data by removing reads that do not pass the GA analysis software called `Failed_Chastity`. To stress

test the assemblers on harder datasets, in this study we use the full output of the machine, usually contained in the `export` file. This dataset consists of 49 million 125 bp long reads, for a total coverage 1320X. Since such an high coverage is not typically available for larger genomes, we have subsampled only 100X coverage for comparing the results.

Assembler	#correct	#errors (μ kbp)	#ctgs≥10K (kbp)	N50 (kbp)	Max (kbp)	Mean (kbp)	Cov. all (%)	Cov. correct (%)
SUTTA (exp.)	339	49 (13.8)	147 (37.9%)	24.1	105.6	11.6	97.4	82.7
SUTTA (draft)	168	21 (20.9)	100 (52.9%)	54.6	221.5	24.1	98.2	88.6
SUTTA (ref.)	154	25 (31.4)	86 (48.0%)	71.7	141.6	25.4	98.2	81.3
SOAPdenovo (ctg)	245	80 (18.6)	52 (42.3%)	35.7	100.1	14.1	98.4	66.3
SOAPdenovo (scaf)	106	17 (99.6)	53 (45.3%)	117.6	312.5	37.1	99.3	61.9
ABySS	92	13 (80.9)	54 (49.5%)	134.4	312.5	40.7	102.9	79.7
Velvet	126	60 (32.1)	100 (53.8%)	54.8	148.8	24.5	98.5	56.9

Table 7.6: Assembly results (contigs) for *E. coli* dataset (100X 125bp reads from one lane of Genome Analyzer II). A contig is defined to be correct if it aligns to the reference genome along the whole length with at least 95% base similarity.

Table 7.6 shows a comparison of the assemblies obtained by SUTTA both on the original read set (created by Bustard) and the error corrected set (base-called by TotalReCaller). SUTTA's performance significantly improves on the new reads generated by TotalReCaller. For comparison, we have tested some of the best assemblers for short read technology on the E. coli dataset, specifically SOAPdenovo, ABySS and Velvet. The results are reported in table 7.6. Since the reads are already 125 bp long, only contigs with size ≥ 200 have been considered in the comparison. A contig is defined to be correct if it aligns to the reference genome along the whole length with at least 95% base similarity. Inspecting the results in the table it is clear that SOAPdenovo and ABySS are particularly successful in assembling long contigs, in fact their N50 statistic is the highest. However the assembly quality is inferior to SUTTA: if only correct contigs are



(a) SUTTA (b) Velvet

(c) SOAPdenovo (d) ABySS

Figure 7.6: Dot plots for the *E.coli* assemblies produced by SUTTA, Velvet, SOAPdenovo and ABySS. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

aligned to the reference genome, the total coverage of SOAPdenovo and ABySS are respectively 66.3% and 61.9%, while SUTTA achieve a coverage $\geq$ 80% in all instances. This might be due to the different assembly strategy adopted: both SOAPdenovo and ABySS first create a set of contig using solely the read sequences and only later, in a second step, extend and merge the contigs using the mate-pair information; SUTTA instead assembles the contigs by concurrently optimizing mate-pairs constraints and sequence quality. Another source for the different behavior could be found in the error correction technique: SOAPdenovo

uses the k-mer analysis to correct the reads but, since this process is not error-free, it might be introducing additional errors in the set of reads. Velvet's contigs instead are similar in size to SUTTA's but the coverage achieved with the correct contigs is only 56.9%.

Figure 7.6 shows the dotplot alignments of the contigs generated by the four assemblers using the MUMmer [53] software. All the contigs find a proper alignment to the reference genome, however notice that MUMmer generates the best possible alignment for each contig (even if the alignment similarity is $\leq 95\%$). So in this case, when all the contigs have already a fairly high quality, this kind of alignment plots becomes less informative.

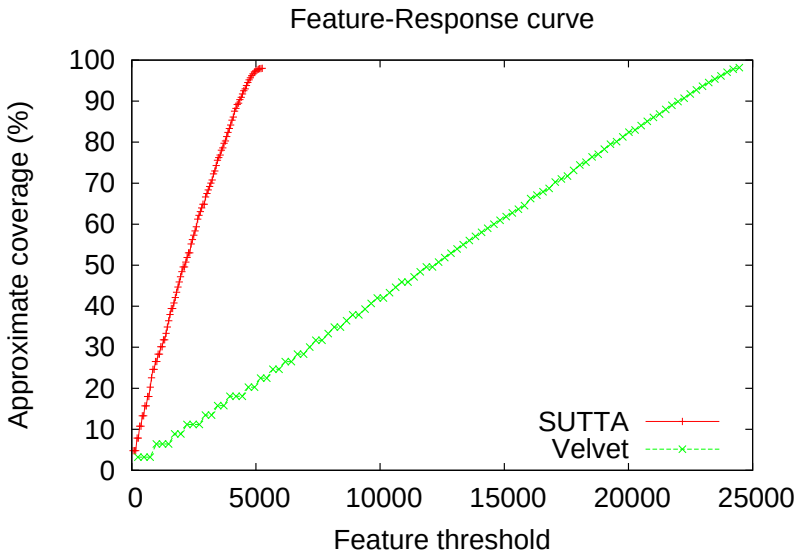


Figure 7.7: Feature-Response curve comparison for SUTTA and Velvet on the 100X *E.coli* data set.

More explanatory information can be gleaned from the Feature-Response

curve analysis presented in figure 7.7. SUTTA clearly outperforms Velvet assembly in quality. These results are in accordance with the coverage analysis presented in table 7.6. We were unable to compute the Feature-Response curve for the other two assemblers, SOAPdenovo and ABySS, because their output could not be converted into AMOS format. However, based on the previous coverage in table 7.6, it is fair to presume that the results would not have significantly changed.

Conclusion

Sequence assembly accuracy has become particularly important in: (1) genome-wide association studies, (2) detecting new polymorphisms, and (3) finding rare and de novo mutations. New sequencing technologies have reduced cost and increased the throughput. However, they have sacrificed read length and accuracy by allowing more single nucleotide (base-calling) and indel (e.g., due to homopolymer) errors. Overcoming these difficulties without paying for high computational cost requires (a) better algorithmic frameworks (not greedy), (b) being able to adapt to new and combined hybrid technologies (allowing significantly larger coverages and auxiliary long rage information) and (c) prudent experimental designs.

This dissertation has presented a novel assembly algorithm, SUTTA, that has been designed to satisfy these goals as it exploits many new algorithmic ideas. Challenging the popular intuition, SUTTA enables "fast" global optimization of the WGSA problem by taming the complexity using the branch-and-bound method. The resulting assembly paradigm gives the first priority to accuracy; secondly, it employs judicious experiment design concurrently (using long range information) to validate the results; and finally, it achieves computational ef-

ficiency by adaptive allocation of the resources. Because of the generality of the proposed approach, SUTTA has the potential to adapt to future sequencing technologies without major changes to its infrastructure: technology-dependent features can be encapsulated into the *lookahead* procedure and well-chosen *score functions*. Note that SUTTA also uses heuristics and has certain assumptions regarding consistent coverage levels, well behaved mates, reducing memory requirements (keeping the queue size small), resolving dead-ends and bubbles, etc. However these tools are plugged into a more general and flexible branch-and-bound (B&B) framework (similar to the one successfully applied to TSP and SAT problems) used to solve the full sequence assembly problem.

Nonetheless, SUTTA is still in its embryonic stage and has yet to achieve its full potential. In particular, it remains to be practically demonstrated that SUTTA can assemble a human haplotypic genome of any individual using single-molecule (optical) maps of moderate coverage (single molecules from only a handful of cells) rapidly and cheaply – a goal that has remained elusive to genomics research thus far. To achieve this goal we are planning several extensions of the SUTTA framework: (1) additional targeted error correction and local optimization routines to improve the quality results for any specific short reads technology, (2) software re-engineering to scale to large genomes, and (3) finally integration of long-range map data (e.g., optical maps) to concurrently validate and assemble in a *dovetailing* fashion.

A recent article entitled "Revolution Postponed" in Scientific American [31] commented "The Human Genome Project has failed so far to produce the med-

ical miracles that scientists promised. Biologists are now divided over what, if anything, went wrong...". And yet, excitement over the rapid improvements in biochemistry (pyrosequencing, sequencing by synthesis, etc.) and sensing (zeroth-order waveguides, nanopores, etc.) has now pervaded the field, as newer and newer sequencing platforms have started mass-migration into laboratories. Thus, biologists stand at a cross-road, pondering over the question of how to tackle the challenges of large-scale genomics with the high-throughput next-gen sequencing platforms. One may ask: Do biologists possess correct reference sequence(s)? If not, how should they be improved? How important are haplotypes? Does it suffice to impute the haplotype-phasing from population? How much information is captured by the known genetic variants (e.g., SNPs and CNVs)? How does one find the de novo mutations and their effects on various complex traits? Can exon-sequencing be sufficiently informative?

Central to all these challenges is the second problem we have addressed in this thesis: namely, how correct are the existing sequence assemblers for making reference genome sequences? How good are the assumptions they are built upon? Unfortunately, we have discovered that the quality and performance of the existing assemblers varies dramatically. The standard metrics used to compare assemblers for the last ten years emphasized contig size while poorly capturing the assembly quality. For these reasons, we have developed a new metric, Feature-Response curve (FRC), to compare assemblers and assemblies that more satisfactorily captures the trade-off between contigs' size and quality. This metric shares many similarities with the receiver operating characteristic curve widely

used in medicine, radiology, machine learning and other areas for many decades. Moreover the FRC does not require any reference sequence (except an estimate of the genome size) to be used for validation, thus making it a very useful tool in *de novo* sequencing projects. Furthermore the inspection of separate FRCs for each feature type enables to scrutinize the relative strengths and weaknesses of each assembler.

Note that although in this thesis the aim has been to test and compare as many *de novo* sequence assemblers and covering known assembly paradigms as exhaustively as possible, in a fast evolving field such as this, this goal has not been completely met — some of the assemblers listed in table 3.1 were only released very recently, and not early enough to be included in the statistical comparison. It is hoped that the community of researchers interested in sequence assembly algorithms will close this gap with the FRC software, which is now available as part of the AMOS open-source consortium.

The third and final topic of this dissertation has been the introduction of a new Base-Caller, TotalReCaller, that yields better quality sequence reads, SNP-calls, variant detection, etc., as well as an alignment at the best possible location in the reference genome. In the same spirit as SUTTA, TotalReCaller is based on global optimization of a Bayesian score function (combining a linear error model for the raw intensity and Burrows-Wheeler transform based alignment) using an efficient branch-and-bound approach. TotalReCaller and SUTTA have been integrated into a full pipeline (from base-calling to assembly) that achieves competitive performance compared to the state-of-the-art assemblers for next-

generation technology.

Returning to our earlier concerns, and the quandaries of computational biologists, biotechnologists also need to reflect on related issues: Given the unavoidable computational complexity burden of assembly, how do we best design the sequencing platforms? There are many parameters that characterize a sequencing platform: read-length, base-call-errors, homopolymer-length, throughput, cost, latency, augmentation with mate-pairs, scaffolding, long-range information, etc. And not all can be addressed equally well in all the platforms. The history of sequencing technology has been a random-walk in this complex design-space. To speed up the classical Sanger sequencing, while increasing throughput, ten years ago, engineers focused on effects of increased electric field, Joule-heating, calibrating with smaller amount of materials, etc., as was done with multi-lane capillary-sequencers. The next improvement came from pyro-sequencing or ligation-based sequencing carried out synchronously using small number of clonal copies of DNA fragments, following bridge or emulsion PCR. However, since it was difficult to keep the reactions synchronized (with confounding leading, lagging and fading reactions), the reads shortened and became more error-prone — somewhat, compensated by higher depth of coverage. To avoid the synchronization problem, it was necessary to go to a single (genomic DNA) molecule technology, in which either the molecules are kept fixed and sensing mobile (optical mapping, optical sequencing, Heliscope sequencer, AFM-based mapping, SMASH, etc.) or the molecules mobile and sensing fixed (PacificBioscienes, nanopores, etc.). However, the single-molecule technologies face the problem of mismatched speed of

the mobile molecules or mobile sensors and the resulting resolution. For the time being, low-resolution map technology (e.g., optical restriction maps) for very long immobile molecules points to the most profitable avenue. If one were to speculate what the next step should be, as was done by Schwartz and Waterman [90], one may "project that over the next two years, reference genomes will be constructed using new algorithms combining long-range physical maps with voluminous Gen-2/3 datasets. In this regard, the Optical Mapping System constructs genome-wide ordered restriction maps from individual ($\sim$ 500 kbp) genomic DNA molecules" [9, 65, 4, 5, 6, 3, 74]. Among all the assemblers examined in this thesis, SUTTA appears to be best suited for such a strategy.

Appendix A

This appendix contains the pseudo-code for the subroutines used in Algorithm 3. Note that for the sake of a clear exposition we give only a high level description of the algorithms while most of the implementation details and optimizations are omitted. The following subroutines are also used in the pseudo-code:

$isSuffix(x, y)$**:** returns true if the suffix of x overlaps with y

$Consistent(x, y)$**:** checks the consistency property between reads x and y according to the definition 3 in chapter 2.

$checkTransitivity(x, y, z)$**:** checks if there is a transitivity relation between reads x, y and z.

