UNIVERSITY OF CALIFORNIA
RIVERSIDE

Efficient Methods For Analysis of Ultra-deep Sequencing Data

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Seyed Hamid Mirebrahim

December 2015

Dissertation Committee:

Dr. Stefano Lonardi, Chairperson
Dr. Timothy J Close
Dr. Eamonn Keogh
Dr. Michalis Faloutsos

The Dissertation of Seyed Hamid Mirebrahim is approved:

_____

_____

_____

_____

Committee Chairperson

University of California, Riverside

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincerest gratitude to my advisor Dr.Stefano Lonardi for the invaluable guidance and support during my doctoral study. I joined Dr.Lonardi's lab in 2011 without much experience in research, academic writing, and deep knowledge of bioinformatics. I spent 5 years learning from him every single day. Thank you, Dr.Lonardi, for being such an extraordinary advisor.

My gratitude also goes to Dr. Timothy J. Close and his colleagues in the department of Botany and Plant Sciences at UC Riverside. This achievement was not possible without his guidance and advises.

I would also like to thank Dr.Eamonn Keogh, Dr.Tao Jiang, Dr.Tamar Shinar, Dr.Gianfranco Ciardo and Dr.Michalis Faloutsos, my oral qualifying exam, proposal and dissertation defense committee members, for their insightful guidance and valuable comments.

Foremost, I would like to express my gratitude to my parents, my sister and my brother who always had faith in me, and their supports helped me focus on my studies.

Finally, I would like to thank my friends, especially Mohammad Shokoohi-yekta, Mohammad Mehdi Eslamimehr, Ali Basiri, Mehdi Sadri, Masoud Akhoondi, Sara Naseri and Mohammad Asghari who have helped and encouraged me to move forward.

To my parents, Aliakbar and Farzaneh, my brother, Ali and my sister, Sepideh for their constant love, patience and support. Without them, it was simply impossible.

ABSTRACT OF THE DISSERTATION

Efficient Methods for Analysis of Ultra-deep Sequencing Data

by

Seyed Hamid Mirebrahim

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2015
Dr.Stefano Lonardi, Chairperson

Thanks to continuous improvements in sequencing technologies, life scientists can now easily sequence DNA at depth of sequencing coverage in excess of 1,000x, especially for smaller genomes like viruses, bacteria or BAC/YAC clones. As "ultra deep" sequencing becomes more and more common, it is expected to create new algorithmic challenges in the analysis pipeline.

In this dissertation, I explore the effect of ultra-deep sequencing data in two domains: (i) the problem of decoding reads to bacterial artificial chromosome (BAC) clones and (ii) the problem of *de novo* assembly of BAC clones. Using real ultra-deep sequencing data, I show that when the depth of sequencing increases over a certain threshold, sequencing errors make these two problems harder and harder (instead of easier, as one would expect with error-free data), and as a consequence the quality of the solution degrades with more and more data.

For the first problem, I propose an effective solution based on "divide and conquer": the method 'slices' a large dataset into smaller samples of optimal size, decodes each

slice independently, and then merges the results. For the second problem, I show for the first time that modern *de novo* assemblers cannot take advantage of ultra-deep sequencing data. I then introduce a new divide and conquer approach to deal with the problem of *de novo* genome assembly in the presence of ultra-deep sequencing data.

Finally, I report on a novel computational protocol to discover high quality SNPs for cowpea genome. I show how the knowledge of approximate SNP order can be used to order and merge BAC clones and WGS contigs.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The Human Genome Project started in 1990 and reached a major milestone in 2001 with the publication of [1] and was declared "complete" in 2003. The availability of the primary DNA sequence of the human genome completely revolutionized the way life scientists study human health and biological processes in the human body. For instance, it helped researchers to understand the underlying mechanisms that control gene expression or shed light on to the causes of several genetic diseases. The quality of the human genome draft has been improving continuously due to the advances in sequencing technology and the design of novel algorithms for *de novo* genome assembly.

The Human Genome Project approached the task of sequencing and assembling the human genome in a hierarchical manner. The human genome was divided into several large DNA fragments (e.g., BAC clones), each of which was sequenced and assembled individually by sequencing centers around the world. Finally, the sequences were ordered and merged at UC Santa Cruz. The methods developed during the human genome project revolutionized all the aspects of genome sequencing, assembly and analysis.

The majority of the human genome was sequenced using a technique developed by Dr. Frederick Sanger and his colleagues in 1977. Sanger sequence was the dominant sequencing approach for almost three decades: while it is time-consuming and expensive, the results are very accurate.

The cost of sequencing dropped dramatically when the next (second) generation sequencing (NGS) techniques became popular in mid 2000s (Figure 1 reports the average sequencing cost of DNA since 2001).

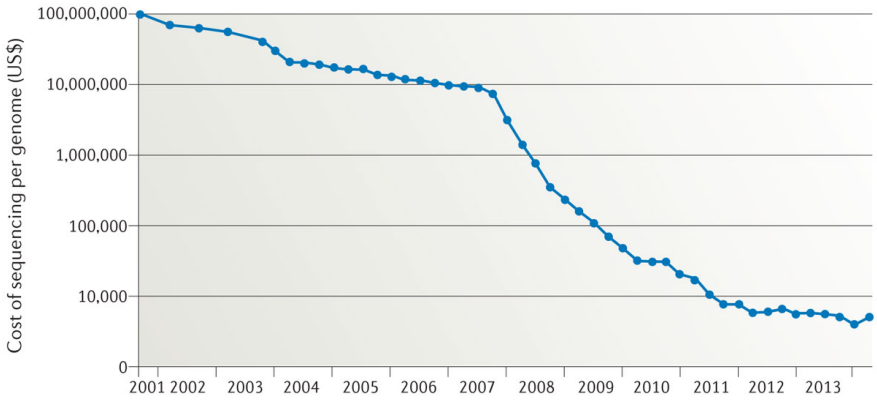

Figure 1. Average sequencing cost of DNA per base between 2001 and 2014 [2]

NGS is a high throughput approach for sequencing DNA that takes advantage of DNA amplification (PCR) *in situ*. Besides sequencing genomic DNA, NGS has been successfully used in gene discovery and in investigation of regulatory elements associated with diseases. Targeted sequencing, which has applications in identification of disease-causing mutations for diagnosis pathological conditions, is much easier with NGS. Also, RNA-Seq (based on NGS) is a powerful alternative to microarray that provides richer information about the transcriptome of a sample and does not require prior knowledge of the target genome.

Despite the advances in sequencing technologies, none of the available technologies is able to sequence a genome from the beginning to the end. Instead, they produce the sequence of short DNA fragments, called *reads*. There are two types of reads: single end

(SE) and paired end (PE). A paired end read consist of two single end reads with a gap in the middle, where the approximate length of the gap is known. Sanger reads are relatively long (800-1000bp) and quite accurate (the quality drops towards the end). NGS reads are significantly shorter than Sanger reads, but the number of reads is orders of magnitude higher than Sanger (at the same cost). An important concept in sequencing is the *depth of sequencing* or *coverage*, which is defined as the average number of reads covering any single base in the genome. For instance, 10x coverage means that we have sufficient reads to cover each base of the genome with ten reads (on average).

Whether they are long or short, reads need to be assembled based on their overlapping ends. A collection of mutually overlapping sequences is called a *contig* which is represented as the consensus of the collections of reads. Ordered sequences of contigs interleaved by gaps are generated through a process called *scaffolding*. The length of the gaps between the contigs are estimated based on paired end reads.

There are two approaches to assemble genomic reads: *de novo* assembly and *reference guided* assembly. *De novo* assemblers exclusively work based on detecting overlaps among the short reads. *De novo* assembly is a difficult computational task, especially when the target genome is large and highly repetitive. It becomes harder when the reads are short because detecting reliable overlaps is more challenging. *De novo* assembly is NP-hard [3] and is used when no prior information about the genome is available. There are different algorithmic approaches for implementing *de novo* assembly, namely the greedy approach, the overlap-layout-consensus method and the eulerian method (based on the de Bruijn graph).

Most of the modern assemblers rely on a data structure called the *de Bruijn graph* that is built by processing the reads. The advantage of the de Bruijn graph is that it does not require the assembler to compute all pair-wise overlaps. Each node in the de Bruijn graph represents a *k*-mer (i.e., a substring of length *k*) that appears in at least one read. Two nodes are connected with an edge if there is an overlap of length *k*-1 between the two corresponding *k*-mers. Eulerian paths in the de Bruijn graph correspond to contigs.

The idea behind reference-guided assembly is to use a closely related genome as a guide to assemble the reads of the target genome. Typically, this approach starts by first mapping the sequenced reads to the reference genome. Once the positions are available, the assembly is carried out locally.

Since the invention of DNA sequencing in the seventies, computational biologists have had to deal with the problem of genome assembly with limited (or insufficient) depth of sequencing. In Chapter 2 of this dissertation, we investigate the opposite problem, that is, the challenge of dealing with excessive depth of sequencing. We explore the effect of ultra-deep sequencing data in two domains: (i) the problem of decoding reads to *bacterial artificial chromosome* (BAC) clones (in the context of the combinatorial pooling design), and (ii) the problem of *de novo* assembly of BAC clones. Using real ultra-deep sequencing data, we show that when the depth of sequencing increases over a certain threshold, sequencing errors make these two problems harder and harder (instead of easier, as one would expect with error-free data), and as a consequence the quality of the solution degrades with more and more data. For the first problem, we propose an effective solution based on 'divide and conquer': we 'slice' a large dataset

into smaller samples of optimal size, decode each slice independently, and then merge the results. Experimental results in Chapter 2 demonstrate a significant improvement in the quality of the decoding and the final assembly. For the second problem, we show for the first time that modern *de novo* assemblers cannot take advantage of ultra-deep sequencing data.

We investigate the problem of *de novo* genome assembly in the presence of ultra-deep sequencing data (i.e., coverage of 1,000x or higher) in more details in Chapter 3. In this chapter, we introduce a new divide and conquer approach to improve the quality of assemblies created from ultra-deep sequencing data. Our proposed meta-assembler SLICEMBLER partitions the input data into optimal- sized "slices" and uses a standard assembly tool to assemble each slice individually. SLICEMBLER uses majority voting among the individual assemblies to identify long contigs that can be merged to the consensus assembly. To improve its efficiency, SLICEMBLER uses a generalized suffix tree to identify these frequent contigs (or fraction thereof). Extensive experimental results on real ultra-deep sequencing data (8,000x coverage) and simulated data show that SLICEMBLER significantly improves the quality of the assembly compared to the performance of the base assembler. In fact, most of the times SLICEMBLER generates error-free assemblies. We also show that SLICEMBLER is much more resistant against high sequencing error rate than the base assembler.

As said, several sequencing technologies are now available to approach the problem of genome sequencing, each of which generates reads with different features like length, error rate, etc. In Chapter 4, we report on a method to assemble heterogeneous

sequencing data. We show that the quality of the assemblies created for a malaria strain with this protocol is higher than the previously available assembly.

Among the variety of analyses carried out once a genome is available is the detection of single nucleotide polymorphism (SNP) and genotyping. A SNP is a DNA sequence variation occurring commonly within a population in which a single nucleotide in the genome differs between members of a biological species or paired chromosomes. For instance, in plants genotyping supports higher-density genetic mapping, pedigree validation, germplasm characterization and marker-assisted breeding. In Chapter 5, we report a novel protocol for detecting high quality single nucleotide polymorphism (SNP) in complex genomes. Our protocol was applied to the cowpea (*Vigna unguiculata*), genome, which is one of the most important legume crops in the semiarid tropics, where it is a good source of protein, fiber, and certain vitamins and minerals. We have discovered around 51,000 high quality SNPs that were used for the design of a high-throughput genotyping platform. In addition, we have ordered and oriented the cowpea genome-wide contigs and BAC assemblies, using the discovered SNPs.