$Lookahead_{de}(x, y, W_{de})$**:** returns the maximum depth of the lookahead tree created starting from node/read y with ancestor x. The local tree is constructed using the Branch-and-Bound strategy similarly to the "Node expansion" routine described in Algorithm 3 in chapter 4, however to avoid recursion, dead-end, bubble and mate-pair pruning are not performed during the construction.

$Converge(x, y)$**:** return *true* if the path constructed starting from nodes x and y converges after at most W_{bb} steps.

$Lookahead_{bb}(x, y, W_{bb})$**:** returns the depth of the tree constructed starting from node y with ancestor x.

$Lookahead_{mp}(x, y, W_{mp})$**:** computes the local tree starting from node y with ancestor x of maximum depth W_{mp} and returns the score of the best path (highest score). Path's score is computed according to the mate-pair score defined in equations (4.5) and (4.6).

Algorithm 6: Extensions

Input: Read/Node r to extend
Output: Set of reads/nodes $\mathcal{E}$ that can be extended

```
1  R := getReads(r);                    /* Set of reads overlapping with r */
2  for (j=1 to |R|) do
3      if (¬isUsed(r_j)) then
4          if (isRoot(r)) then
5              if (leftTree ∧ ¬isSuffix(r_j, r) ∥ (rightTree ∧ isSuffix(r_j, r) then
6                  E := E ∪ {r_j};
7              end
8              if (leftTree ∧ isSuffix(r_j, r) ∥ (rightTree ∧ ¬isSuffix(r_j, r) then
9                  E := E ∪ {r_j};
10             end
11         else
                                                /* r is not the root node */
12             r_i := ParentNode(r_j);
13             if (Consistent(r_i, r_j)) then
14                 E := E ∪ {r_j};          /* r_j is a possible extension */
15             end
16         end
17     end
18 end
19 return E;
```

Algorithm 7: Transitivity

Input: Set of reads/nodes $\mathcal{E}$, Read/Node r to extend
Output: Set of reads/nodes $\mathcal{E}^{(1)}$ after removing transitivity between siblings

```
for (i=1 to |E|) do
    for (j=i+1 to |E|) do
        if (checkTransitivity(r, r_i, r_j)) then
            E := E \ {r_j};
        end
    end
end
E^(1) := E
return E^(1);
```

Algorithm 8: DeadEnds

Input: Set of reads/nodes $\mathcal{E}^{(1)}$, Read/Node r to extend, max depth W_{de}
Output: Set of reads/nodes $\mathcal{E}^{(2)}$ after removing dead-ends

```
for (j=1 to |E^(1)|) do
    d := Lookahead_de(r, r_j, W_de);
    if (d ≤ W_de) then
        E^(1) := E^(1) \ {r_j};
    end
end
E^(2) := E^(1)
return E^(2);
```

Algorithm 9: Bubbles

Input: Set of reads/nodes $\mathcal{E}^{(2)}$, Read/Node r to extend, max depth W_{bb}
Output: Set of reads/nodes $\mathcal{E}^{(4)}$ after removing bubbles

```
for (i=1 to |E^(2)|) do
    for (j=i+1 to |E|^(2)) do
        if (Converge(r_i, r_j, W_bb)) then
            d_1 := Lookahead_bb(r, r_i, W_bb);
            d_2 := Lookahead_bb(r, r_j, W_bb);
            if (d_1 < d_2) then
                E^(2) := E^(2) \ {r_i};
            else
                E^(2) := E^(2) \ {r_j};
            end
        end
    end
end
E^(3) := E^(2)
return E^(3);
```

Algorithm 10: MatePairs

Input: Set of reads/nodes $\mathcal{E}^{(3)}$, Read/Node r to extend, max depth W_{mp}
Output: Set of reads/nodes $\mathcal{E}^{(4)}$ after removing low mate-pair scoring extensions

```
1  for (i=1 to |E^(3)|) do
2  |  for (j=i+1 to |E|^(3)) do
3  |  |  s_1 := Lookahead_mp(r, r_i, W_mp);
4  |  |  s_2 := Lookahead_mp(r, r_j, W_mp);
5  |  |  if (s_1 > s_2) then
6  |  |  |  E^(2) := E^(2) \ {r_i};
7  |  |  end
8  |  |  if (s_1 < s_2) then
9  |  |  |  E^(2) := E^(2) \ {r_j};
10 |  |  else
   |  |  |                                   /* Do nothing */
11 |  |  end
12 |  end
13 end
14 E^(4) := E^(3)
15 return E^(4);
```

Appendix B

The results in Appendix B are organized in two main sections:

Long Reads

1. No Mate-Pairs constraints
   (a) *Brucella suis*
   (b) *Staphylococcus epidermidis*
   (c) *Wolbachia Sp.*
   (d) *Chromosome Y*
2. With Mate-Pairs constraints
   (a) *Brucella suis*
   (b) *Staphylococcus epidermidis*
   (c) *Wolbachia sp.*
   (d) *Chromosome Y*

Short Reads

1. With Mate-Pairs constraints

   (a) *Escherichia coli*

For each genome we report the Feature-Response curves (FRC) cumulative over all the features, the FRCs for each feature type, and the dot plot alignments of the set of contigs generated by each assembler computed using the MUMmer package. For the short read E. coli data, only Velvet is used in the comparison since it is the only short read assembler whose output can be converted into an AMOS bank for validation. For the same reason Euler is excluded from the dot plots for long reads. The mis-assembly FRC is computed using the mis-assembly feature which, according to the *amosvalidate* description, is obtained by applying a feature combiner that collects all of the evidence for a mis-assembly and outputs regions with multiple mis-assembly features when present at the same region. Note that the mis-assembly feature is not used in computing the FRC curve. Finally note that when the number of features of a specific type is 0 for each contig in the set, the FRC reduces to a single point. Although not all the curves are equally informative, for the sake of completeness of exposition, we present all the FRCs.

Long Reads

No Mate-Pairs constraints

Brucella suis

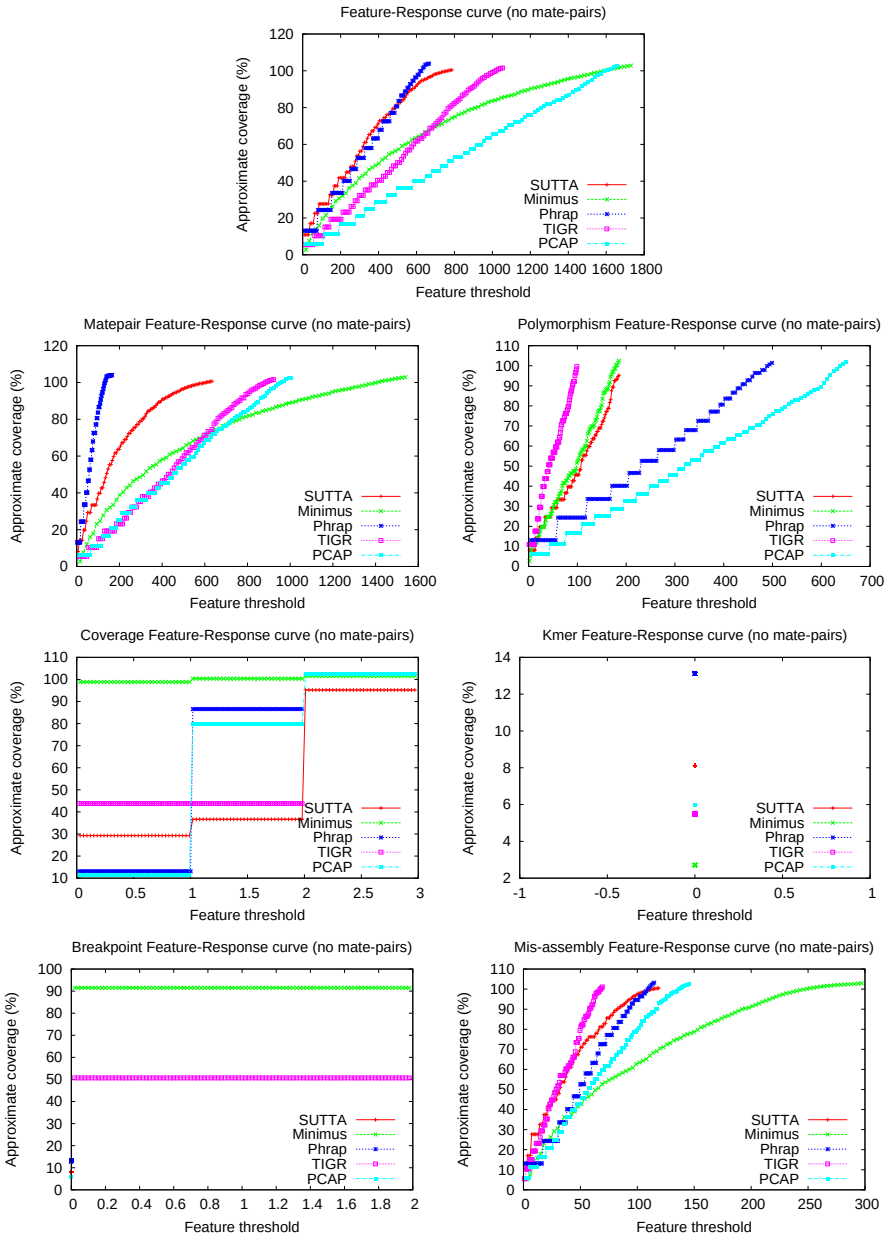
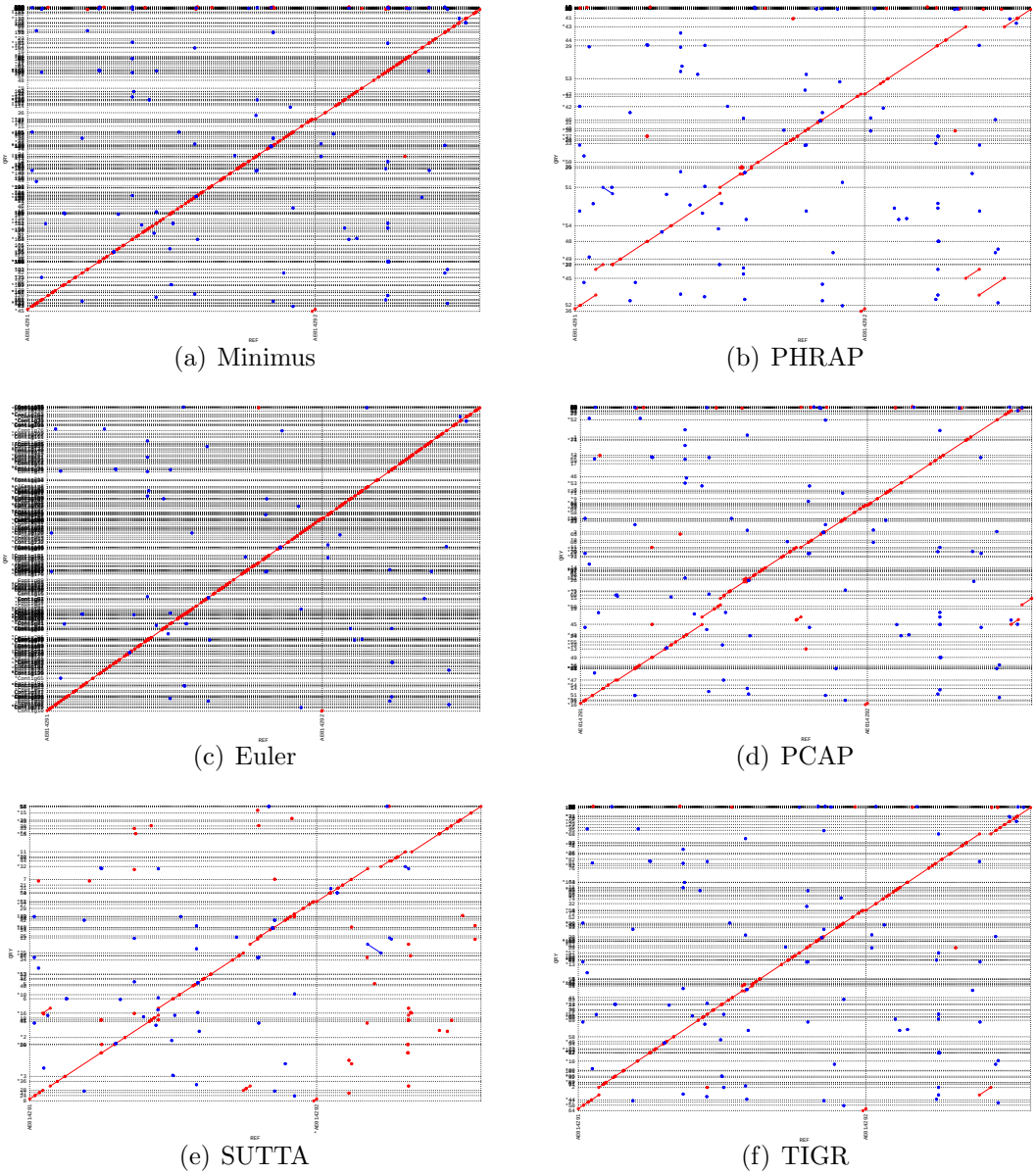


Figure 8: Feature-Response curves by feature type for *Brucella suis* without mate-pair constraints.



(a) Minimus (b) PHRAP

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 9: Dot plots for *Brucella suis* (no mate-pairs). Assemblies produced by Minimus, PHRAP, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Staphylococcus epidermidis

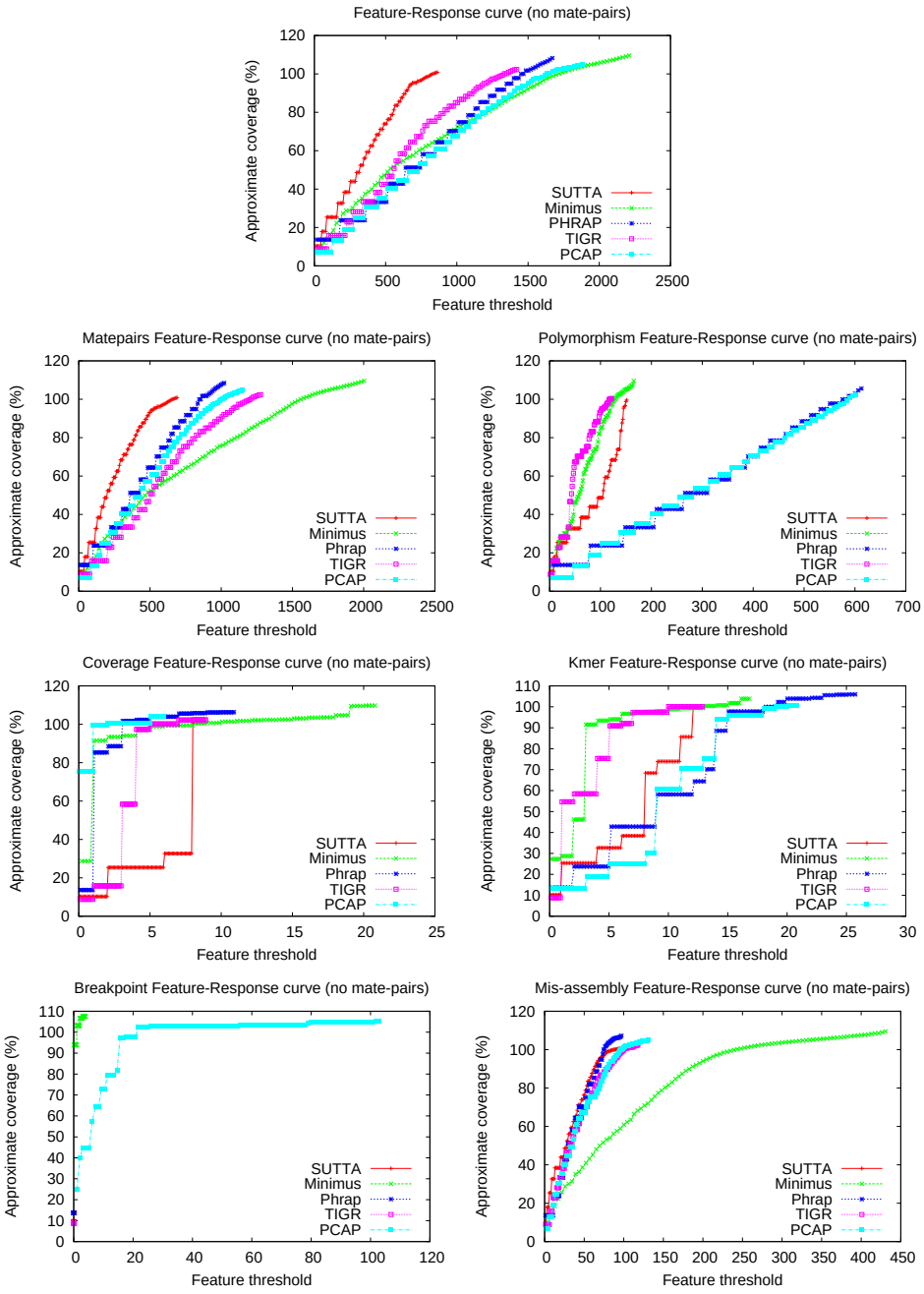
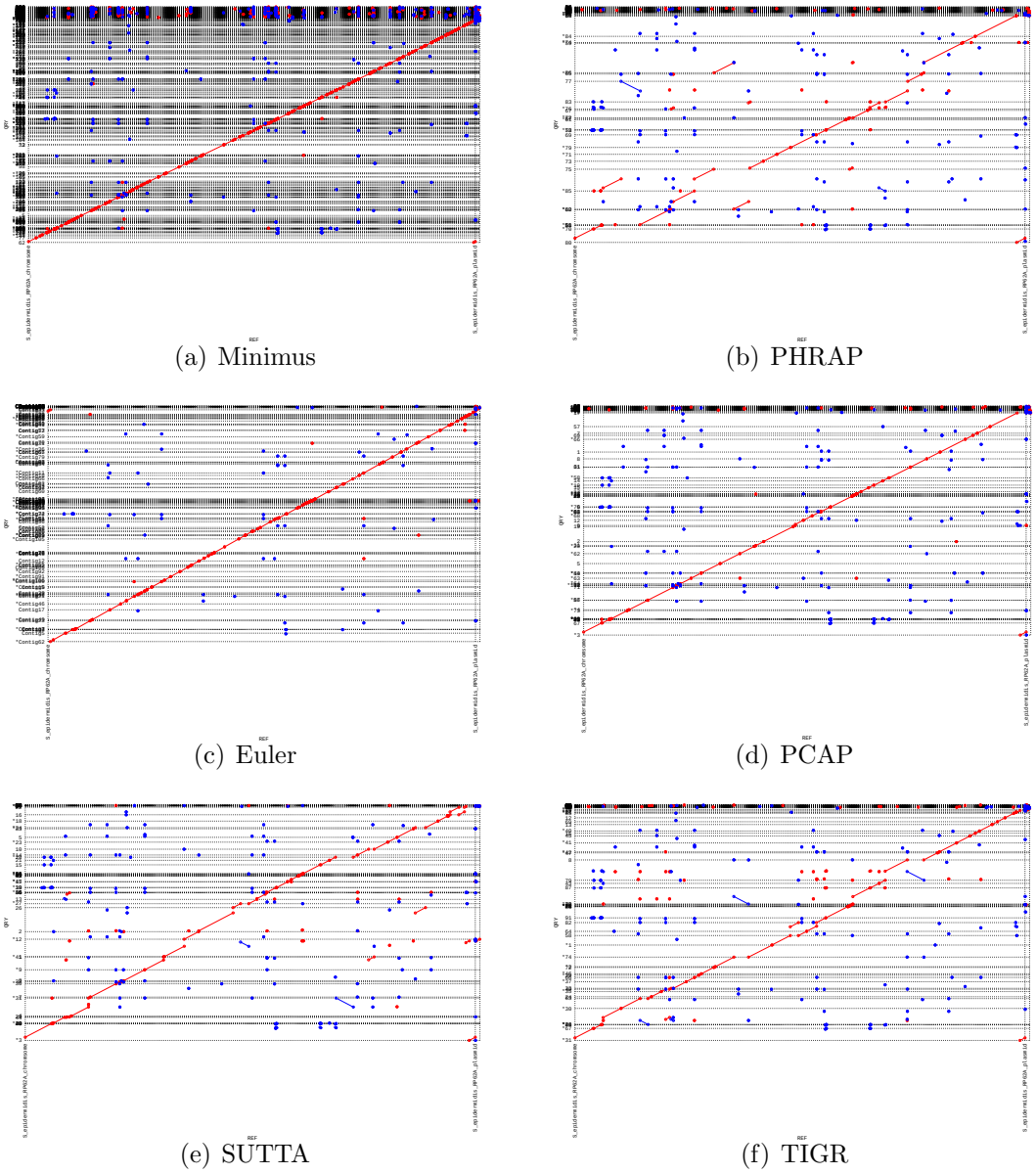


Figure 10: Feature-Response curves by feature type for *Staphylococcus epidermidis* without mate-pair constraints.



(a) Minimus (b) PHRAP

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 11: Dot plots for *Staphylococcus epidermidis* (no mate-pairs). Assemblies produced by Minimus, PHRAP, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Wolbachia sp.

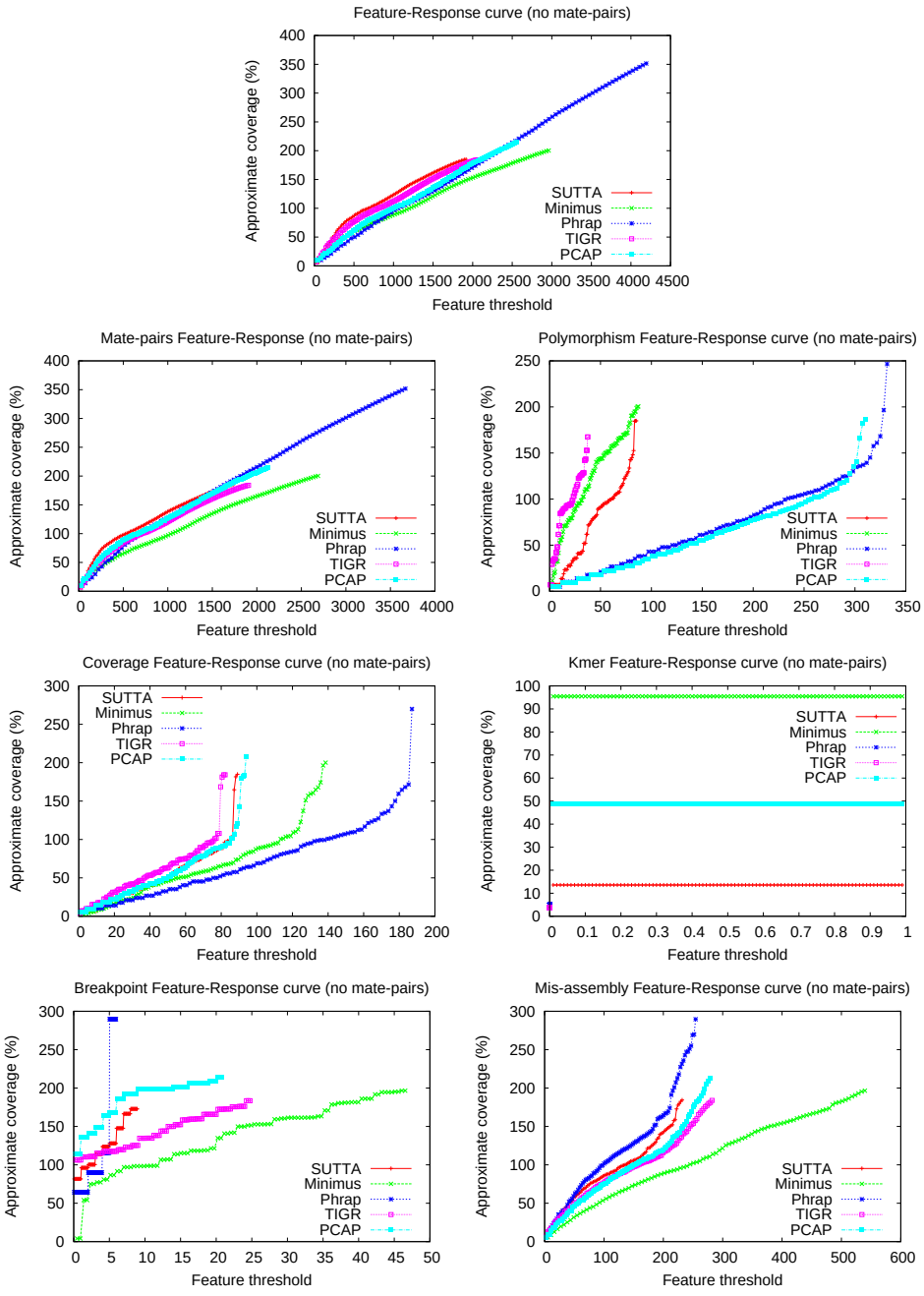
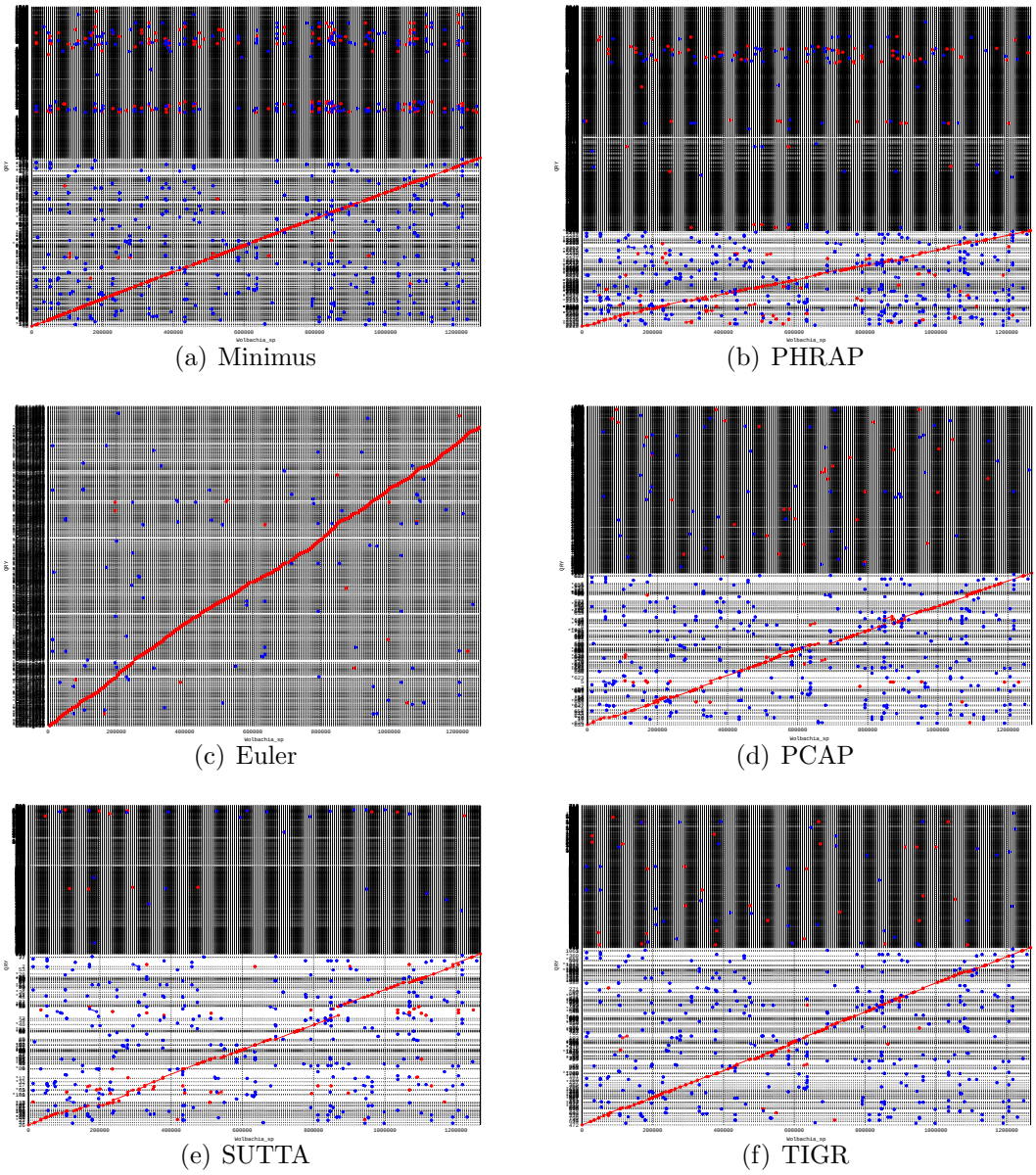