Chapter 2: 'Slicing' sequencing data to improve read decoding accuracy and *de novo* assembly quality

Our group introduced in [4] a novel protocol for clone-by-clone *de novo* genome sequencing that leverages recent advances in combinatorial pooling design (also known as group testing). In the proposed sequencing protocol, subsets of non-redundant genome-tiling bacterial artificial chromosomes (BACs) are chosen to form intersecting pools, then groups of pools are sequenced on an Illumina sequencing instrument via low-multiplex (DNA barcoding). Sequenced reads can be assigned/decoded to specific BACs by relying on the combinatorial structure of the pooling design: since the identity of each BAC is encoded within the pooling pattern, the identity of each read is similarly encoded within the pattern of pools in which it occurs. Finally, BACs are assembled individually, simplifying the problem of resolving genome-wide repetitive sequences.

In [4], the group reported preliminary assembly statistics on the performance of the protocol in four barley (*Hordeum vulgare*) BAC sets (Hv_3–Hv_6). Further analysis on additional barley BAC sets and two genome-wide BAC sets for cowpea (*Vigna unguiculata*) revealed that the raw sequence data for some datasets was of significantly lower quality (i.e., higher sequencing error rate) than others. We realized that our decoding strategy, solely based on the software HASHFILTER [4], was insufficient to deal with the amount of noise in poor quality datasets. We attempted to (i) trim/clean the reads more aggressively or with different methods, (ii) identify low quality tiles on the flow

cell and remove the corresponding reads (e.g. tiles on the 'bottom middle swath'), (iii) identify positions in the reads possibly affected by sequencing 'bubbles' and (iv) post-process the reads using available error-correction software tools (e.g. QUAKE, REPTILE). Unfortunately, none of these steps accomplished a dramatic increase in the percentage of reads that could be assigned to BACs, indicating that the quality of the dataset did not improve very much. These attempts to improve the outcome led however, to a serendipitous discovery: we noticed that when HASHFILTER processed only a portion of the dataset, the proportion of assigned/decoded reads increased. This observation initially seemed counterintuitive: we expected that feeding less data into our algorithm meant that we had less information to work with, thus decrease the decoding performance. Instead, the explanation is that when data is corrupted, more (noisy) data is not better, but worse. The study reported here directly addresses the observation that when dealing with large quantities of imperfect sequencing data, 'less' can be 'more'. More specifically, we report (i) an extensive analysis of the trade off between the size of the datasets and the ability of decoding reads to individual BACs; (ii) a method based on 'slicing' datasets that significantly improves the number of decoded reads and the quality of the resulting BAC assemblies; (iii) an analysis of BAC assembly quality as a function of the depth of sequencing, for both real and synthetic data. Our algorithmic solution relies on a divide-and-conquer approach, as illustrated in Figure 2.

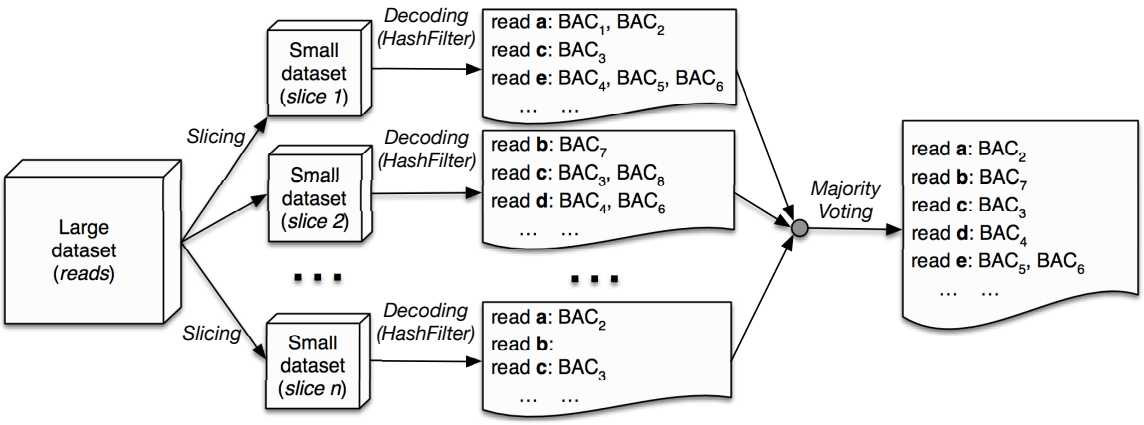

Figure 2. An illustration of the strategy to improve read decoding: (i) a large dataset of reads to be decoded is "sliced" in *n* smaller datasets of optimal size, (ii) each slice is decoded independently and (iii) read-to-BAC assignments for each slice are merged and conflicts are resolved

2.1 Methods

2.1.1. Pooling design

We applied the combinatorial pooling scheme described in [4] to BAC clones for (i) a gene-enriched portion of the genome of *H. vulgare L.* (barley), and (ii) the whole genome of *V. unguiculata* (cowpea). Briefly, in our sequencing protocol we (i) obtain a BAC library for the target organism; (ii) select gene-enriched BACs from the library (optional); (iii) fingerprint BACs and build a physical map; (iv) select a minimum tiling path (MTP) from the physical map; (v) pool the MTP BACs according to the shifted transversal design; (vi) sequence the DNA in each pool, trim/ clean sequenced reads; (vii) assign reads to BACs (*deconvolution*); (viii) assemble reads BAC-by-BAC using a short-read assembler.

We should first note that a rough draft of the ≈5,300Mb barley genome is now available [5]: our BAC sequencing work had contributed to that effort, but is distinct. In

our work, we focused on the gene-enriched portion of the genome [6]. We started with a 6.3×genome equivalent barley BAC library which contains 313,344 BACs with an average insert size of 106 kb [7]. About 84,000 gene-enriched BACs were identified and fingerprinted using high-information content fingerprinting [6, 8]. From the fingerprinting data a physical map was produced [9, 10] and a MTP of about 15,000 clones was derived [6, 11]. Seven sets of n=2,197 clones were chosen to be pooled according to the shifted transversal design [12], which we called Hv_3, Hv_4, . . . , Hv_9 (Hv_1 and Hv_2 were pilot experiments). An additional set of n=1,053 clones (called Hv_{10}) was pooled using the shifted transversal design with different pooling parameters (see below).

A pooling scheme based on the shifted transversal design [12], is defined by (P, L, Γ), where P is a prime number, L defines the number of layers and Γ is a small integer. A layer is one of the classes in the partition of BACs and consists of exactly P pools: the larger the number of layers, the higher is the *decodability*. The decodability of the pooling design determines what is the largest number of 'positive' objects that can be decoded: in our case, a d-decodable pooling design will handle the overlap of at most d MTP clones. By construction the total number of pools is $P \times L$. If we set Γ to be the smallest integer such that $P^{\Gamma+1} \geq N$ where N is the number of BACs that need to be pooled, then the decodability of the design is $\lfloor (L-1)/\Gamma \rfloor$.

For barley sets Hv_3, Hv_4, . . . , Hv_9, we chose parameters P=13, L=7 and Γ=2, so that we could handle $P^{\Gamma+1}$=2,197 samples and make the scheme $\lfloor (L-1)/\Gamma \rfloor$=3-decodable. We expected each non-repetitive read to belong to at most two BACs if the MTP had been computed perfectly, or rarely three BACs when considering imperfections, so we set

d=3. Each of the *L*=7 layers consisted of *P*=13 pools, for a total of 91 BAC pools. In this pooling design, each BAC is contained in *L*=7 pools and each pool contains P^{Γ} =169 BACs. We call the set of *L* pools to which a BAC is assigned, the BAC signature. Any two BAC signatures can share at most Γ=2 pools, and any three BAC signatures can share at most 3Γ = 6 pools. For sets Hv_3–Hv_8, Vu1 and Vu_2, we manually pooled 2,197 BACs thus exhausting all the 'available' signature for the pooling design. However, for set Hv_9 we only used 1,717 signatures. Set Hv_{10} was pooled using a different design: we chose pooling parameters *P*=11, *L*=7 and Γ=2, for a total of $P^{\Gamma+1}$=1,331 BAC signatures, however, we only used 1,053 signatures. BAC signatures that were available but not used in the pooling were called *ghosts*.

Cowpea's genome size is estimated at 620 Mb and it is yet to be fully sequenced. For cowpea we started from a 17X depth of coverage BAC library containing about 60,000 BACs from the African breeding genotype IT97K-499-35 with an average insert size of 150 kb. Cowpea BACs were fingerprinted using high information content fingerprinting [8, 13]. A physical map was produced from 43,717 fingerprinted BACs with a depth of 11X genome coverage [9, 10], and a MTP comprised of 4,394 clones was derived [11]. The set of MTP clones was split in two sets of *n*=2197 BACs (called hereafter Vu_1 and Vu_2), each of which was pooled according to the shifted transversal design [12], with the same pooling parameters used for Hv_3–Hv_9.

To take advantage of the high throughput of sequencing of the Illumina HiSeq2000, 13–20 pools in each set were multiplexed on each lane, using custom multiplexing adapters. After the sequenced reads in each lane were demultiplexed, we obtained an

average of 1,764 million reads in each set with a read length of about 92 bases and an insert size of 275 bases. Reads were quality-trimmed and cleaned of spurious sequencing adaptors, and then reads affected by *Escherichia coli* contamination or BAC vector were discarded. The percentage of *E.coli* contamination averaged around 43%: as a consequence, the average number of usable reads after quality trimming and cleaning decreased to about 824 million, with an average high quality read length of about 89 bases. Table 1 reports the number of reads, number of bases, average read length and *E.coli* contamination for each of the 10 sets (Hv_3, Hv_4, Hv_5, Hv_6, Hv_7, Hv_8, Hv_9, Hv_{10}, Vu_1 and Vu_2). Raw reads for barley and cowpea BACs have been deposited in NCBI SRA accession number SRA051780, SRA051535, SRA051768, SRA073696, SRA051739 (barley); SRA052227 and SRA052228 (cowpea).

Table 1. Basic statistics on the ten sequenced read datasets (seven for barley, two for cowpeas) analyzed in this manuscript

	after demultiplexing				***after demultiplexing/cleaning/trimming***		
	reads (M)	*bases* (Mbp)	*read len* (bp)	*% E.coli*	*reads* (M)	*bases* (Mbp)	*read len* (bp)
Hv3	2,476	227,773	92.00	41.12%	1,240.2	110,056	88.74
Hv4	1,363	125,273	91.91	39.36%	713.4	63,384	88.85
Hv5	1,142	105,089	92.00	51.11%	505.1	45,088	89.27
Hv6	1,133	104,239	92.00	65.96%	282.4	24,970	88.42
Hv7	2,288	210,535	92.00	46.11%	928.9	82,503	88.82
Hv8	1,802	165,803	92.00	44.04%	730.8	64,651	88.46
Hv9	1,596	146,816	92.00	40.66%	736.2	65,697	89.24
Hv10	971	89,370	92.00	20.95%	748.2	67,600	90.36
Vu1	2,475	227,696	92.00	36.66%	1,208.1	108,666	89.95
Vu2	2,402	221,006	92.00	43.12%	1,144.6	103,026	90.01

2.1.2. Read decoding analysis

The 91 pools (77 for Hv_{10}) of trimmed reads for barley and cowpea were processed using our *k*-mer based algorithm called HASHFILTER, which is fully described in [4].

Briefly, HASHFILTER builds a hash table of all distinct *k*-mers in the 91 (or 77) pools of reads, and records for each *k*-mer the set of pools where it occurs. Then it processes each read individually: (i) a read *r* is decomposed in its constitutive *k*-mers; (ii) the set of pools of each *k*-mer is fetched from the hash table, and matched against the BAC signatures (allowing for a small number of missing/extra pools); (iii) the union of *k*-mer signatures that match a valid BAC signature determines the BAC assignment for read *r*. Recall that since our pooling is 3-decodable, each read can be assigned to 0–3 BACs.

For some of the datasets, the percentage of reads decoded using this procedure was very low. For instance HASHFILTER could decode only 23.8% of the reads in Hv_9. We suspected a higher percentage of sequencing errors in Hv_9 compared with previous datasets, so we conducted many experiments to improve the decoding performance on this dataset, including (i) tweaking the parameters and the algorithm HASHFILTER, (ii) correcting the reads using QUAKE and REPTILE, (iii) increasing the stringency for quality values in the trimming step, (iv) considering only reads that appeared exactly at least twice, (v) using on the left or the right read (for paired-end reads). None of these actions increased the number of decoded reads in $Hv_9 > 36.6\%$, which was still unsatisfactory. To our initial surprise, running HASHFILTER on a fraction of the reads yielded higher decoding percentages, which suggested the idea to 'slice' the data.

We remind the reader that HASHFILTER has the ability to ignore *k*-mers affected by sequencing errors: if the number t of non-zero counts of a *k*-mer signature belongs to the interval $[L+1, 2L-\Gamma-1]$, HASHFILTER removes from the *k*-mer signature the $t - L$ pools with the lowest counts [for details, see Case 4 and 6 of step G in [4]]. If one assumes that

k-mers with sequencing errors are rarer than error-free *k*-mers, spurious pools will have a low *k*-mer count and will be removed before the reads are decoded. In addition to this feature, HASHFILTER also has the option to disregard entirely a *k*-mer that appears rarely, which is likely to contain sequencing errors.

The next question was to study the dependency between the size of the dataset and the performance of the decoding algorithm. To this end, we took samples of the original 91 (or 77) set of reads in sizes of 0.5, 1, 2, 3, 4 and 5M reads (details on the sampling method can be found in the next section) and computed the percentage of reads decoded by HASHFILTER on these samples of increasing sizes. Figure 3A shows the percentages of decoded reads for sets Hv_3, Hv_4, Hv_5, Hv_6 and Hv_7; Figure 3B is for Hv_8, Hv_9, Hv_{10}, Vu_1 and Vu_2. The x-axis is the number of reads per pool (in millions) given in input to HASHFILTER (*k*=26). The rightmost point on these graphs corresponds to the full dataset.

Several observations on Figure 3 are in order. First, observe that when the number of reads per pool is too small (0.5–1M) the percentage of reads decoded by HASHFILTER is low. Similarly, when the number of reads per pool is large, the percentage of reads decoded by HASHFILTER can be low for some datasets. We believe that when the input size is small, there is not enough information in the hash table of *k*-mers to accurately decode the reads. However, when the input size is large, sequencing errors in the data introduce spurious *k*-mers in the hash table, which has the effect of deteriorating HASHFILTER's decoding performance. Observe that almost all these curves reach a maximum in the range 1–3M reads.

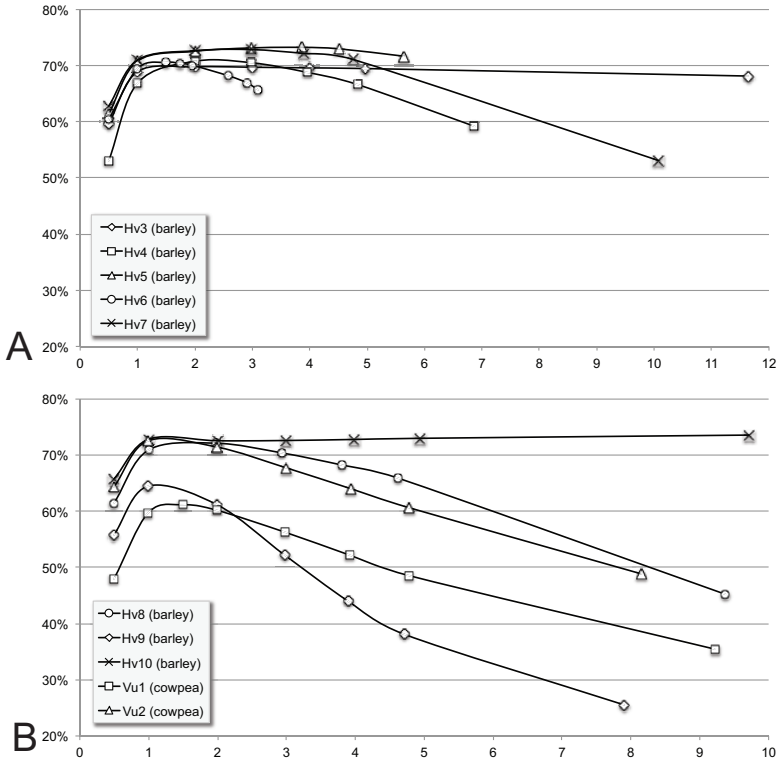

Figure 3. The percentage of reads decoded by HASHFILTER (k=26) on dataset (A) Hv_3, Hv_4, Hv_5, Hv_6 and Hv_7 (B) Hv_8, Hv_9, Hv_{10}, Vu_1, and Vu_2 as a function of the number of reads given in input (x: number of million of reads sampled in each dataset)

For datasets whose 'optimal number' of reads is low, we can speculate the amount of sequencing error to be higher. Also observe the large variability among these 10 datasets. At one extreme, graphs for Hv_3, Hv_{10} and Hv_5 are very 'flat' indicating low sequencing errors; at the other extreme, graphs for Vu_1 and Vu_2 degrade very quickly after the peak, indicating poorer data quality. We also carried out a simulation study using synthetic reads generated from the rice genome (*Oryza sativa*). For this simulation we started from an MTP containing 3,827 BACs with an average length of about 150 kb, which spanned 91% of the rice genome (which is about 390 Mb). We pooled *in silico* a subset of 2,197 BACs from the set above according to the shifted transversal design [see Lonardi et al.,

(2013) for details]. We generated 2M synthetic reads using WGSim (github.com/lh3/wgsim) for each of the 91 resulting rice BAC pools. Reads were 104 bases long with 1% sequencing error rate (no insertions and deletions errors were allowed). A total of 208 Mbp gave an expected 56X coverage for each BAC. We ran HASHFILTER on the read datasets in slices of 0.25, 0.5, 1, 1.5 and 2M (full dataset). The percentage of decoded reads (see Figure 4) peaks at 1.5M, and mirrors the observations made on real data. Even for synthetic reads, more data does not necessarily imply improved decoding performance.

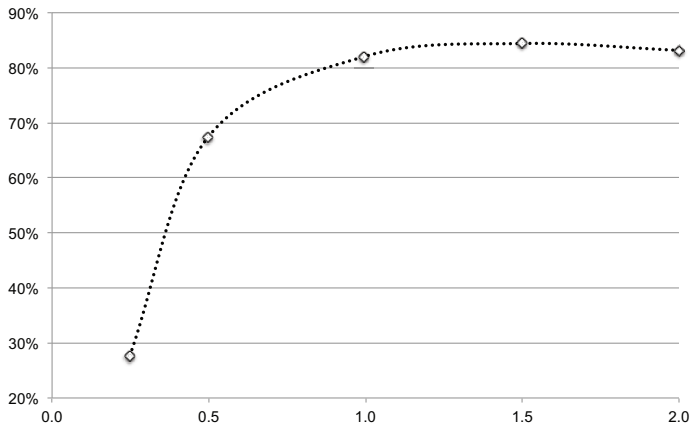

Figure 4. The percentage of synthetic reads decoded by HASHFILTER on the rice genome as a function of the number of reads given in input (x: number of million of reads per pool)

2.1.3. Improved decoding algorithm

Our improved decoding algorithm first executes HASHFILTER on progressively larger samples of the dataset (e.g. 0.5, 1, 2, 3, 4, 5M and full dataset) for a given value of *k*. Our sampling algorithm selects reads uniformly at random along the input file: taking a prefix

of the dataset is not a good idea because reads in the file are organized according to their spatial organization on the flowcell, possibly introducing biases.

When the sample size is greater than the pool size, the entire pool is used for decoding. Otherwise, reads in pools larger than the sample size are uniformly sampled in order to meet the sample size constraint. As a result of this process, the size of each pool in a 'slice' will be *at most* the sample size, but some of the pools will be smaller. The objective is to find the sample size that maximizes the number of reads decoded by HASHFILTER.

We observed that the optimal value of the sample size is somewhat independent from k as long as it is chosen 'reasonably large', say k >20 for large eukaryotic genomes. Figure 5 illustrates that running HASHFILTER with k = 20; 23; . . . ; 32 gives rise to parallel curves. One can save time by running HASHFILTER with smaller values of k in order to find the optimal data size.

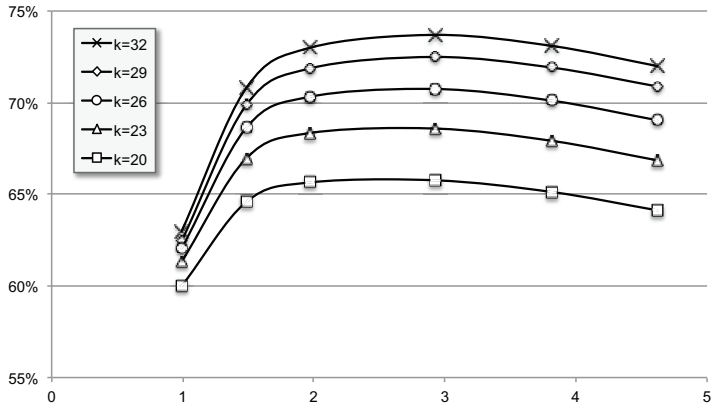

Figure 5. The percentage of reads decoded by HASHFILTER on one of the barley datasets (Hv_8) for several choices of the k-mer size (x: number of million of reads sampled per pool)

Once the optimal sample size n is determined, the algorithm finds the size m of the largest pool to calibrate d datasets (hereafter called slices) each one of which has at most n reads per pool. For instance, if the optimal slice size is n = 2M reads, and the largest pool has m = 10M reads, the algorithm will create $d = m/n = 5$ slices: each one will be composed of 91 pools, each of which has at most 2M reads. Observe that the number of reads in each pool can vary significantly. For instance in Hv_3, the largest pool has almost 23M reads, and the smallest has about 3M reads. Smaller pools will contribute their reads to multiple slices. For instance, if there is a pool of size 2M in the same example described earlier, these reads will appear in all five slices. In general, if a pool size is n, the entire pool will be used in each slice.

Then, the algorithms run HASHFILTER d times, once on each of the d slices—which involves creating d individual hash tables. For this step, we recommend using the largest possible value of k (k=32), because the percentage of decoded reads for a given input size increases with k (see Figure 5). Then, the algorithm merges the d independent HASHFILTER ‘s outputs. If a read is decoded in only one slice, it will be simply copied in the output. If a read is decoded multiple times in different slices and the independent decodings do not agree, a conflict resolution step is necessary. In our running example, reads in the small 2M-reads pool will be decoded five times: it is possible that HASHFILTER will assign a read to five different BAC sets. In order to identify reads decoded multiple times, our algorithm first concatenates the d text outputs of HASHFILTER, then sorts the reads by their unique identifier (ID), so that reads with the same ID are consecutive in the file. Recall that HASHFILTER assigns each read to a set composed of 0-3 BACs. A *group* is the

set of all BAC (assignment) sets for a single-end read. When a read is paired-end, we have a *left group* for the left read and a *right group* for the right read. For instance in Figure 2, single-end read **c** is decoded by HASHFILTER at least three times: in slice 1 read **c** is assigned to BAC_3, in slice 2 it is assigned to BAC_3 and BAC_8, and in slice n it is assigned to BAC_3. The set $\{\{BAC_3\}; \{BAC_3,BAC_8\} ; \{BAC_3\}\}$ is the group for read **c**. If a read has been decoded at least twice by HASHFILTER and the sets in its group are not identical, the following algorithm computes the most likely assignment according to a set of rules, which are checked in order (i.e. the first one that applies is used, and subsequent rules are not considered).

i. if a read is single-end and its group contains one or more BACs which have 75%-majority or higher, then the read is assigned to those majority BAC(s);

ii. if a read is paired-end, and both its left group and its right group are non-empty, and the union of the left and the right group contains one or more BACs which have 50%-majority or higher, then both the left and the right read are assigned to those majority BAC(s);

iii. if a read is paired-end, and either its left or its right group are empty, and the non-empty group contains one or more BACs which have 75%-majority or higher, then both the left and the right read are assigned to those majority BAC(s);

iv. if a read is paired-end, and its left group is not identical to its right group, then both the left and the right read are not assigned.