Figure 12: Feature-Response curves by feature type for *Wolbachia sp.* without mate-pair constraints.



(a) Minimus (b) PHRAP

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 13: Dot plots for *Wolbachia sp.* (no mate-pairs). Assemblies produced by Minimus, PHRAP, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Chromosome Y

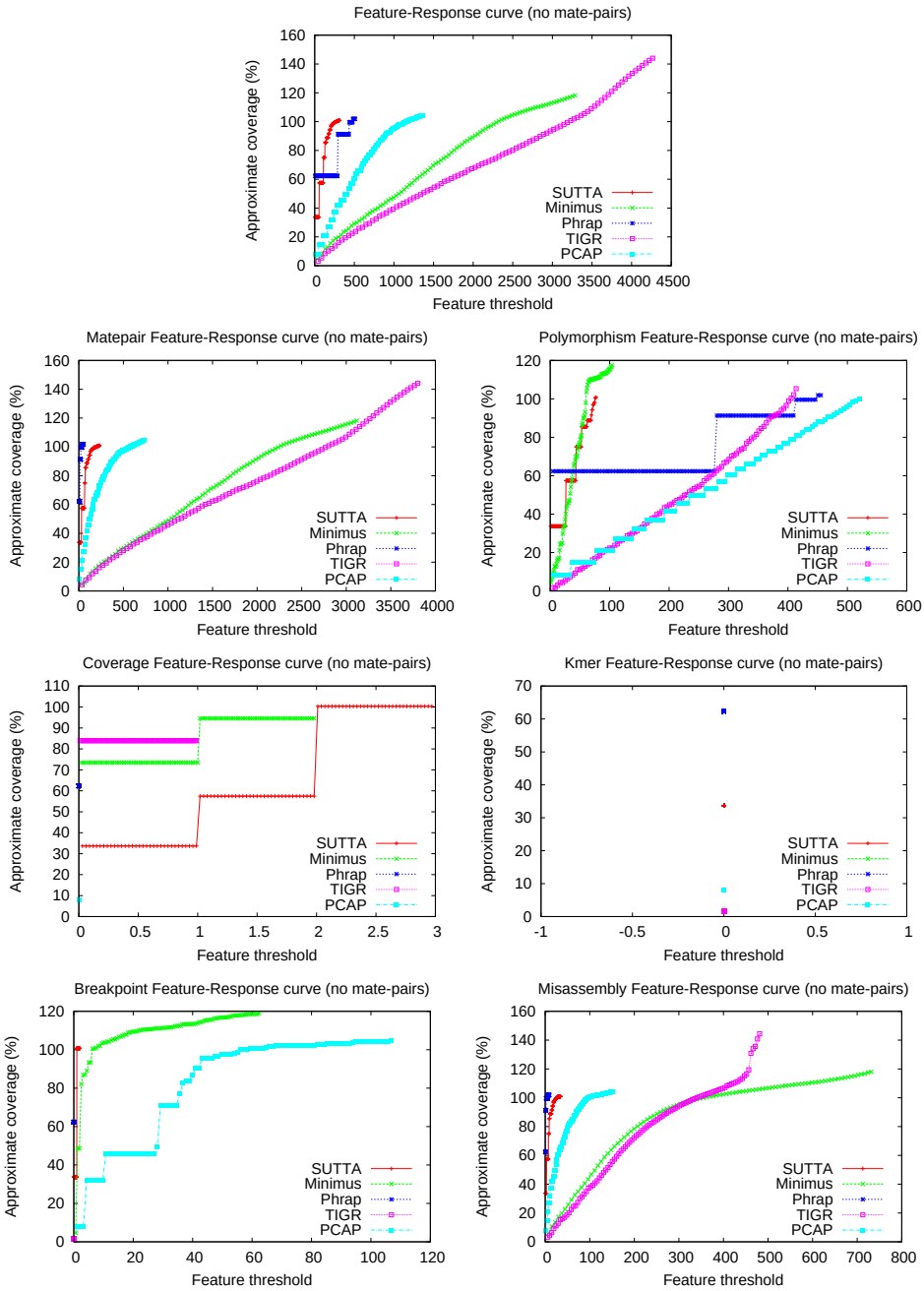
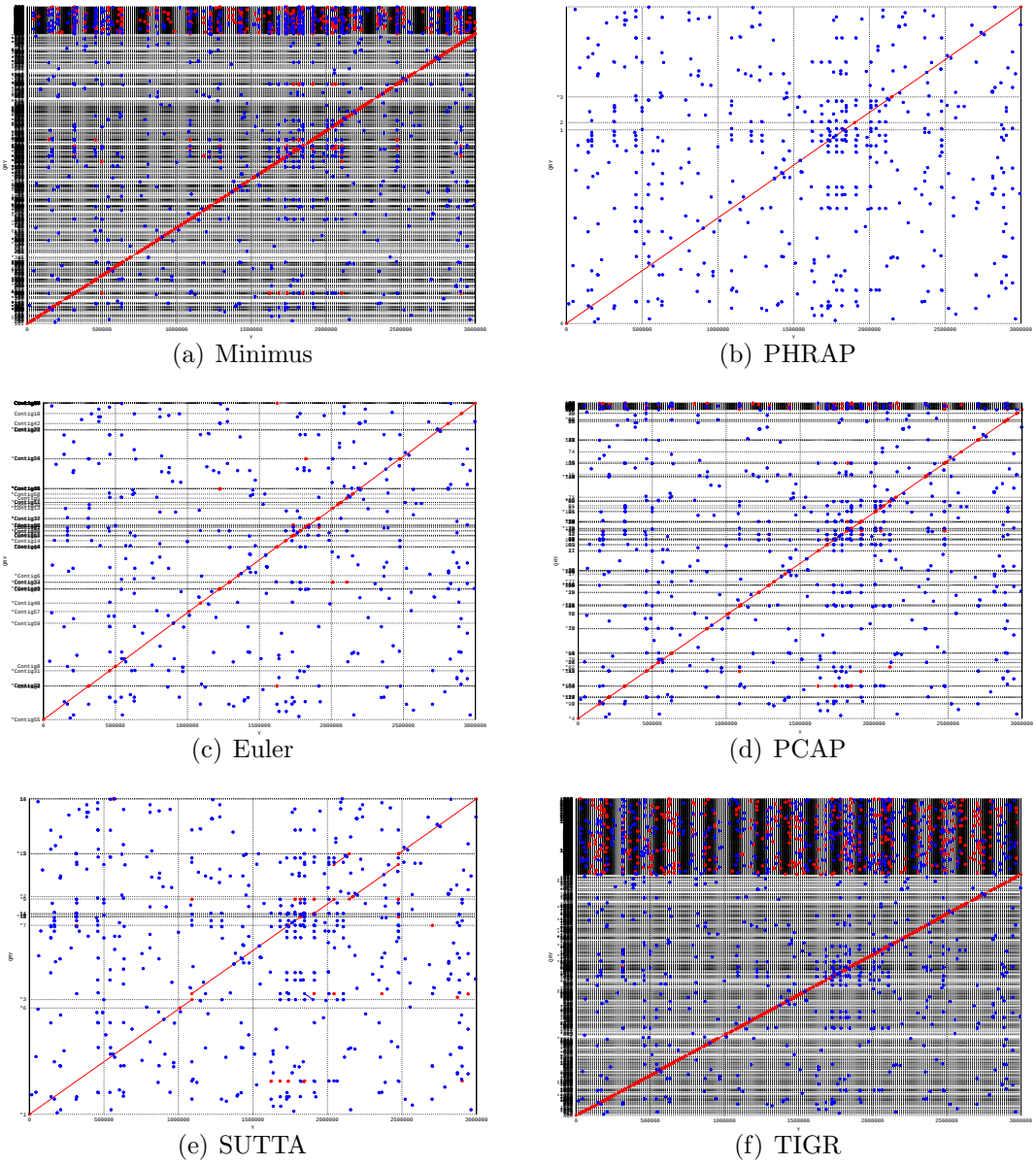


Figure 14: Feature-Response curves by feature type for *Chromosome Y* without mate-pair constraints.



(a) Minimus (b) PHRAP

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 15: Dot plots for *Chromosome Y* 3Mbs region (no mate-pairs). Assemblies produced by Minimus, PHRAP, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