In the example on read **c**, since BAC_3 is has 100%-majority (appears in all three assignments) but BAC_8 has only 33%-majority (appears in one of the three assignments), we assign read c to BAC_3 but not to BAC_8.

2.2 Experimental Results

Once all the decoded reads are assigned to 1–3 BACs using the procedure above, VELVET [14] is executed to assemble each BAC individually. As was done in [4], we generated multiple assembly for several choices of VELVET's *l*-mer (hash) size (25–79, step of 6). The assembly reported is the one that maximizes the n50 (n50 indicates the length for which the set of all contigs of that length or longer contains at least half of the total size of all contigs).

We employed several metrics to evaluate the improvement in read decoding and assembly enabled by the slicing algorithm. For one of the barley sets (Hv_{10}) we executed HASHFILTER using several choices of k (k = 20; 23; 26; 29; 32) on the full 748M reads dataset (i.e. with no slicing) as well as with k=32 using the slicing algorithm described ealrier. The first five rows of Table 2 summarize the decoding results. First, observe that as we increase k, the number of decoded reads increases monotonically. However, if one fixes k (in this case k=32, which is the maximum allowed by HASHFILTER), slicing Hv_{10} in 4 slices of 4M reads increases significantly the number of decoded reads (84.60 compared with 77.19%) available for assembly. Analysis of the number of assignments to ghost BACs also shows significant improvement in the decoding accuracy when using slicing: 0.000086% of the reads are assigned to unused BAC signatures compared with

0.000305–0.001351% when HASHFILTER is used on the full dataset. We carried out a similar analysis on Hv_9: when the full dataset was processed with HASHFILTER (k=26), the number of reads assigned to ghost BACs was very high, 1.9M reads out of 196M (0.9653%). When the optimal slicing is used (k=32), only 19,140 reads out of 516Mare assigned to ghost BACs (0.0037%). Also, observe in Table 2 how the improved decoding affects the quality of the assembly for Hv_{10}. When comparing no slicing to slicing-based decoding, the average n50 jumps from 12,260 to 42,819 bp (both for k=32) and the number of reads used by VELVET in the assembly increases from 86.7 to 90.7%.

Table 2. Decoding and assembly statistics for the Hv_{10} barley set for several choices of k on the full dataset, and for the improved slicing algorithm

	no slicing					***slicing***
	k=20	*k=23*	*k=26*	*k=29*	*k=32*	*k=32*
reads decoded (%)	67.76%	71.07%	73.57%	75.56%	77.19%	84.60%
reads decoded (M)	511	536	555	570	582	617
reads assigned to ghost BACs (%)	0.000498%	0.000305%	0.000480%	0.000484%	0.001351%	0.000086%
reads to be assembled (M)	704	724	739	748	723	695
coverage (x)	502	502	528	502	517	499
reads used by VELVET (%)	73.6%	77.9%	80.8%	81.8%	86.7%	90.7%
n50 (bp)	3,634	5,143	7,069	8,877	12,260	42,819
sum/size (%)	102.8%	102.8%	100.5%	97.9%	89.5%	121.9%
observed genes (27 expected)	20	20	20	20	20	20
coverage of observed genes (%)	94.0%	94.0%	94.0%	94.0%	94.1%	94.0%

For Hv_{10}, we also measured the number of decoded reads that map (with 0, 1, 2 and 3 mismatches) to the assembly of a subset of 26 BACs that are available from [15]. Table 3 reports the average percentage of decoded reads (either from the full dataset or from the optimal slicing) that BOWTIE can map to the 454-based assemblies. Observe how the slicing step improves by 6–7% the number of reads mapped to the corresponding BAC assembly, suggesting a similar improvement in decoding accuracy. Similar improvements in decoding accuracy were observed on the other datasets (data not shown).

Table 3. A subset of 26 BACs in Hv_{10} have a 454-based assembly available from (Stein et al., 2012). The table reports the percentage of the reads for those 26 BACs that can be mapped (with BOWTIE with 0,1,2 and 3 mismatches) to the corresponding assemblies

	no slicing *k=32*	***slicing*** *k=32*
0 mismatches	75.2%	82.4%
1 mismatch	78.7%	85.9%
2 mismatches	80.5%	87.4%
3 mismatches	82.3%	88.7%

On Hv_8, we investigated the effect of the slice size on the decoding and assembly statistics: earlier we claimed that the optimal size corresponds to the peak of the graphs in Table 3. For instance, notice that the peak for Hv_8 is 2M reads. We decoded and assembled reads using slicing sizes of 2M reads as well as (non-optimal) slice size of 3M reads. The experimental results are shown in Table 4. Observe that the decoding with 3M does not achieve the same decoding accuracy or assembly quality of the slicing with 2 M, but again both are significantly better than without slicing. Again, notice in Table 4 how improving the read decoding affects the quality of the assembly. The average n50 increases from 4126 bp (k=26, no slicing) to 34,262 bp (k=32, optimal slicing) and the number of reads used by VELVET in the assembly increases from 55.6 to 91.2%, respectively. For Hv_8, 207 genes were known to belong to a specific BAC clone [4]: the assembly using slicing-based coding recovered at least 50% of the sequence of 187–190 of them, compared with 178 using no slicing.

Table 4. Decoding and assembly statistics for Hv_8: comparing no slicing and slicing with two different slice sizes (2M reads is optimal according to the peak in Figure 2)

	no slicing	***slicing***	
	k=26	*k=32, 3M*	*k=32, 2M*
reads decoded (%)	31.68%	78.98%	82.74%
reads decoded (M)	270	539	600
reads to be assembled (M)	289	591	669
coverage (x)	94	197	223
reads used by VELVET (%)	69.0%	92.6%	91.6%
n50 (bp)	4,126	31,226	34,262
sum/size (%)	55.6%	97.0%	102.0%
observed genes (207 expected)	178	190	187
coverage of observed genes (%)	86.0%	91.1%	91.2%

Finally, we compared the performance of our slicing method against the experimental results in [4], which were obtained by running HASHFILTER with no data slicing (k=26). The basic decoding and assembly statistics when no slicing is used are reported in Table 5. First, observe the large variability of results among the 10 sets. Although the average number of decoded reads for k=26 is 460M, there are sets which have less than half that amount (Hv_6 and Hv_9) and sets have more than twice the average (e.g. Hv_3). As a consequence, the average fold-coverage ranges from 72X (Hv6) to 528X (Hv_{10}). In general, the assembly statistics (without slicing-based decoding) are not very satisfactory: the n50 ranges from 2630 (Hv_9) to 8,190 bp (Hv_3); the percentage of reads used by VELVET ranges from 66.0 (Hv_9) to 85.9% (Hv_3 and Hv_4); the percentage of known genes covered at least 50% of their length by the assemblies ranged from 66% (Hv_4) to 97% (Hv_3).

Table 5. Decoding and assembly statistics for the ten datasets using k = 26 on the full dataset (no slicing)

	decoding		***Velvet assembly*** ($l = 25, 31, \ldots, 79$, best n50)			
	reads (M)	*coverage* (x)	*n50* (bp)	*reads used* (%)	*sum/size* (%)	*observed/expected genes* (%)
Hv3	1,099.0	431.0	8,190	85.9%	96.7%	1,433/1,471 (97.42%)
Hv4	393.2	135.5	5,718	85.9%	85.9%	312/473 (65.96%)
Hv5	483.0	158.9	8,048	84.5%	93.2%	194/226 (85.84%)
Hv6	218.0	72.0	6,032	83.2%	79.9%	208/244 (85.25%)
Hv7	330.0	110.0	5,352	75.9%	63.7%	201/228 (88.16%)
Hv8	289.0	94.2	4,126	69.0%	55.6%	178/207 (85.99%)
Hv9	208.0	95.8	2,630	66.0%	38.5%	262/361 (72.58%)
Hv10	739.0	528.0	7,069	80.8%	100.5%	20/27 (74.07%)
Vu1	369.0	88.5	4,150	67.6%	49.5%	461/612 (75.33%)
Vu2	448.0	126.0	5,670	75.1%	56.1%	406/503 (80.72%)
Average	457.6	184.0	5,699	77.4%	72.0%	(81.13%)

Table 6. Decoding and assembly statistics for the ten datasets using k = 32 and optimal slicing

slicing	***decoding***		***Velvet assembly*** ($l = 25, 31, \ldots, 79$, best n50)			
(no. slices×size)	*reads* (M)	*coverage* (x)	*n50* (bp)	*reads used* (%)	*sum/size* (%)	*observed/expected genes* (%)
Hv3 (11×2M)	1,156.0	460.0	28,477	90.4%	123.0%	1,437/1,471 (97.69%)
Hv4 (8×2M)	595.6	205.9	28,341	93.9%	114.5%	319/473 (67.44%)
Hv5 (4×4M)	471.0	155.5	31,038	93.6%	101.0%	196/226 (86.73%)
Hv6 (6×1.5M)	243.0	81.1	25,194	92.9%	89.4%	206/244 (84.43%)
Hv7 (15×3M)	785.0	264.0	39,742	91.1%	104.0%	204/228 (89.47%)
Hv8 (12×2M)	669.0	223.0	34,262	91.6%	102.0%	187/207 (90.34%)
Hv9 (14×1.25M)	516.0	246.0	32,634	94.3%	103.2%	309/361 (85.60%)
Hv10 (4×5M)	695.0	499.0	42,819	90.7%	121.9%	20/27 (74.07%)
Vu1 (12×1.5M)	907.0	232.0	16,388	89.7%	89.7%	510/612 (83.33%)
Vu2 (14×1.5M)	970.0	283.0	20,748	91.5%	93.6%	446/503 (88.67%)
Average	700.8	265.0	29,964	92.0%	104.2%	(84.78%)

When we decoded the same 10 datasets using the optimal slice size (using this time k=32) the assemblies improved drastically. The decoding and assembly statistics are summarized in Table 6: note that each set has its optimal size and the corresponding number of slices. First observe how the number of decoded reads increased significantly for most datasets (e.g. 330–785M for Hv_7, 289–669M for Hv_8, 209–516M for Hv_9, 369–907M for Vu_1 and 448–695M for Vu_2). Only for two datasets the number of decoded reads decreased slightly (by 12M reads in Hv_5, and by 44M in Hv10). For all the datasets, the average n50 increased significantly from an average of about 5.7 to 30 kbp. Even for

datasets for which slicing decreased the number reads (Hv_5 and Hv_{10}), the n50 increased significantly. The number of reads used by VELVET increased from an average of 77–92%; the fraction of known genes that were recovered by the assemblies increased from 81 to 85%. We recognize that the improvement from Table 5 to 6 is not just due to the slicing, but also to the increased *k* (from 26 to 32). We have already addressed this point in Tables 1–3, where we showed that increasing *k* from 26 to 32 helps the decoding/assembly but the main boost in accuracy and quality is due to slicing. Recall that the assemblies in Table 5 to 6 were carried out using VELVET with $l = 25; 31; \ldots; 79$ and choosing the assembly with the largest n50. On the Hv_3 dataset, we have also tested VELVET with fixed *l*=49, SPADES [16]with $l = 31; 33; \ldots; 79$, and IDBA-UD [17] with $l = 31; 33; \ldots; 79$ (see Table 7). VELVET (best n50) and SPADES' performance were comparable, while IDBA-UD achieved lower n50. We also tested VELVET with *l*=49, and SPADES with $l = 31; 33; \ldots; 79$ on all the other datasets (Table 7). Setting *l*=49 for VELVET led to less 'bloated' assemblies, somewhat comparable to SPADES' output.