With Mate-Pairs constraints

Brucella suis

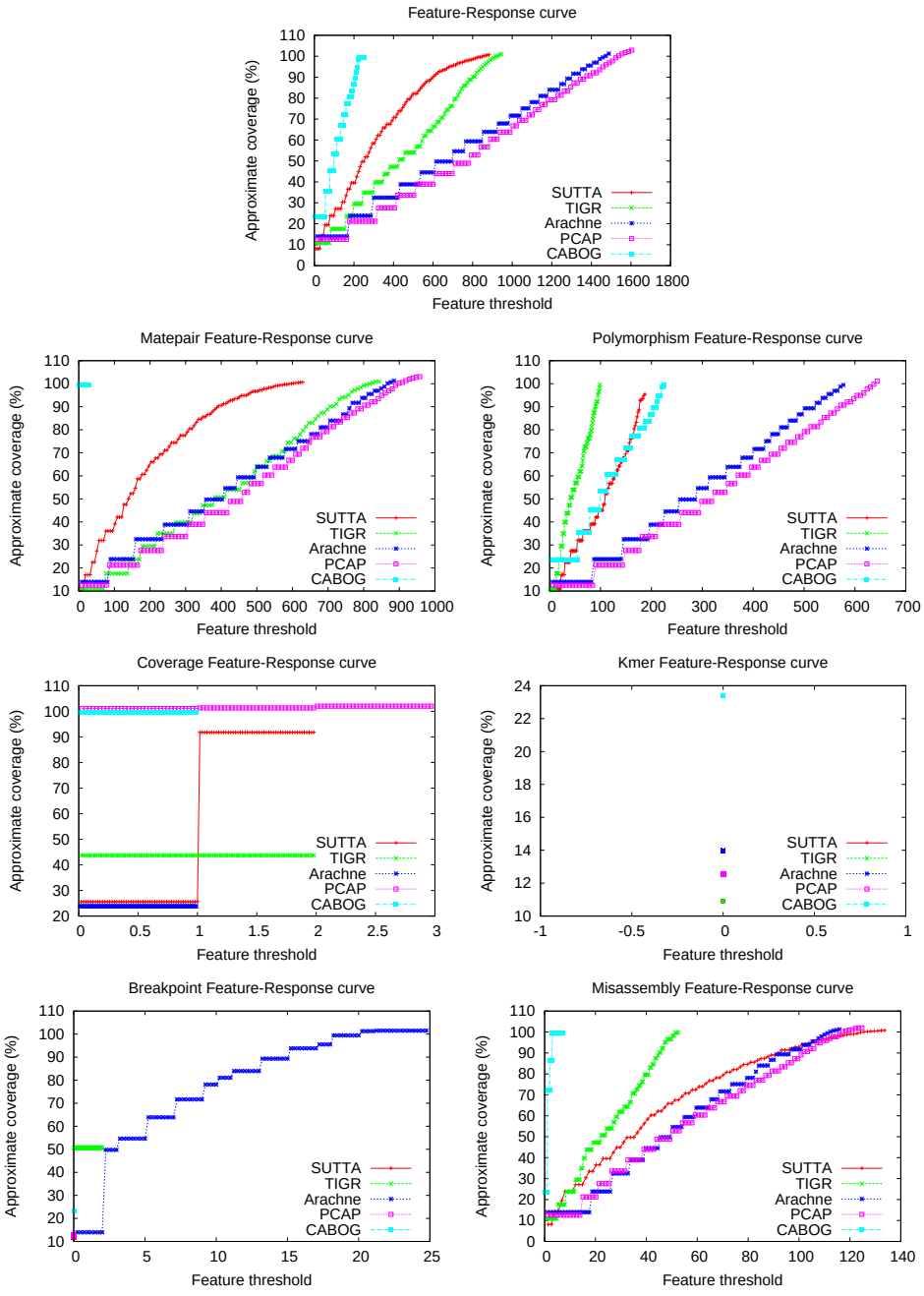
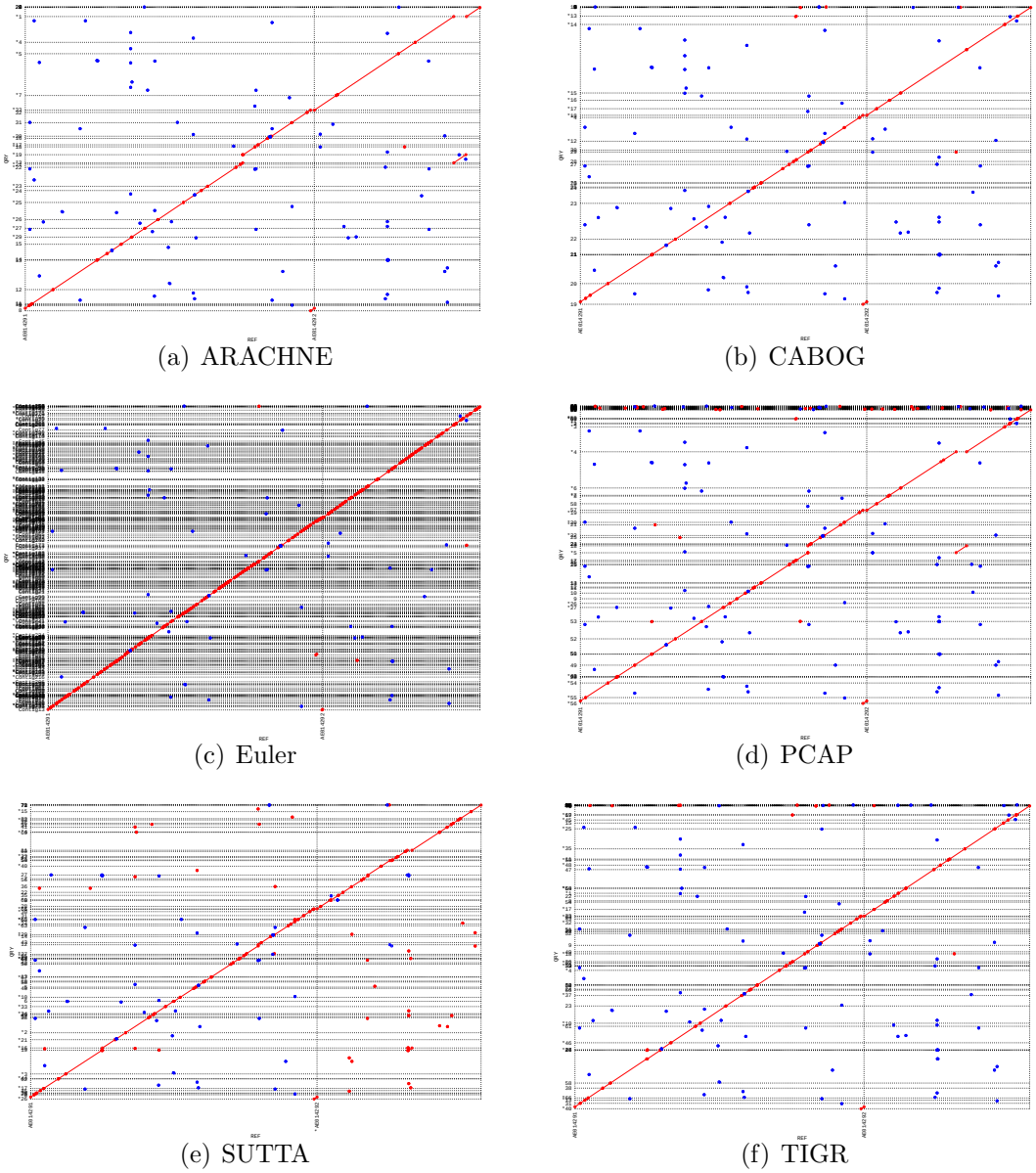


Figure 16: Feature-Response curves by feature type for *Brucella suis* with mate-pair constraints.



(a) ARACHNE (b) CABOG

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 17: Dot plots for *Brucella suis* (with mate-pairs). Assemblies produced by ARACHNE, CABOG, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Staphylococcus epidermidis

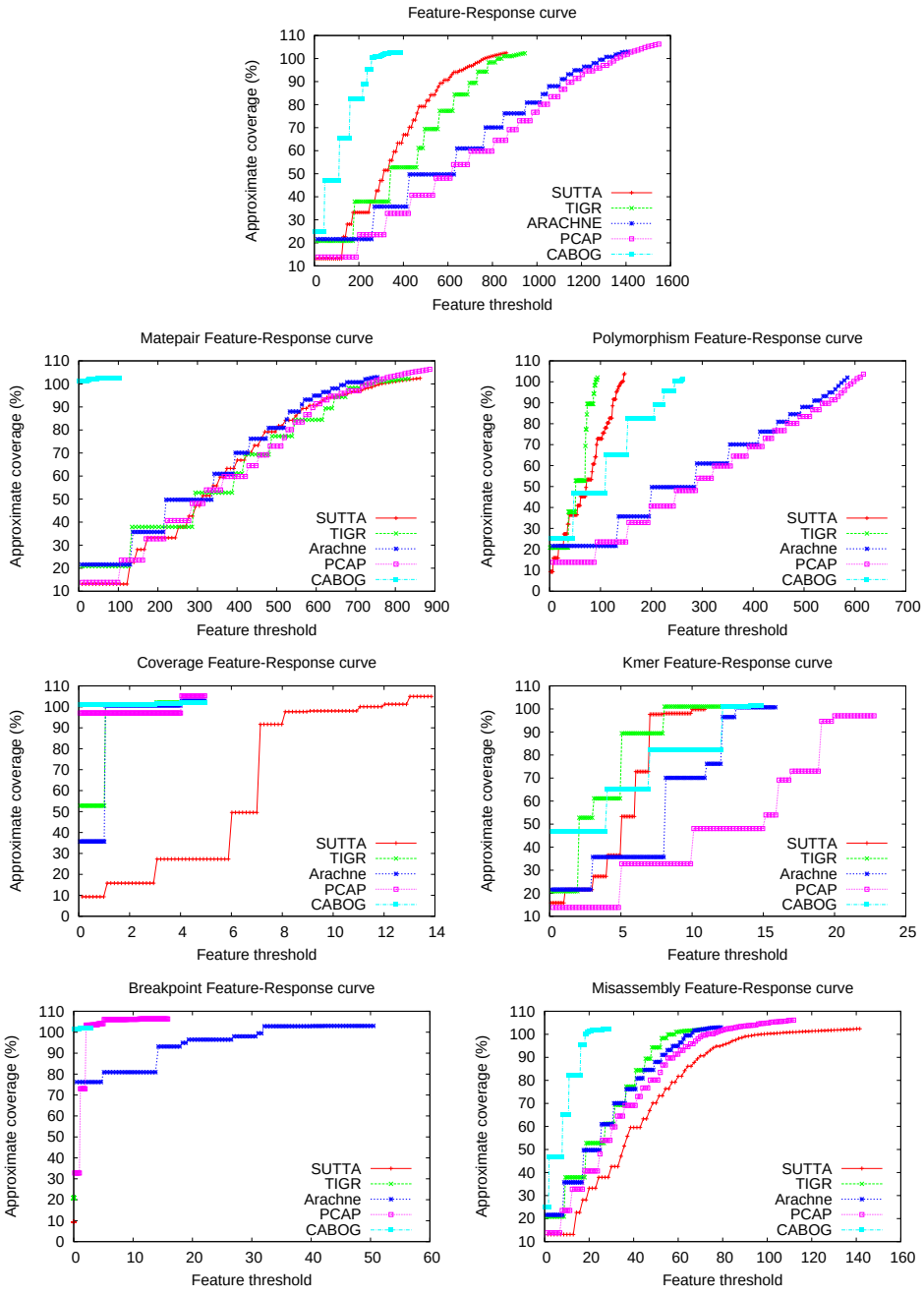
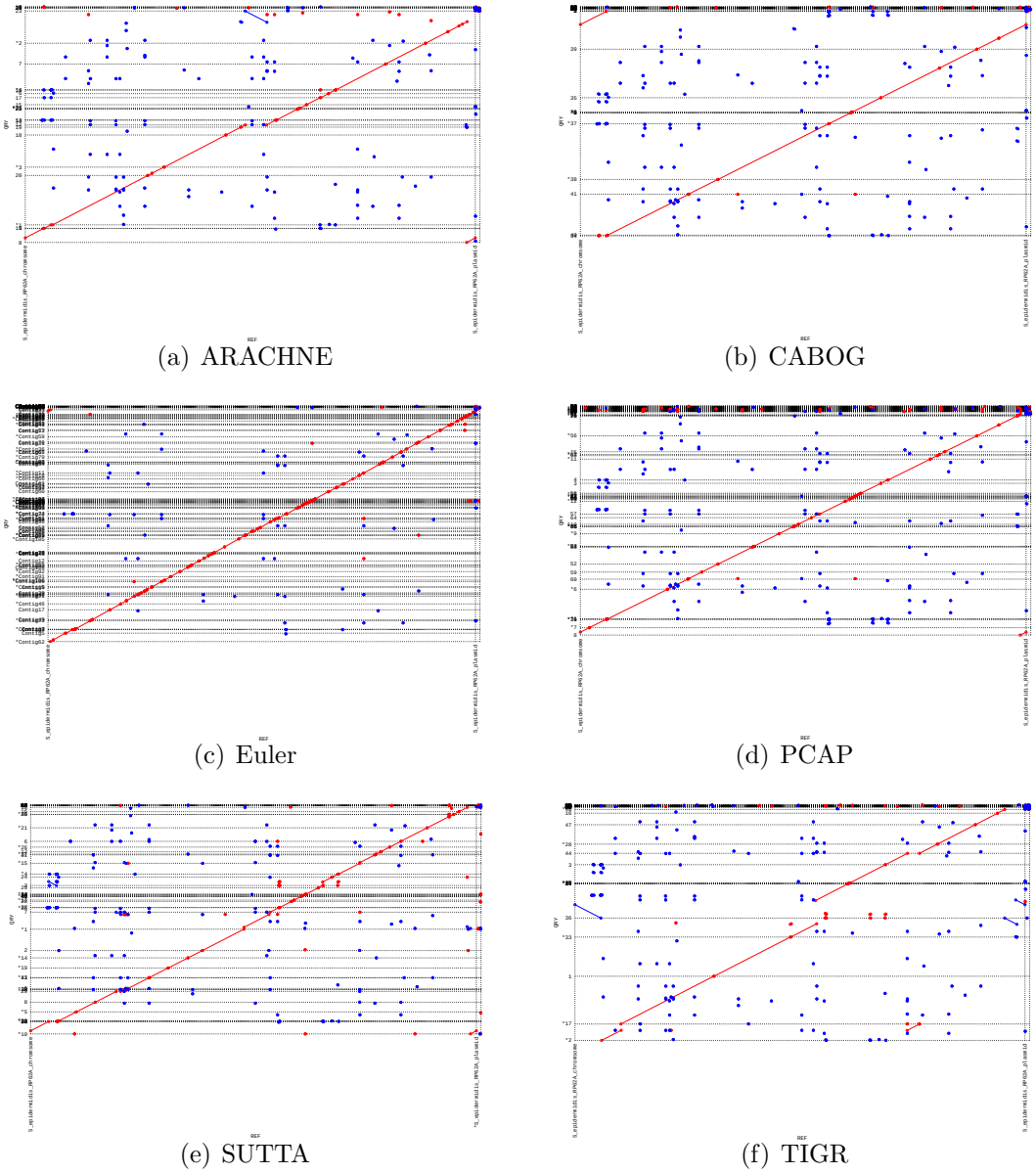


Figure 18: Feature-Response curves by feature type for *Staphylococcus epidermidis* with mate-pair constraints.



(a) ARACHNE (b) CABOG

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 19: Dot plots for *Staphylococcus epidermidis* (with mate-pairs). Assemblies produced by ARACHNE, CABOG, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Wolbachia sp.