Table 7. Assembly statistics on all barley and cowpea datasets (sliced optimally as in Table 5 in the main manuscript, and decoded using $k = 32$ in HASHFILTER) using VELVET, SPADES and IDBA_UD for several choices of the hash size

		average number of scaffolds of a given size						average number contigs of a given size										
		≥50	≥100	≥200	≥400	≥1K	≥10K	≥50	≥100	≥200	≥400	≥1K	≥10K	*n50*	*max*	*sum*	*sum/size*	*expected/observed*
Hv3	VELVET ($l = 49$)	29.94	29.94	18.76	14.38	10.22	3.23	36.16	35.94	23.09	17.76	12.98	3.08	22,120	33,911	103,795	96.90%	1,471/1,435
	SPADES ($l = 49$)	43.50	41.32	37.78	22.81	13.64	2.67	43.74	41.55	38.01	23.04	13.87	2.65	14,350	27,255	102,260	95.60%	1,471/1,435
	SPADES ($l = 31, 33, \ldots, 79$)	27.15	26.78	25.71	17.04	10.79	2.96	27.32	26.95	25.89	17.21	10.96	2.97	24,121	35,758	104,721	97.90%	1,471/1,434
	IDBA ($l = 31, 33, \ldots, 79$)	54.12	44.39	32.93	24.46	14.35	2.74	54.15	44.41	32.95	24.47	14.36	2.75	13,063	26,168	105,889	99.20%	1,471/1,435
Hv4	VELVET ($l = 49$)	31.26	31.26	23.63	19.76	13.72	3.67	40.20	40.04	30.78	25.21	17.43	3.40	20,751	34,380	123,644	104.90%	473/319
	SPADES ($l = 31, 33, \ldots, 79$)	27.72	27.43	26.48	20.91	13.61	3.48	27.89	27.60	26.65	21.09	13.78	3.48	23,914	37,041	124,614	105.70%	473/320
Hv5	VELVET ($l = 49$)	26.40	26.40	19.13	16.11	11.34	3.44	34.61	34.46	25.68	21.38	15.24	3.28	24,890	37,365	114,926	92.30%	226/196
	SPADES ($l = 31, 33, \ldots, 79$)	23.92	23.70	22.88	17.91	11.68	3.25	24.09	23.86	23.04	18.07	11.84	3.26	26,246	38,463	115,537	92.90%	226/196
Hv6	VELVET ($l = 49$)	26.33	26.33	19.27	16.37	11.95	3.13	37.79	37.67	28.06	23.01	16.42	2.59	20,038	31,443	104,192	84.91%	244/206
	SPADES ($l = 31, 33, \ldots, 79$)	22.32	22.02	21.31	17.64	12.36	2.98	22.47	22.17	21.46	17.79	12.50	2.98	20,287	31,613	104,100	84.91%	244/205
Hv7	VELVET ($l = 49$)	22.46	22.46	15.31	12.75	9.33	3.16	28.34	28.21	19.85	16.22	11.98	3.17	27,415	38,302	104,602	85.72%	228/204
	SPADES ($l = 31, 33, \ldots, 79$)	19.96	19.65	18.70	14.61	9.58	2.85	20.12	19.81	18.86	14.77	9.74	2.86	31,109	41,680	105,170	86.29%	228/203
Hv8	VELVET ($l = 49$)	27.76	27.76	20.86	17.34	11.62	3.04	33.71	33.59	25.35	20.68	13.92	2.92	22,222	34,148	104,857	85.48%	207/188
	SPADES ($l = 31, 33, \ldots, 79$)	24.79	24.59	23.81	18.37	11.56	2.91	24.95	24.74	23.97	18.53	11.72	2.92	24,591	36,298	105,466	86.02%	207/188
Hv9	VELVET ($l = 49$)	26.94	26.94	20.77	17.18	11.31	3.08	32.69	32.57	25.24	20.37	13.51	3.02	22,419	34,677	105,679	92.44%	361/310
	SPADES ($l = 31, 33, \ldots, 79$)	24.59	24.29	23.38	18.35	11.33	2.94	24.76	24.46	23.55	18.53	11.50	2.95	25,293	37,225	106,874	93.52%	361/310
Hv10	VELVET ($l = 49$)	43.55	43.55	21.91	12.63	8.63	3.21	51.54	51.17	26.00	16.18	11.98	3.33	31,052	42,916	113,098	93.16%	27/20
	SPADES ($l = 31, 33, \ldots, 79$)	34.95	34.52	33.22	15.44	9.18	2.81	35.14	34.71	33.41	15.63	9.37	2.83	36,244	47,544	113,904	93.92%	27/20
Vu1	VELVET ($l = 49$)	45.42	45.42	35.62	27.83	16.73	3.13	50.08	49.93	38.94	29.66	17.64	3.01	12,470	24,417	118,080	69.54%	612/519
	SPADES ($l = 31, 33, \ldots, 79$)	44.33	42.73	39.14	27.15	15.07	3.31	44.43	42.83	39.24	27.25	15.17	3.31	16,972	29,771	123,597	73.60%	612/536
Vu2	VELVET ($l = 49$)	29.97	29.97	22.78	18.75	12.71	2.90	33.75	33.64	25.31	20.21	13.56	2.80	14,894	25,388	99,006	68.15%	503/449
	SPADES ($l = 31, 33, \ldots, 79$)	29.99	28.68	25.83	19.07	11.39	2.94	30.07	28.76	25.91	19.15	11.47	2.94	19,686	30,269	102,694	71.19%	503/450

As a final step, we investigated how the depth of sequencing affects BAC assembly quality. To this end, we multiplexed 16 barley BACs on one lane of the Illumina HiSeq2000, using custom multiplexing adapters. The size of these BACs ranged 70–185 kbp. After demultiplexing the sequenced reads, we obtained 34.4M 92-bases paired-end reads (insert size of 275 bases). We quality-trimmed the reads, then cleaned them of spurious sequencing adaptors; finally reads affected by *E.coli* contamination or BAC vector were discarded. The final number of cleaned reads was 23.1 M, with an average length of 88 bases. The depth of sequencing for the 16 BACS ranged from 6,600X to 27,700X (see Table 7).

Table 8. Basic statistics on the read datasets for the 16 barley BACs sequenced individually

			after demultiplexing			***after demux/cleaning/trimming***		
BAC	*approx size* (bp)	*set*	*reads* (M)	*bases* (Mbp)	*% E.coli*	*reads* (M)	*bases* (Mbp)	*coverage* (x)
052L22	105,788	Hv4	21.534	1,981	10.94%	16.795	1,488	14,065
152O10	117,543	Hv3, Hv9	18.575	1,709	16.93%	13.101	1,154	9,812
192B13	112,841	Hv4	19.581	1,801	12.20%	13.950	1,225	10,853
574B01	92,859	Hv3	32.952	3,032	15.27%	23.414	2,059	22,172
630P05	110,490	Hv3	16.224	1,493	15.45%	11.102	971	8,792
727J05	131,648	Hv8	22.613	2,080	14.80%	15.801	1,388	10,546
772L04	116,367	Hv3, Hv10	21.837	2,009	17.44%	14.407	1,255	10,784
773A02	185,718	Hv3, Hv10	21.756	2,002	15.86%	14.909	1,309	7,049
773F12	96,385	Hv3, Hv10	20.868	1,920	14.61%	14.661	1,286	13,341
773H21	71,701	Hv3, Hv10	23.679	2,178	16.38%	16.770	1,473	20,540
773L22	92,859	Hv3, Hv10	17.031	1,567	16.16%	12.282	1,081	11,639
774D07	117,543	Hv3, Hv10	13.077	1,203	20.51%	8.880	778	6,621
774G18	90,508	Hv3, Hv10	24.571	2,261	15.89%	16.763	1,469	16,229
774L04	103,437	Hv3, Hv10	22.053	2,029	17.13%	15.219	1,334	12,895
774O01	95,209	Hv3, Hv10	41.579	3,822	14.76%	29.877	2,642	27,754
789L09	84,631	Hv3	37.000	3,404	15.20%	25.730	2,264	26,754

Another set of 52 barley BACs was sequenced by the Department of Energy Joint Genome Institute using Sanger long reads. All BACs were sequenced and finished using PHRED/PHRAP/CONSED to a targeted depth of 10X. The primary DNA sequences for

each of these 52 BACs were assembled in one contig, although two of them were considered partial sequence.

The intersection between the set of 16 BACs sequenced using the Illumina instrument and the set of 52 BACs sequenced using Sanger is a set of seven BACs (highlighted in bold in Table 8), but one of these seven BACs is not full-length (052L22). We used the six full-length Sanger-based BAC assemblies as the 'ground truth' to assess the quality of the assemblies from Illumina read at increasing depth of sequencing. To this end, we generated datasets corresponding to 100, 250, 500, 1000, 2000, 3500, 5000, 6000, 7000 and 8000X depth of sequencing (for each of the six BACs), by sampling uniformly short reads from the high-depth datasets. For each choice of the depth of sequencing, we generated 20 different datasets, for a total of 1,200 datasets. We assembled the reads on each dataset with VELVET v1.2.09 (with hash value k=79 to minimize the probability of false overlaps) and collected statistics for the resulting assemblies. Figure 6 shows the value of n50 (A), the size of the largest contig (B), the percentage of the target BAC not available in the assembly (C) and number of assembly errors (D) for increasing depth of sequencing. Each point in the graph is the average over the 20 datasets, and error bars indicate the SD. In order to compute the number of assembly errors we used the tool developed for the GAGE competition [18]. According to GAGE, the number of assembly errors is defined as the number of locations with insertion/deletions of at least six nucleotides, plus the number of translocations and inversions.

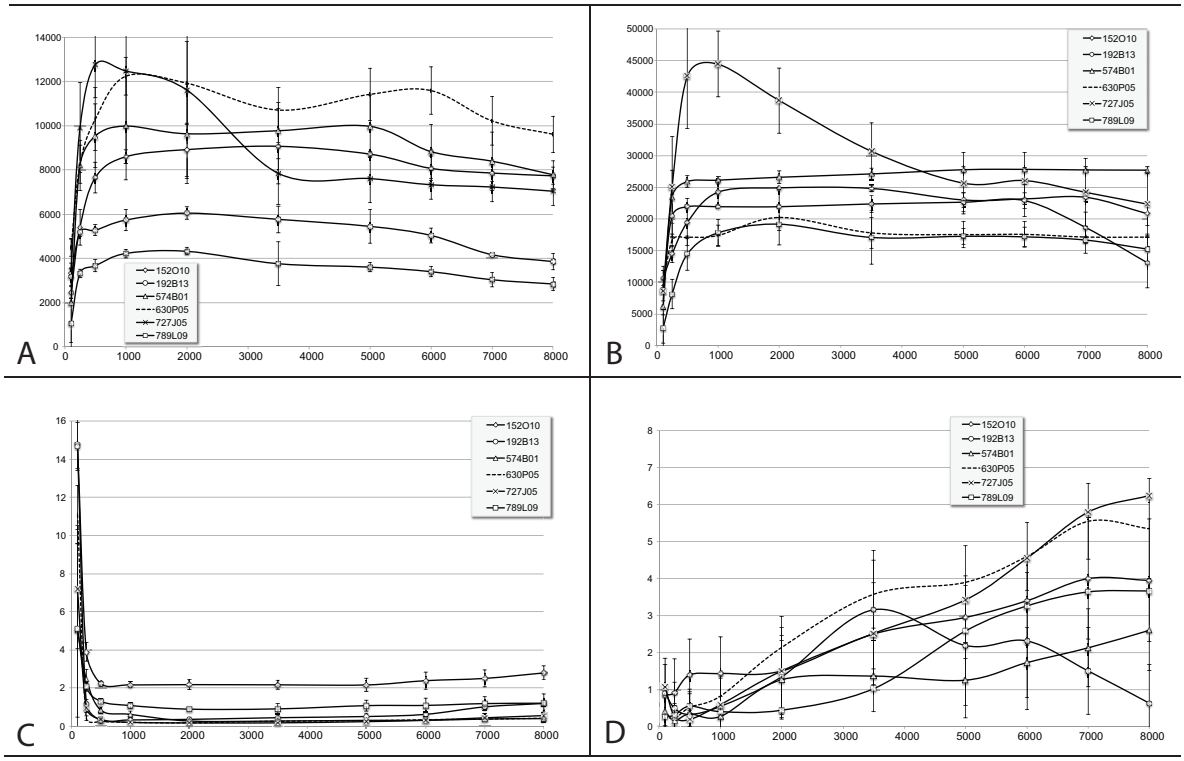

Figure 6. VELVET assembly statistics as a function of the depth of sequencing coverage: (A) n50, (B) longest contig, (C) percentage of the target BAC not covered by the assembly, (D) number of assembly errors; each point is an average over 20 samples of the reads, errors bars indicate standard deviation among the samples

A few observations on Figure 3 are in order. First, note that both the n50 and the size of the longest contig reach a maximum in the 500X – 2,000X range, depending on the BAC. Also observe that in order to minimize the percentage of BAC missed by the assembly one needs to keep the depth of sequencing below 2,500X (too much depth decreases the coverage of the target). Finally, it is very clear from (D) that as the depth of sequencing increases so do the number of assembly errors (with the exception of one BAC).