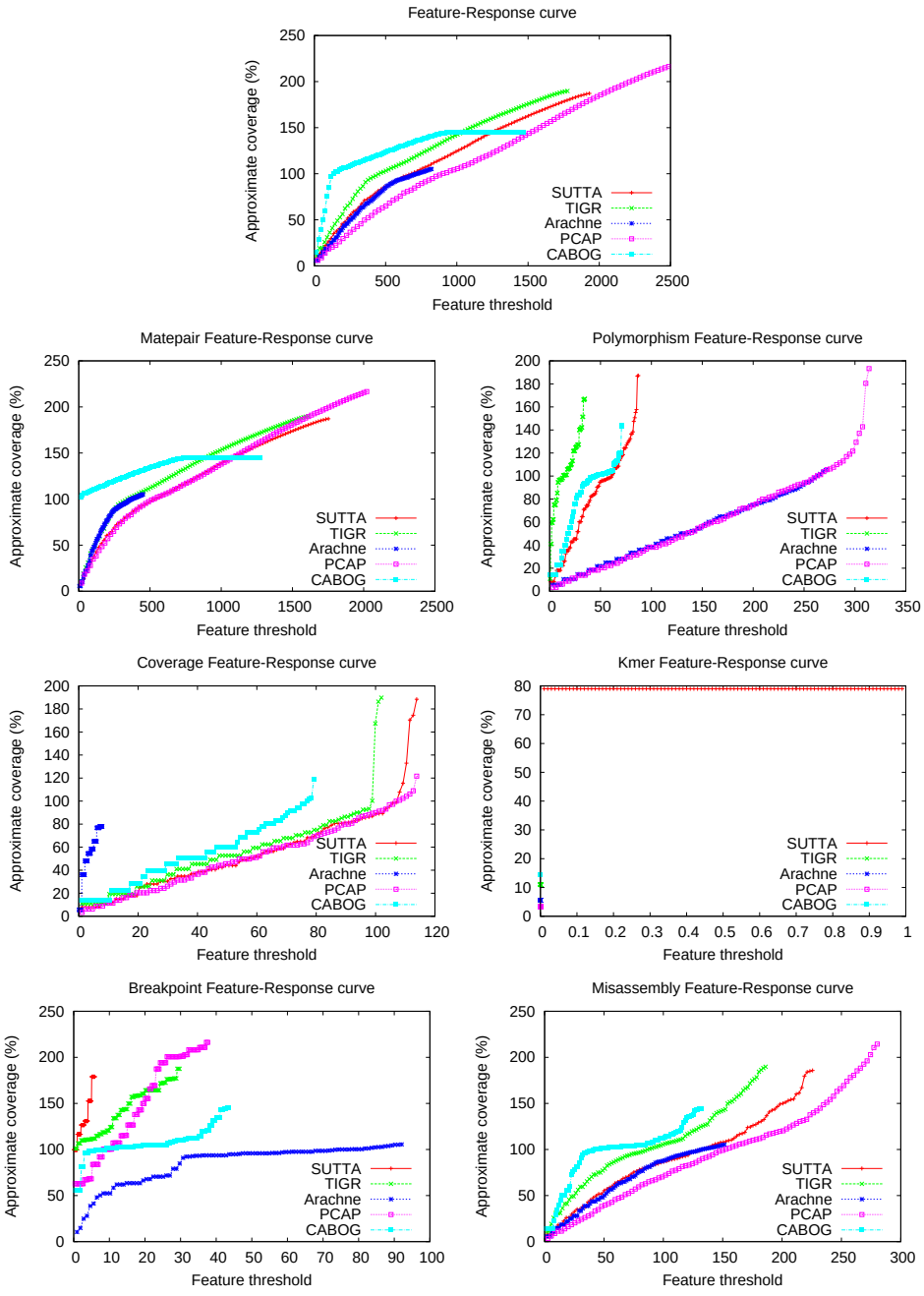
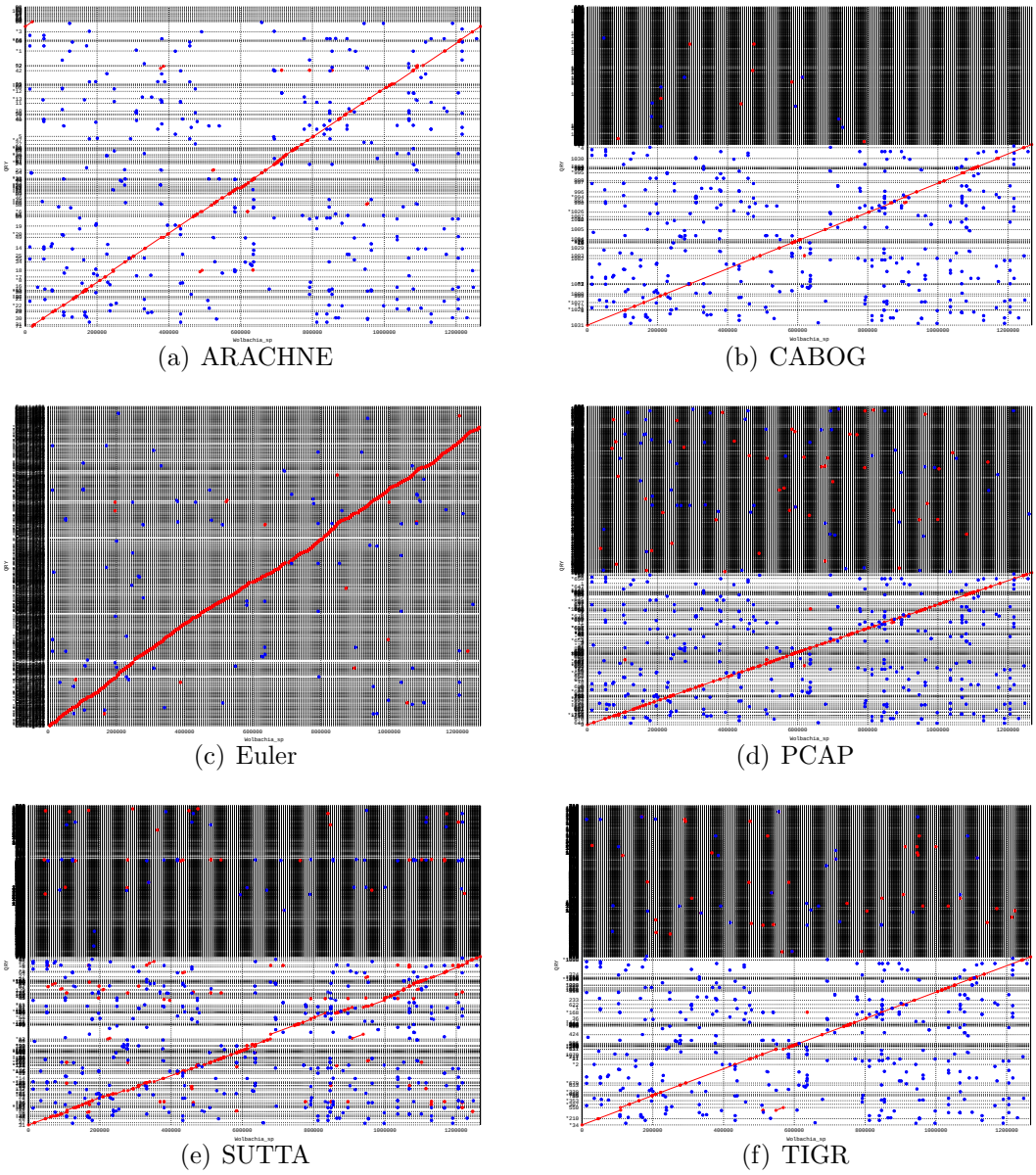


Figure 20: Feature-Response curves by feature type for *Wolbachia sp.* with mate-pair constraints.



(a) ARACHNE (b) CABOG

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 21: Dot plots for *Wolbachia sp* (with mate-pairs). Assemblies produced by ARACHNE, CABOG, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Chromosome Y

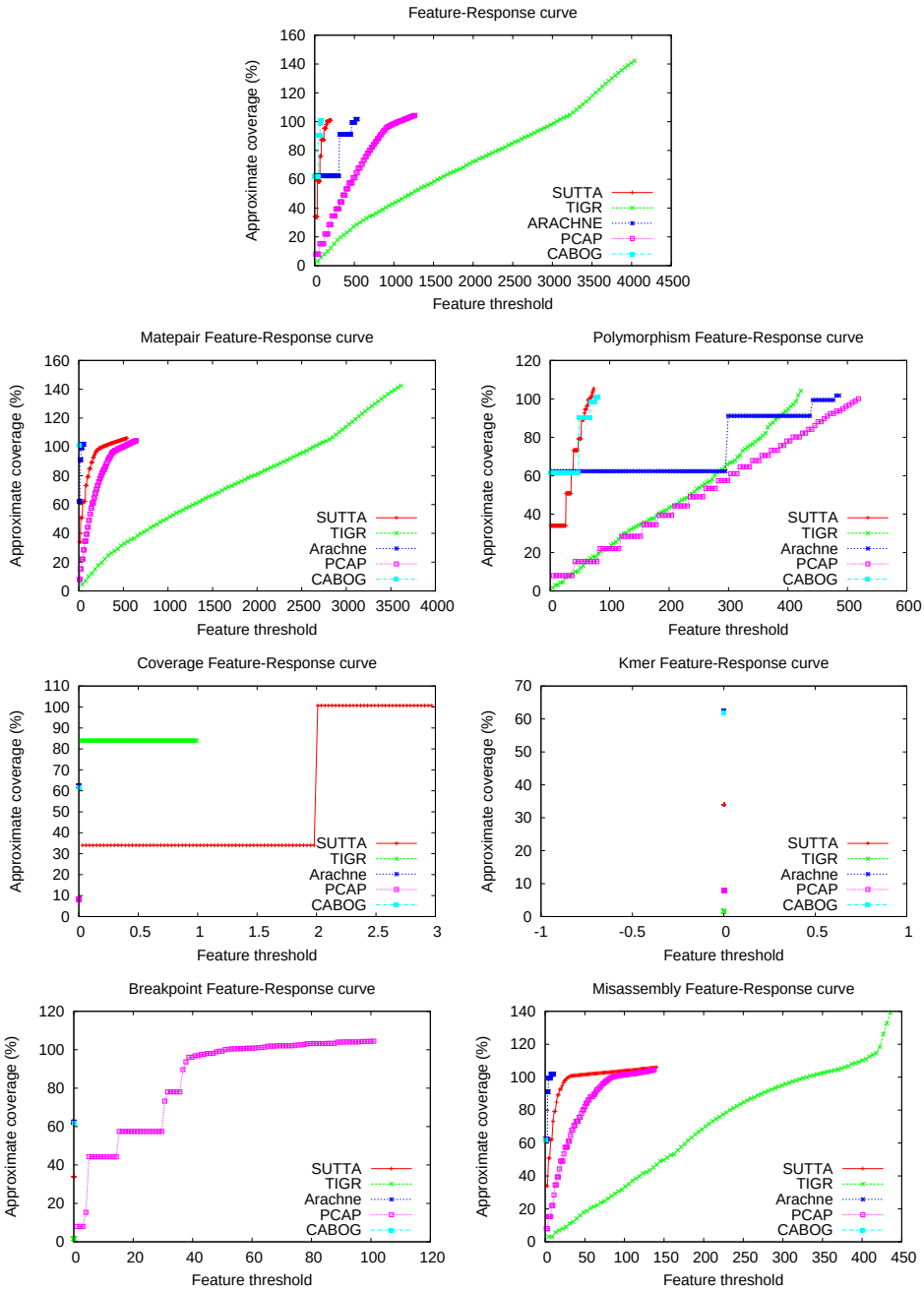
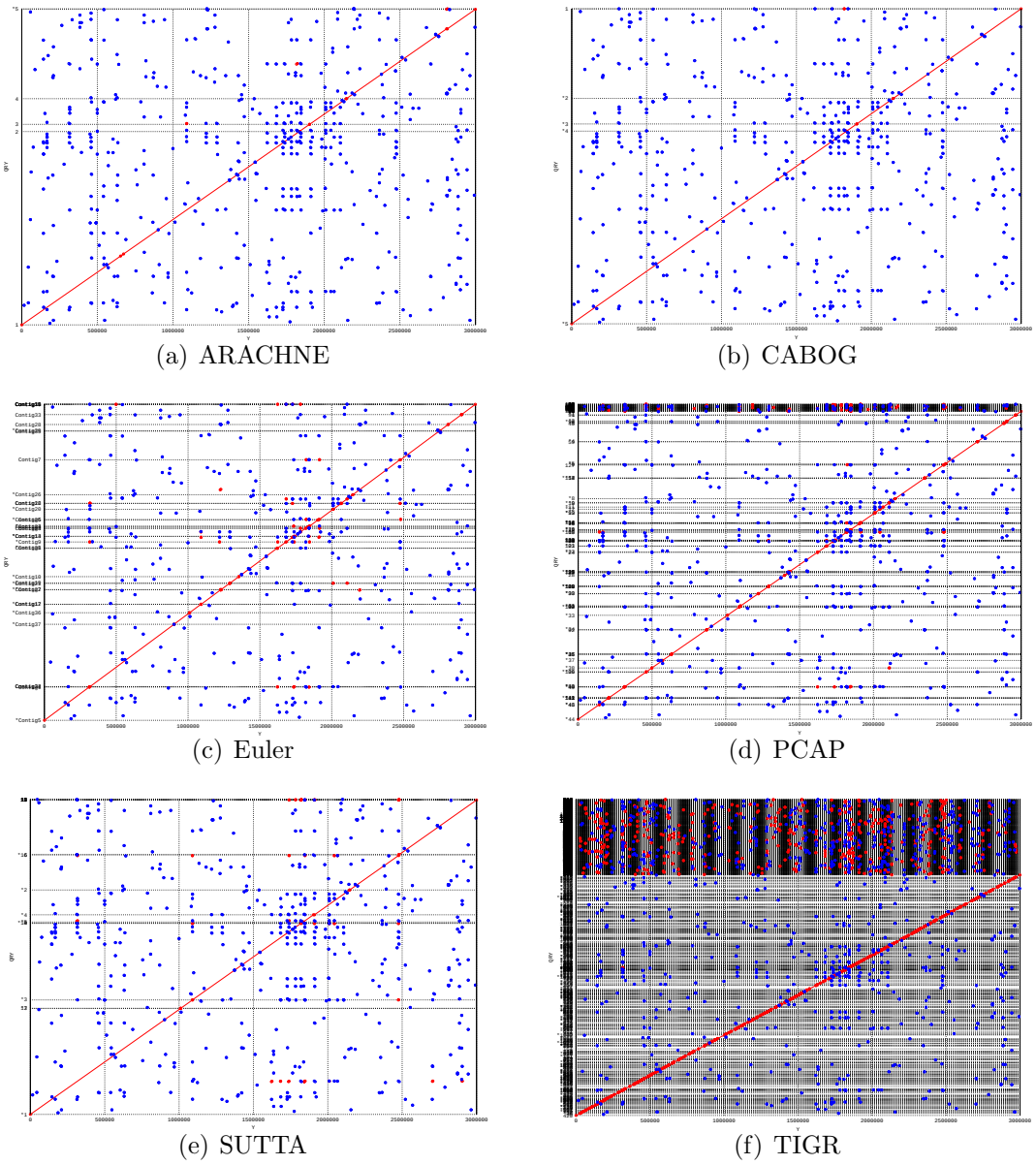


Figure 22: Feature-Response curves by feature type for *Chromosome Y* with mate-pair constraints.