We have also investigated whether similar observations could be drawn for other assemblers. In Figure 11, we report the same assembly statistics, namely (A) the value of

n50, (B) the size of the largest contig, (C) the percentage of the target BAC not available in the assembly and (D) number of assembly errors for increasing depth of sequencing for one of the BACs. This time we used three assemblers, namely VELVET, SPADES v3.1.1 [16] and IDBA-UD [17] (statistics for all BACs are available in Figure 7 to Figure 11). Although there are performance differences among the three assemblers, the common trend is that as the coverage increases, the n50 and the size of the largest contig decreases, while the percentage of the BAC missing and the number of assembly errors increases. Among the three assemblers, SPADES appears to be less affected by high coverage. SPADES was run with hash values k=25, 45, 65 and option careful (other parameters were default). IDBA-UD was run with hash values k=25, 45, 65 (other parameters were default). The reported assembly is the one chosen by IDBA-UD.

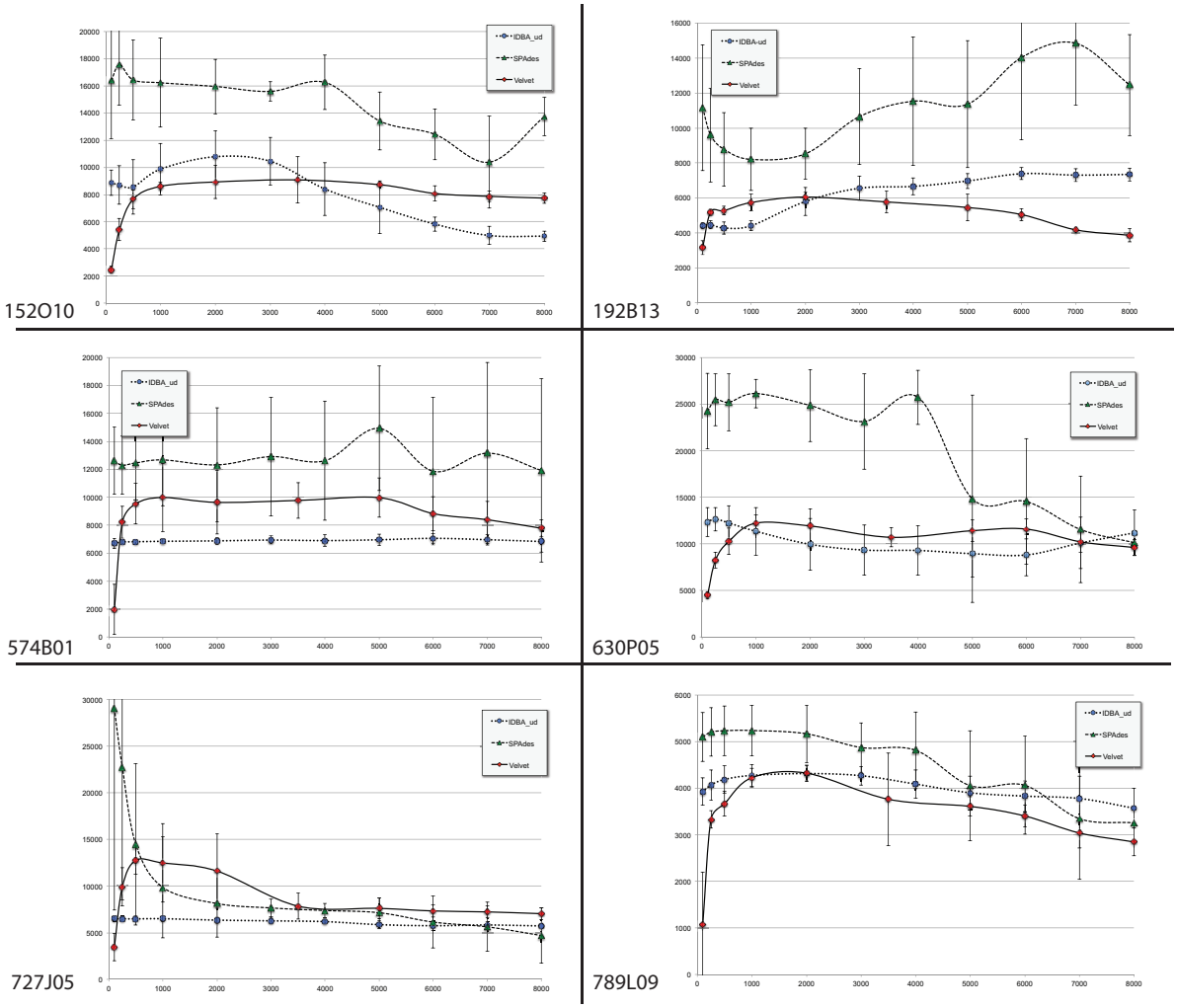

Figure 7. n50 statistics (Y-axis) for the six ultra-deep coverage BACs, assembled with VELVET, SPADES and IDBA-UD for various levels of depth of sequencing (X-axis)

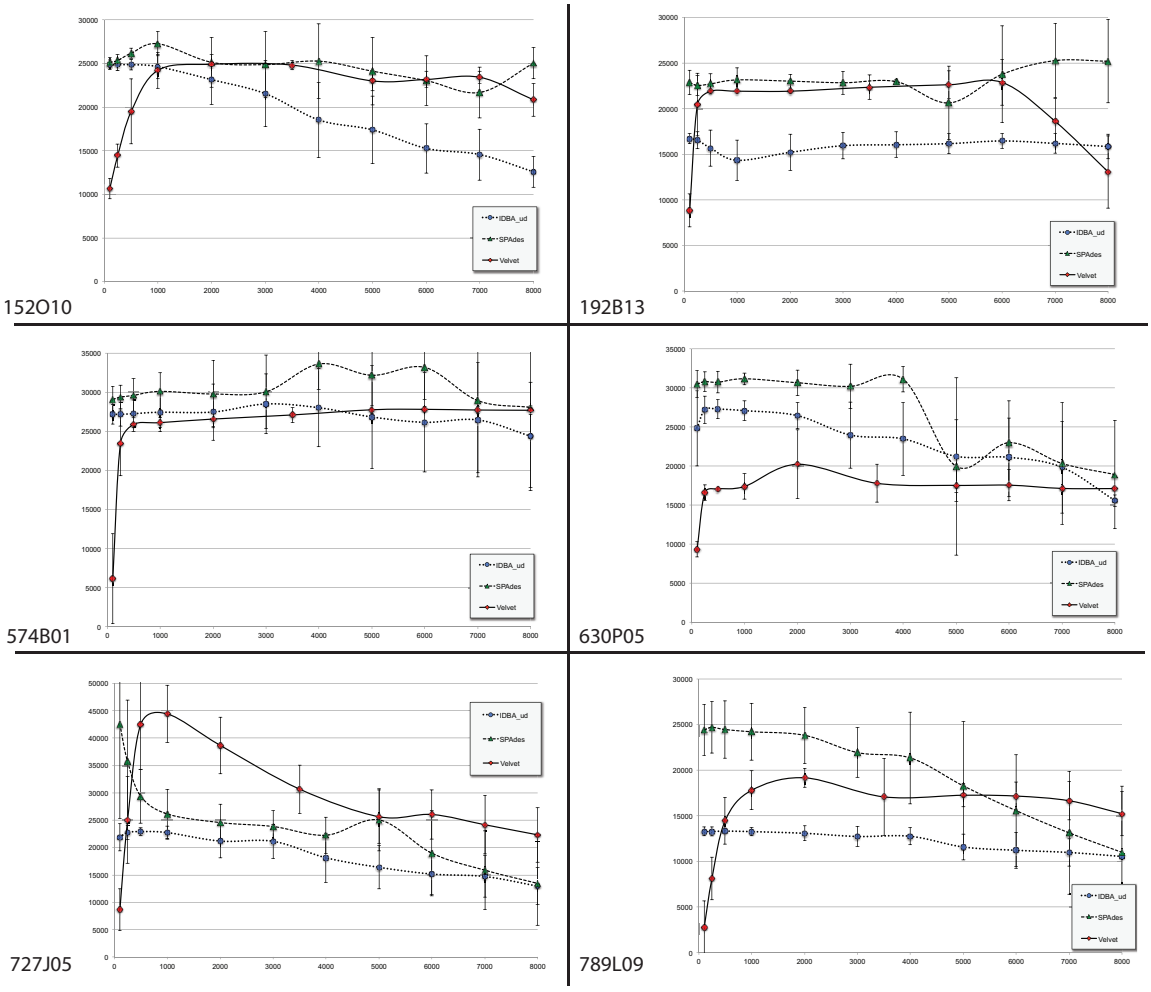

Figure 8. Largest contig statistics (Y-axis) for the six ultra-deep coverage BACs, assembled with VELVET, SPADES and IDBA-UD for various levels of depth of sequencing (X-axis)

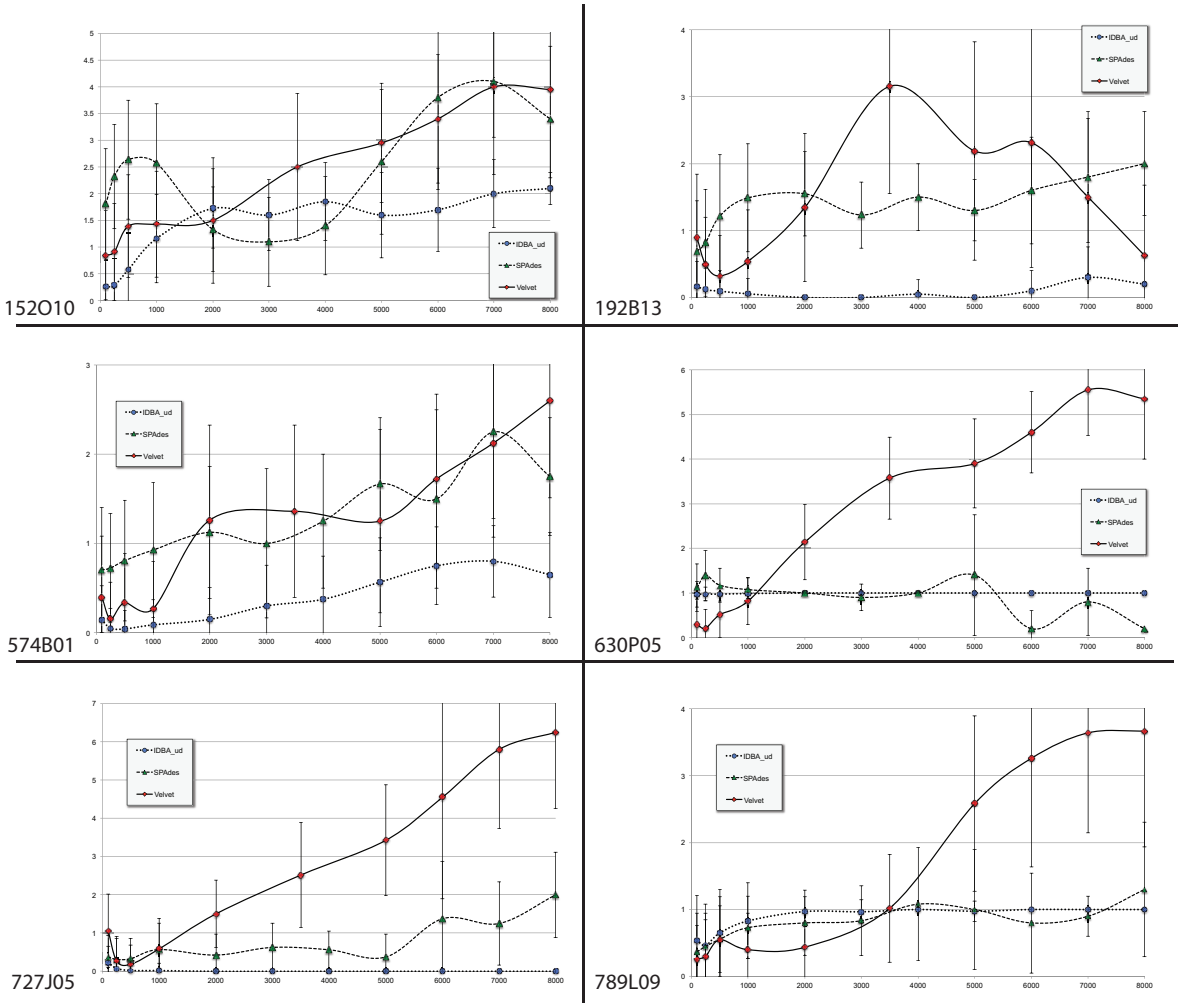

Figure 9. Mis-assembly error statistics (Y-axis) for the six ultra-deep coverage BACs, assembled with VELVET, SPADES and IDBA-UD for various levels of depth of sequencing (X-axis)

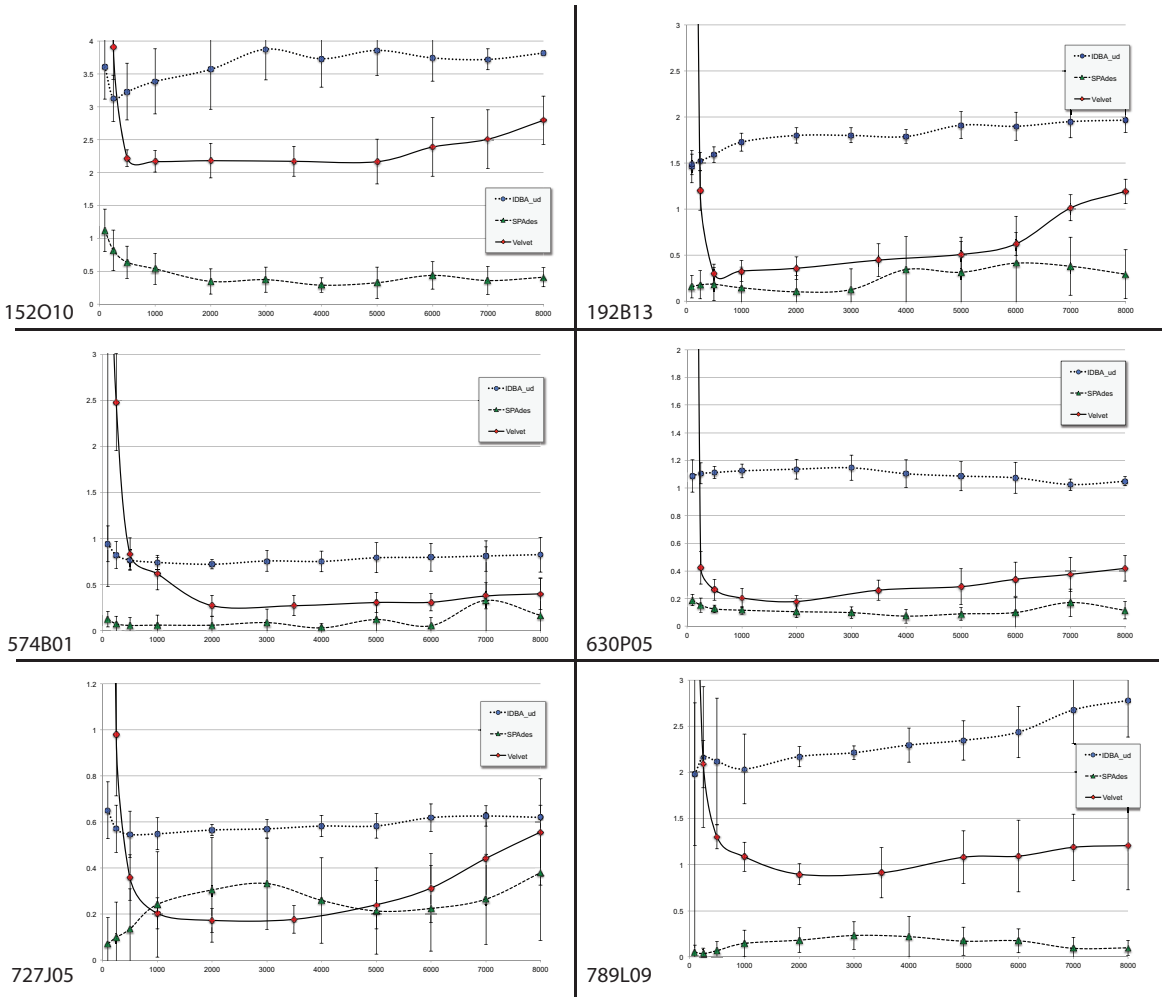

Figure 10. Genome percentage missing (Y-axis) for the six ultra-deep coverage BACs, assembled with VELVET, SPADES and IDBA-UD for various levels of depth of sequencing (X-axis)

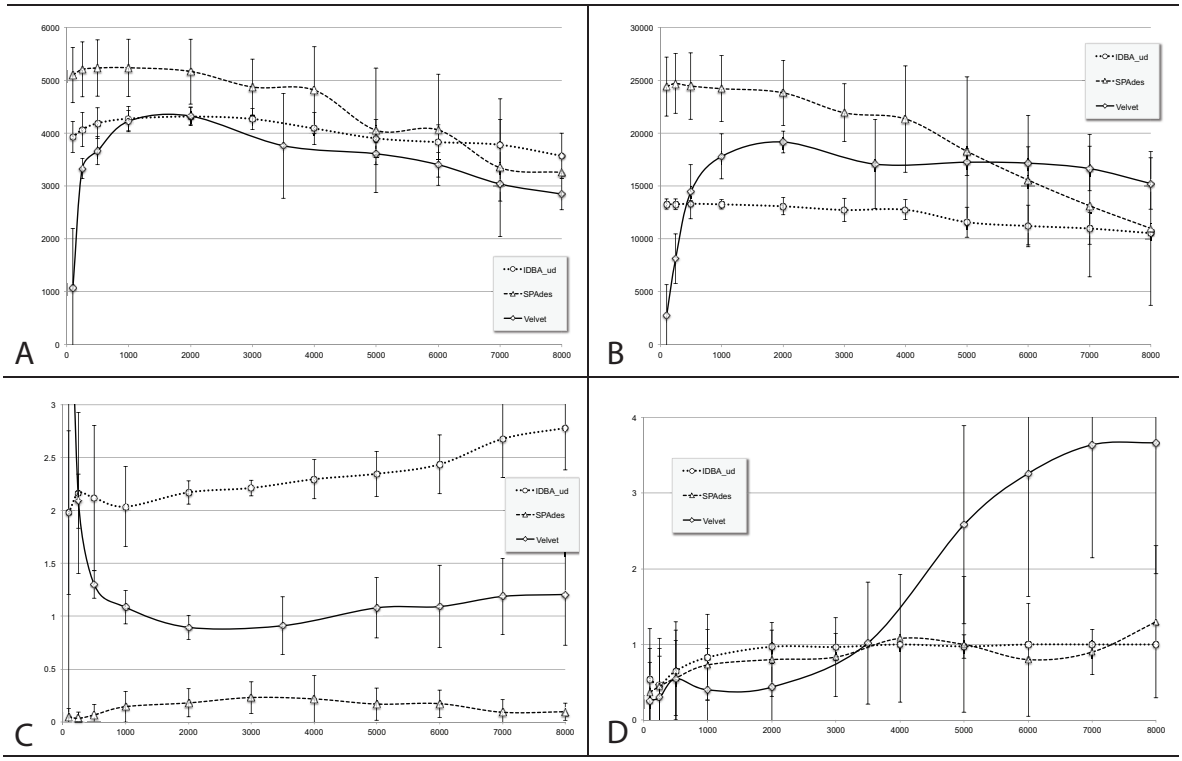

Figure 11. Assembly statistics as a function of the depth of sequencing coverage for BAC 789L09 for three assemblers: VELVET, SPADES and IDBA_UD; (A) n50, (B) longest contig, (C) percentage of the target BAC not covered by the assembly, (D) number of assembly errors; each point is an average over 10 subsamples of the reads, errors bars indicate standard deviation among the samples

Independently from us, the authors of (Desai et al., 2013) made similar observations on assembly degratadation. In their study, the authors assembled *E.coli* (4.6 MB), *Saccharomyces kudriavzevii* (11.18 MB) and *Caenorhabditis. elegans* (100 MB) using SOAPDENOVO, VELVET, ABYSS, MERACULOUS and IDBA-UD at increasing sequencing depths up to 200X. Their analysis showed that the optimum-sequencing depth for assembling these genomes is about 100X, depending on the specific genome and assembler.

Finally, we analyzed the performance of IDBA-UD, SPADES and VELVET on simulated reads. We generated 100 bp×2 paired-end reads from the Sanger assembly of

BAC 574B01 using the read simulator WGSIM (github.com/lh3/wgsim) at 100, 250, 500, 1000, 2000, 3500, 5000, 6000, 7000 and 8000X depth of sequencing. Insert length was 250 bp, with a standard deviation of 10 bp. For each depth of sequencing, we generated simulated reads at 0, 0.5, 1 and 2% sequencing error rate (substitutions). Insertions and deletions were not allowed.

IDBA-UD was executed with hash values k=25, 45, 65 (other parameters were default). VELVET was run with k=49. We repeated the simulations 20 times for IDBA-UD and 10 times for VELVET and SPADES. In Figure 12 to Figure 14, we report the usual assembly statistics, namely n50, largest contig, percentage missing, and number of assembly errors for VELVET, IDBA-UD and SPADES on these datasets. Observe that with 'perfect' reads (0% error rate), ultra-deep coverage does not affect the performance of IDBA-UD and VELVET. With higher and higher sequencing errors, however, similar behaviors to the assembly of real data can be observed for IDBA-UD and VELVET: n50 and longest contig rapidly decrease, and missing portions of the BAC and number of mis-assemblies increase. Surprisingly, SPADES seems to be immune to higher sequencing error rates.