(a) ARACHNE (b) CABOG

(c) Euler (d) PCAP

(e) SUTTA (f) TIGR

Figure 23: Dot plots for *Chromosome Y* 3Mbps region (with mate-pairs). Assemblies produced by ARACHNE, CABOG, Euler, PCAP, SUTTA and TIGR. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Short Reads

With Mate-Pairs constraints

Escherichia coli

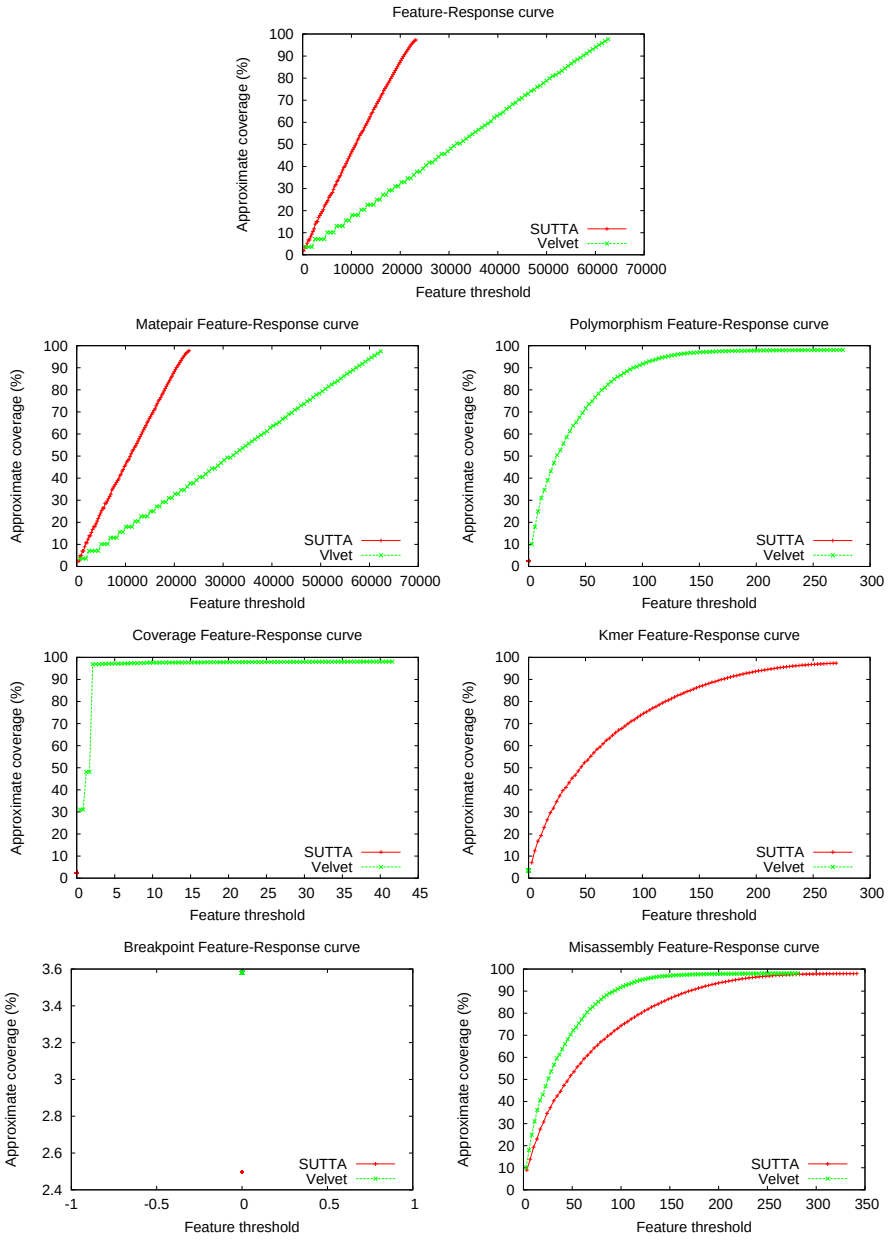
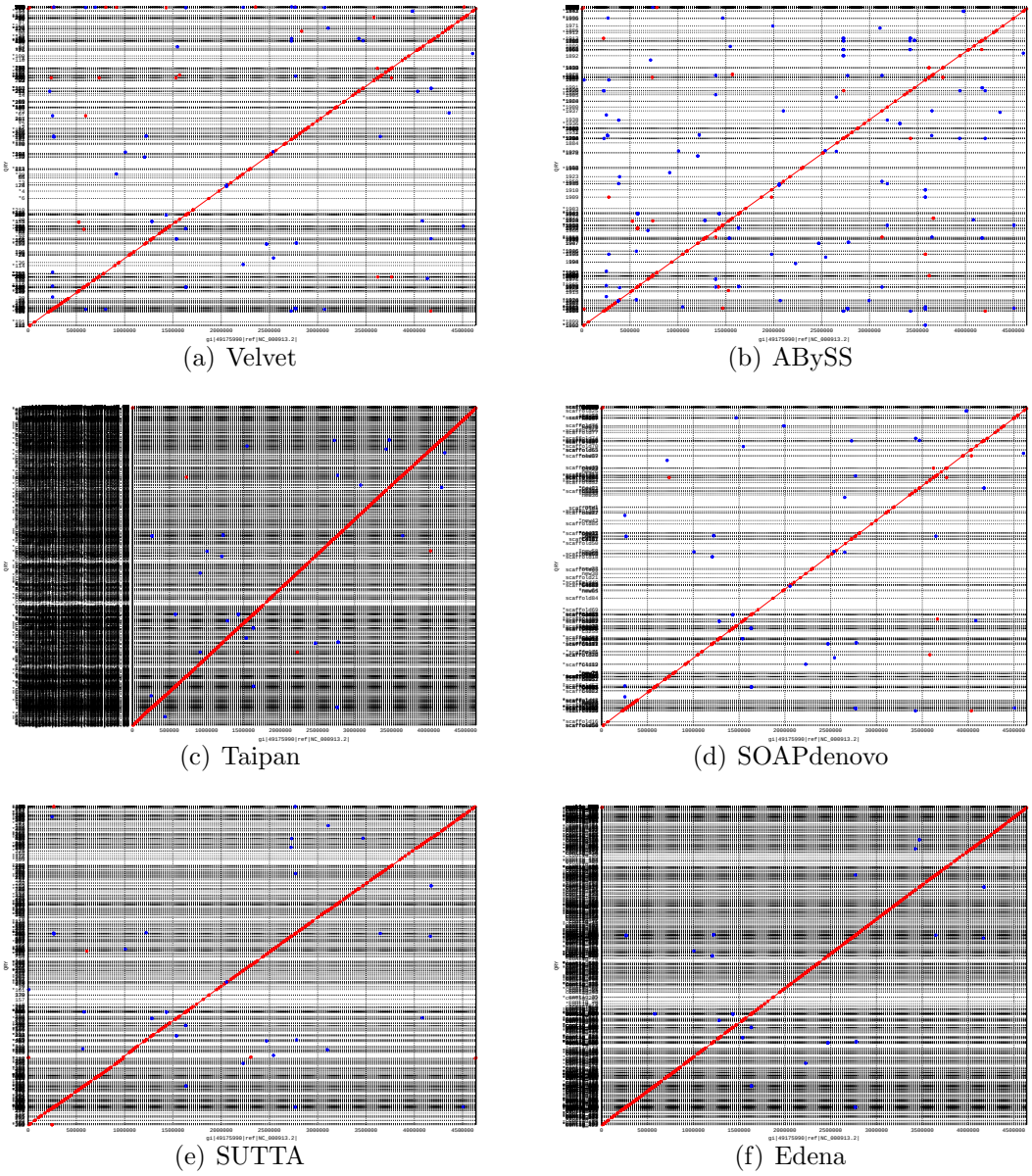


Figure 24: Feature-Response curves by feature type for *Escherichia coli* with mate-pair constraints.



(a) Velvet (b) ABySS

(c) Taipan (d) SOAPdenovo

(e) SUTTA (f) Edena

Figure 25: Dot plots for *E. coli* (with mate-pairs). Assemblies produced by Velvet, ABySS, Taipan, SOAPdenovo, SUTTA and Edena. The horizontal lines indicate the boundary between assembled contigs represented on the y axis.

Table 7: Short Read Assemblers parameter setting.

Assembler	*S. aureus*	*H. acininychis*	*E. coli*
ABySS	k=23	k=27	k=31 n=5
Edena	m=21	m=27	m=30
EULER-SR	k=21	k=27	k=28 CloneLength=215 CloneVar=40
SOAPdenovo	k =21	k=27	k=25 -R
SSAKE	default	default	m=17 o=4 r=0.7 t=1
SUTTA	k=21 W_{mp}=150 W_{de}=10 W_{bb}=140	k=27 W_{mp}=150 W_{de}=30 W_{bb}=140	k=29 W_{mp}=150 W_{de}=20 W_{bb}=140
Taipan	k=19 T=8	k=27 T=18	k=29 T=10
Velvet	k =21 cov_cutoff=7	k=27 cov_cutoff=8	k=29 ins_length=215 cov_cutoff=12 -exp_cov=24

NOTE: Long-read Assemblers have been ran with their default parameters.

Bibliography

[1] Can Alkan, Saba Sajjadian, and Evan E. Eichler. Limitations of next-generation genome sequence assembly. *Nature Methods*, 8(1):61–65, January 2011.

[2] S F Altschul, T L Madden, A A Schäffer, J Zhang, Z Zhang, *et al.* Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–402, September 1997.

[3] T. S. Anantharaman, V. Mysore, and B. Mishra. Fast and cheap genome wide haplotype construction via optical mapping. In *Pacific Symposium on Biocomputing*, p. 385 396, 2005.

[4] Thomas S. Anantharaman, Bud Mishra, and David C. Schwartz. Genomics via optical mapping iii: Contiging genomic dna and variations (extended abstract). In *Proceedings 7th Intl. Cnf. on Intelligent Systems for Molecular Biology: ISMB '99*, volume 7, pp. 18–27. AAAI Press, 1997.

[5] TS Anantharaman, B Mishra, and DC Schwartz. Genomics via optical mapping. ii: Ordered restriction maps. *J Comput Biol.*, 4(2):91–118, 1997.

[6] Marco Antoniotti, Thomas Anantharaman, Salvatore Paxia, and Bud Mishra. Genomics via optical mapping iv: Sequence validation via optical map matching. Technical report, New York University, New York, NY, USA, 2001.

[7] David Applegate, Robert E. Bixby, Vasek Chvátal, and William Cook. Tsp cuts which do not conform to the template paradigm. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*, pp. 261–304, London, UK, 2001. Springer-Verlag.

[8] Chris Armen and Clifford Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. In *CPM*, pp. 87–101, 1996.

[9] Christopher Aston, Bud Mishra, and David C. Schwartz. Optical mapping and its potential for large-scale sequencing projects. *Trends in Biotechnology*, 17(7):297 – 302, 1999.

[10] Tadashi Baba, Fumihiko Takeuchi, Makoto Kuroda, Harumi Yuzawa, Ken ichi Aoki, *et al.* Genome and virulence determinants of high virulence community-acquired mrsa. *The Lancet*, 359(9320):1819 – 1827, 2002.

[11] Serafim Batzoglou, David B. Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, *et al.* ARACHNE: A Whole-Genome Shotgun Assembler. *Genome Research*, 12(1):177–189, 2002.

[12] R. Bisiani. Beam search. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pp. 56–58. Wiley & Sons, 1987.

[13] Frederick R. Blattner, Guy Plunkett, Craig A. Bloch, Nicole T. Perna, Valerie Burland, *et al.* The Complete Genome Sequence of Escherichia coli K-12. *Science*, 277(5331):1453–1462, 1997.

[14] Marten Boetzer, Christiaan V. Henkel, Hans J. Jansen, Derek Butler, and Walter Pirovano, *et al.* Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, 2010.

[15] Sbastien Boisvert, Franois Laviolette, and Jacques Corbeil. Ray: Simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *Journal of Computational Biology*, 17(11):1519–1533, 2010.

[16] Douglas Bryant, Weng-Keen Wong, and Todd Mockler. Qsra - a quality-value guided de novo short read assembler. *BMC Bioinformatics*, 10(1):69, 2009.

[17] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm, 1994.

[18] Jonathan Butler, Iain MacCallum, Michael Kleber, Ilya A. Shlyakhter, Matthew K. Belmonte, *et al.* ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5):810–820, 2008.

[19] Mark J. Chaisson and Pavel A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18(2):324–330, 2008.

[20] Alonzo Church and J. B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

[21] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[22] Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina, and Heinz Himmelbauer. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research*, 17(11):1697–1706, 2007.

[23] Mark Eppinger, Claudia Baar, Bodo Linz, Gnter Raddatz, Christa Lanz, *et al.* Who ate whom? adaptive helicobacter genomic changes that accompanied a host jump from early humans to large felines. *PLoS Genet*, 2(7):e120, 07 2006.

[24] Y Erlich, PP Mitra, M delaBastide, WR McCombie, and GJ Hannon, *et al.* Alta-cyclic: a self-optimizing base caller for next-generation sequencing. *Nat Methods*, 5(5):679–82, 2008 Aug.

[25] Brent Ewing, LaDeana Hillier, Michael C. Wendl, and Phil Green. Base-Calling of Automated Sequencer Traces UsingPhred.I. AccuracyAssessment. *Genome Research*, 8(3):175–185, 1998.

[26] P. Ferragina and G. Manzini. Opportunistic data structures with applications. *Annual Symposium on Foundations of Computer Science*, 41:390–398, 2000.

[27] John Gallant, David Maier, and James Astorer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50 – 58, 1980.

[28] Steven R. Gill, Derrick E. Fouts, Gordon L. Archer, Emmanuel F. Mongodin, Robert T. DeBoy, *et al.* 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(7):2426–2438, 2005.

[29] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Joshua N. Burton, *et al.* High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 2010.

[30] Phil Green. Phrap documentation, 1996. http://www.phrap.org/phredphrap/phrap.html.

[31] Stephen S. Hall. Revolution postponed. *Scientific American*, pp. 60–67, 2010. doi:10.1038/scientificamerican1010-60.

[32] David Hernandez, Patrice Franois, Laurent Farinelli, Magne sters, and Jacques Schrenzel, *et al.* De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*, 18(5):802–809, 2008.

[33] Mohammad Hossain, Navid Azimi, and Steven Skiena. Crystallizing short-read assemblies around seeds. *BMC Bioinformatics*, 10(Suppl 1):S16, 2009.

[34] Xiaoqiu Huang and Anup Madan. CAP3: A DNA Sequence Assembly Program. *Genome Research*, 9(9):868–877, 1999.

[35] Xiaoqiu Huang, Jianmin Wang, Srinivas Aluru, Shiaw-Pyng Yang, and LaDeana Hillier, *et al.* PCAP: A Whole-Genome Assembly Program. *Genome Research*, 13(9):2164–2170, 2003.

[36] Ruo-Wei Hung and Maw-Shang Chang. Solving the path cover problem on circular-arc graphs by using an approximation algorithmstar. *Discrete Applied Mathematics*, 154(1):76–105, 2006.

[37] Ramana M. Idury and Michael S. Waterman. A new algorithm for dna sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995.

[38] International. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, February 2001.

[39] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–945, October 2004.

[40] Sorin Istrail, Granger G. Sutton, Liliana Florea, Aaron L. Halpern, Clark M. Mobarry, *et al.* Whole-genome shotgun assembly and comparison of human genome assemblies. *Proceedings of the National Academy of Sciences of the United States of America*, 101(7):1916–1921, 2004.

[41] William R. Jeck, Josephine A. Reinhardt, David A. Baltrus, Matthew T. Hickenbotham, Vincent Magrini, *et al.* Extending assembly of short DNA sequences to handle error. *Bioinformatics*, 23(21):2942–2944, 2007.

[42] Wei-Chun Kao, Kristian Stevens, and Yun S. Song. BayesCall: A model-based base-calling algorithm for high-throughput short-read sequencing. *Genome Research*, 19(10):1884–1895, 2009.

[43] Richard M. Karp. The role of algorithmic research in computational genomics. *Computational Systems Bioinformatics Conference, International IEEE Computer Society*, 0:10, 2003.

[44] J. Kececioglu and E. Myers. Combinatorial algorithms for dna sequence assembly. *Algorithmica*, 13(1):7–51, February 1995.

[45] David Kelley, Michael Schatz, and Steven Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):R116, 2010.

[46] W. J. Kent. BLAT—The BLAST-Like Alignment Tool. *Genome Research*, 12(4):656–664, March 2002.

[47] W. James Kent and David Haussler. Assembly of the Working Draft of the Human Genome with GigAssembler. *Genome Research*, 11(9):1541–1548, 2001.

[48] Jeffrey M. Kidd, Nick Sampas, Francesca Antonacci, Tina Graves, Robert Fulton, *et al.* Characterization of missing human genome sequences and

copy-number polymorphic insertions. *Nature Methods*, 7(5):365–371, April 2010.

[49] Sun Kim, Haixu Tang, and Elaine R. Mardis. *Genome Sequencing Technology and Algorithms*. Artech House, Inc., Norwood, MA, USA, 2007.

[50] Carl Kingsford, Michael Schatz, and Mihai Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinformatics*, 11(1):21, 2010.

[51] Martin Kircher, Udo Stenzel, and Janet Kelso. Improved base calling for the illumina genome analyzer using machine learning strategies. *Genome Biology*, 10(8):R83, 2009.

[52] A. N. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giorn. 1st. Ital. Attuari*, 4:83–91, 1933.

[53] Stefan Kurtz, Adam Phillippy, Arthur Delcher, Michael Smoot, Martin Shumway, *et al.* Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[54] A. H. Land and A. G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[55] Eric S. Lander and Michael S. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2(3):231 – 239, 1988.

[56] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultra-fast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10, 2009.

[57] Samuel Levy, Granger Sutton, Pauline C Ng, Lars Feuk, Aaron L Halpern, *et al.* The diploid genome sequence of an individual human. *PLoS Biol*, 5(10):e254, 09 2007.

[58] H. Li and R. Durbin. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 25(14):1754, 2009.

[59] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-wah Lam, Siu-ming Yiu, *et al.* SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[60] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, *et al.* De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, 2010.

[61] Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *Algorithms in Bioinformatics*, Lecture Notes in Computer Science, chapter 27, pp. 289–301. Springer, 2007.

[62] Fabian Menges, Giuseppe Narzisi, and Bud Mishra. TotalReCaller: Improved Accuracy and Performance via Integrated Alignment & Base-Calling. *Bioinformatics (under review)*, 2011.

[63] Michael L. Metzker. Emerging technologies in DNA sequencing. *Genome Research*, 15(12):1767–1776, 2005.

[64] Jason R. Miller, Arthur L. Delcher, Sergey Koren, Eli Venter, Brian P. Walenz, *et al.* Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, 2008.

[65] Bud Mishra. Optical mapping. *Encyclopedia of Life Sciences*, 2005.

[66] James C. Mullikin and Zemin Ning. The Phusion Assembler. *Genome Research*, 13(1):81–90, 2003.

[67] Eugene W. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2:275–290, 1995.

[68] Eugene W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–85, 2005.

[69] Eugene W. Myers, Granger G. Sutton, Art L. Delcher, Ian M. Dew, Dan P. Fasulo, *et al.* A Whole-Genome Assembly of Drosophila. *Science*, 287(5461):2196–2204, 2000.

[70] Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, July 2009.

[71] Giuseppe Narzisi. Sutta: Scoring-and-unfolding trimmed tree assembler, 2010. 9th IEEE International Workshop on Genomic Signal Processing and Statistics, Cold Spring Harbor Laboratory (poster).

[72] Giuseppe Narzisi and Bud Mishra. A novel technologically agnostic de novo sequence assembler, 2010. Systems Biology and New Sequencing Technologies, Centre for Genomic Regulation (poster).

[73] Giuseppe Narzisi and Bud Mishra. Comparing de novo genome assembly: The long and short of it. *PLoS ONE*, 6(4):e19175, 04 2011.

[74] Giuseppe Narzisi and Bud Mishra. Scoring-and-unfolding trimmed tree assembler: concepts, constructs and comparisons. *Bioinformatics*, 27(2):153–160, 2011.

[75] Pramila Nuwantha Ariyaratne and Wing-Kin Sung. PE-Assembler: De novo assembler using short paired-end reads. *Bioinformatics*, 2010.

[76] Ian T. Paulsen, Rekha Seshadri, Karen E. Nelson, Jonathan A. Eisen, John F. Heidelberg, *et al.* The Brucellasuis genome reveals fundamental similarities between animal and plant pathogens and symbionts. *Proceedings of the National Academy of Sciences of the United States of America*, 99(20):13148–13153, 2002.

[77] Hannu Peltola, Hans Sderlund, and Esko Ukkonen. SEQAID: a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12(1Part1):307–321, 1984.

[78] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National*

Academy of Sciences of the United States of America, 98(17):9748–9753, 2001.

[79] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748–9753, 2001.

[80] Adam Phillippy, Michael Schatz, and Mihai Pop. Genome assembly forensics: finding the elusive mis-assembly. *Genome Biology*, 9(3):R55, 2008.

[81] Mihai Pop, Daniel S. Kosack, and Steven L. Salzberg. Hierarchical Scaffolding With Bambus. *Genome Research*, 14(1):149–159, 2004.

[82] Mihai Pop and Steven L. Salzberg. Bioinformatics challenges of new sequencing technology. *Trends in Genetics*, 24(3):142 – 149, 2008.

[83] Horst W. J. Rittel and Melvin M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4(2):155–169, June 1973.

[84] Michael Roberts, Brian R. Hunt, James A. Yorke, Randall A. Bolanos, and Arthur L. Delcher, *et al.* A preprocessor for shotgun assembly of large genomes. *Journal of Computational Biology*, 11(4):734–752, 2004.

[85] Jacques Rougemont, Arnaud Amzallag, Christian Iseli, Laurent Farinelli, Ioannis Xenarios, *et al.* Probabilistic base calling of solexa sequencing data. *BMC Bioinformatics*, 9(1):431, 2008.

[86] Steven L. Salzberg and James A. Yorke. Beware of mis-assembled genomes. *Bioinformatics*, 21(24):4320–4321, 2005.

[87] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, 74(12):5463–5467, 1977.

[88] Bertil Schmidt, Ranjan Sinha, Bryan Beresford-Smith, and Simon J. Puglisi. A fast hybrid short read fragment assembly algorithm. *Bioinformatics*, 25(17):2279–2280, 2009.

[89] Jan Schrder, Heiko Schrder, Simon J. Puglisi, Ranjan Sinha, and Bertil Schmidt, *et al.* Shrec: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.

[90] David Schwartz and Michael Waterman. New generations: Sequencing machines and their computational challenges. *Journal of Computer Science and Technology*, 25:3–9, 2010. 10.1007/s11390-010-9300-x.

[91] Colin A. M. Semple. Assembling a view of the human genome. In Michael R. Barnes and Ian C. Gray, editors, *Bioinformatics for Geneticists*, chapter 4, pp. 93–117. John Wiley & Sons, Ltd, 2003.

[92] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, *et al.* ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[93] N. V. Smirnov. Approximate laws of distribution of random variables from empirical data. *Uspekhi Mat. Nauk*, 10:179–206, 1944.

[94] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[95] Daniel Sommer, Arthur Delcher, Steven Salzberg, and Mihai Pop. Minimus: a fast, lightweight genome assembler. *BMC Bioinformatics*, 8(1):64, 2007.

[96] G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, 1(1):9–19, 1995.

[97] J. Tarhio and E. Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, 57(1):131–145, 1988.

[98] J. S. Turner. Approximation algorithms for the shortest common superstring problem. *Inf. Comput.*, 83(1):1–20, 1989.

[99] G a Tuskan, S Difazio, S Jansson, J Bohlmann, I Grigoriev, *et al.* The genome of black cottonwood, Populus trichocarpa (Torr. & Gray). *Science (New York, N.Y.)*, 313(5793):1596–604, September 2006.

[100] J. Craig Venter, Mark D. Adams, Eugene W. Myers, Peter W. Li, Richard J. Mural, *et al.* The Sequence of the Human Genome. *Science*, 291(5507):1304–1351, 2001.

[101] Rene L. Warren, Granger G. Sutton, Steven J. M. Jones, and Robert A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500–501, 2007.

[102] David A. Wheeler, Maithreyan Srinivasan, Michael Egholm, Yufeng Shen, Lei Chen, *et al.* The complete genome of an individual by massively parallel DNA sequencing. *Nature*, 452(7189):872–876, April 2008.

[103] Martin Wu, Ling V Sun, Jessica Vamathevan, Markus Riegler, Robert Deboy, *et al.* Phylogenomics of the reproductive parasite wolbachia pipientis wmel: A streamlined genome overrun by mobile genetic elements. *PLoS Biol*, 2(3):e69, 03 2004.

[104] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.