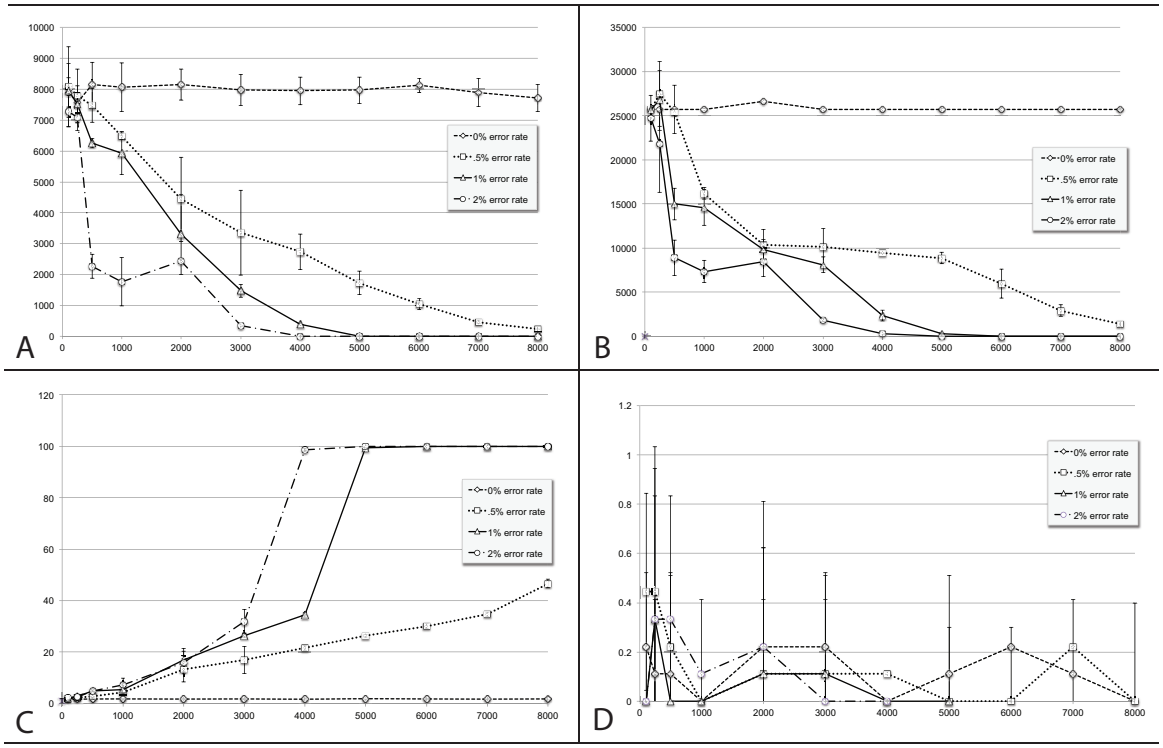

Figure 12. VELVET assembly statistics (Y-axis) as a function of the depth of sequencing coverage (X-axis) for synthetic reads generated from BAC 574B01 for several choices of the sequencing error rate: (A) n50, (B) longest contig, (C) percentage of the target BAC not covered by the assembly, (D) number of assembly errors; each point is an average over twenty samples of the reads, errors bars indicate standard deviation among the samples

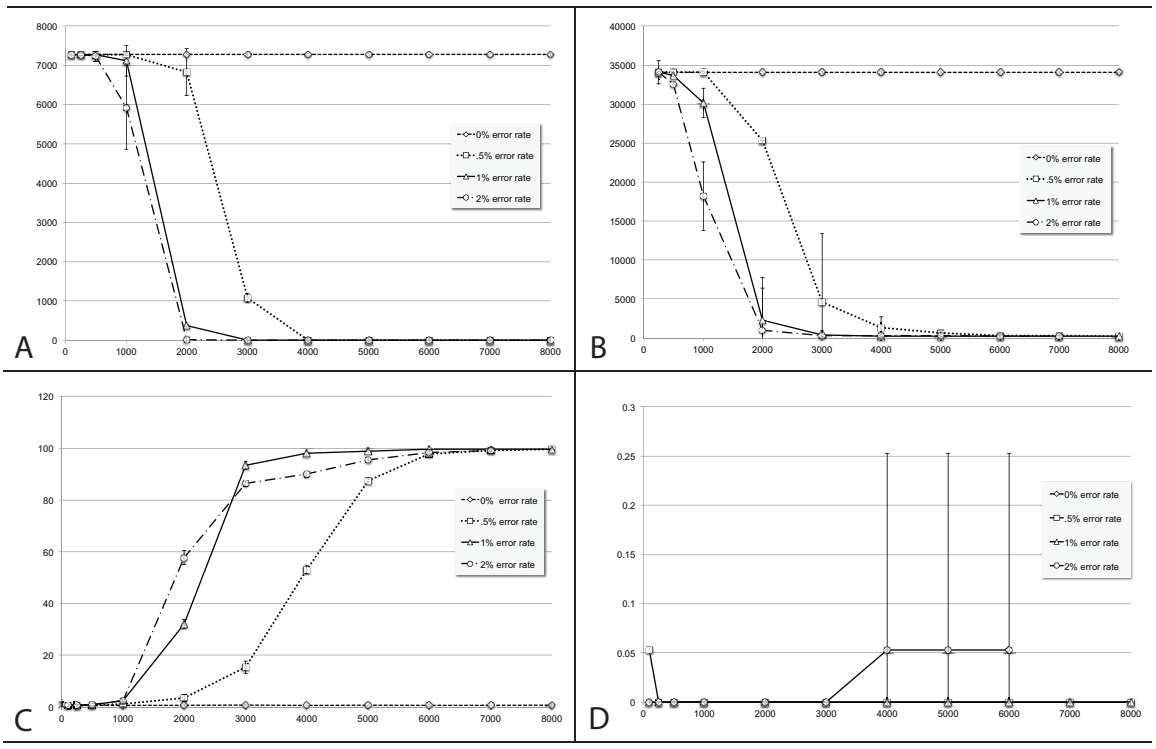

Figure 13. IDBA-UD assembly statistics (Y-axis) as a function of the depth of sequencing coverage (X-axis) for synthetic reads generated from BAC 574B01 for several choices of the sequencing error rate: (A) n50, (B) longest contig, (C) percentage of the target BAC not covered by the assembly, (D) number of assembly errors; each point is an average over twenty samples of the reads, errors bars indicate standard deviation among the samples

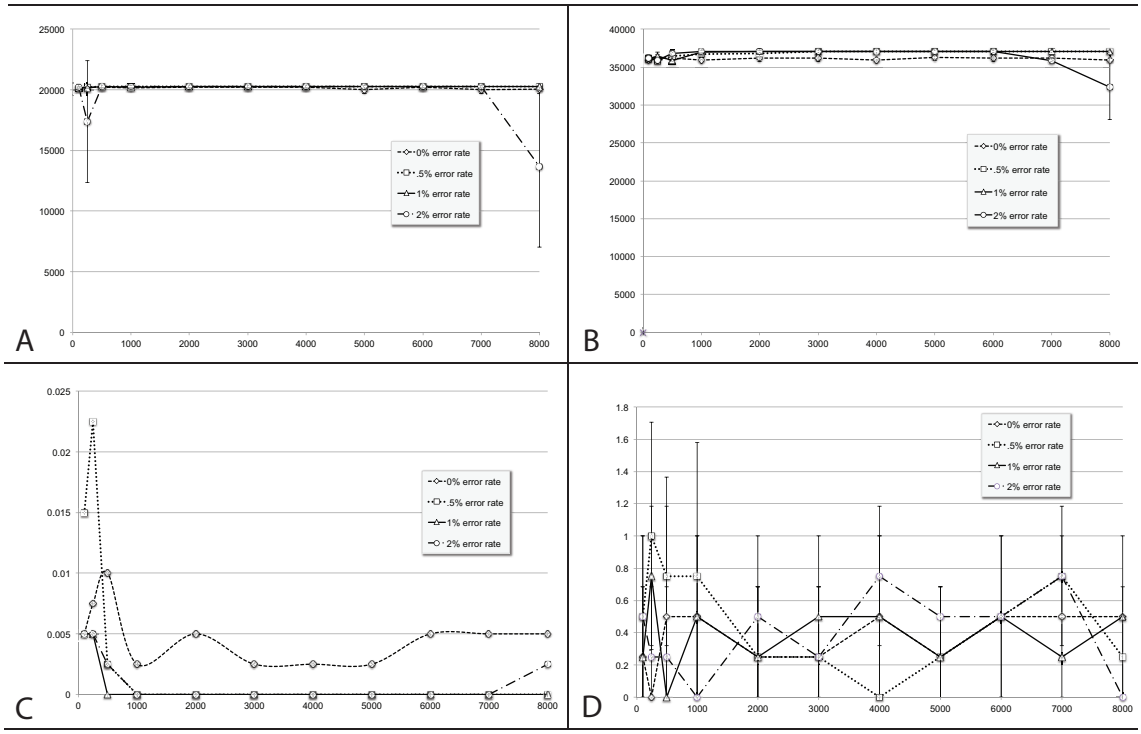

Figure 14. SPADES assembly statistics (Y-axis) as a function of the depth of sequencing coverage (X-axis) for synthetic reads generated from BAC 574B01 for several choices of the sequencing error rate: (A) n50, (B) longest contig, (C) percentage of the target BAC not covered by the assembly, (D) number of assembly errors; each point is an average over twenty samples of the reads, errors bars indicate standard deviation among the samples

2.3 Discussion

Because the introduction of DNA sequencing in the 70s, scientists had to come up with clever solutions to deal with the problem of *de novo* genome assembly with limited depth of sequencing. As the cost of sequencing keeps decreasing, one can expect that computational biologists will have to deal with the opposite problem: excessive amount of sequencing data. The Lander-Waterman-Roach theory [19, 20] has been the theoretical foundation to estimate gap and contig lengths as a function of the depth of sequencing.

We do not have a theory that would explain why the quality of the assembly starts degrading when the depth is too high. Possible factors include the presence (in real data) of chimeric reads, sequencing errors, and read duplications, or their combination thereof. In this study, we report on the *de novo* assembly of BAC clones, which are relatively short DNA fragments (100–150 kbp). With current sequencing technology it is very easy to reach depth of sequencing in the range of 1000–10,000X and study how the assembly quality changes as the amount of sequencing data increases. Our experiments show that when the depth of sequencing exceeds a threshold the overall quality of the assembly starts degrading (Figure 6). This appears to be a common problem for several *de novo* assemblers (Figure 11). The same behavior is observed for the problem of We have also investigated decoding reads to their source BAC (Figure 3), which is the main focus of this article. The important question is how to deal with the problem of excessive sequencing depth. For the decoding problem we have presented an effective 'divide and conquer' solution: we 'slice' the data in subsamples, decode each slice independently, then merge the results. In order to handle conflicts in the BAC assignments (i.e. reads that appear in multiple slices that are decoded to different sets of BACs), we devised a simple set of voting rules. The question that is still open is what to do for the assembly problem: one could assemble slices of the data independently, but it is not clear how to merge the resulting assemblies. In general, we believe that the problem of *de novo* sequence assembly must be revisited from the ground up under the assumption of ultra-deep coverage. We discuss the assembly problem for ultra-deep sequencing data in detail in chapter 3.

Chapter 3: *De Novo* Meta-Assembly of Ultra-deep Sequencing Data

As mentioned in Chapter 1, since the early days of DNA sequencing, the problem of *de novo* genome assembly has been characterized by insufficient and/or uneven depth of sequencing coverage (see, e.g., [21]). Insufficient sequencing coverage, along with other shortcomings of sequencing instruments (e.g., short read length and sequencing errors) exacerbated the algorithmic challenges in assembling large, complex genome – in particular those with high repetitive content. Some of the third generation of sequencing technology currently on the market, e.g., Pacific Biosciences [22] and Oxford Nanopore [23], offers very long reads at a higher cost per base, but sequencing error rate is much higher. As a consequence, long reads are more commonly used for scaffolding contigs created from second generation data, rather than for *de novo* assembly [24].

Thanks to continuous improvements in sequencing technologies, life scientists can now easily sequence DNA at depth of sequencing coverage in excess of 1,000x, especially for smaller genomes like viruses, bacteria or BAC/YAC clones. “Ultra-deep” sequencing (i.e., 1,000x or higher) has already been used in the literature for detecting rare DNA variants including mutations causing cancer [25, 26], to study viruses [27, 28], as well as other applications [21]. As it becomes more and more common, ultra-deep sequencing data is expected to create new algorithmic challenges in the analysis pipeline. In this chapter, we focus on one of these challenges, namely the problem of *de novo* assembly. We showed in chapter 2 that modern *de novo* assemblers SPADES [16],

IDBA_UD [17], and VELVET [14] are unable to take advantage of ultra-deep coverage [29]. Even more surprising was the finding that the assembly quality produced by these assemblers starts degrading when the sequencing depth exceeds 500x-1,000x (depending on the assembler and the sequencing error rate). By means of simulations on synthetic reads we also showed in [29] that the likely culprit is the presence of sequencing errors: the assembly quality degradation cannot be observed with error-free reads, while higher sequencing error rate intensifies the problem. The "message" of our study [29] is that when the data is noisy, more data is not necessarily better. Rather, there is an error-rate-dependent optimum.

Independently from us, study [30] reached similar conclusions: the authors assembled *E. coli* (4.6 MB) *S. kudriavzevii* (11.18 MB) and *C. elegans* (100 MB) using SOAPDENOVO, VELVET, ABySS, MERACOLOUS and IDBA_UD at increasing sequencing depths up to 200x (which is not ultra-deep according to our definition). Their analysis showed an optimum sequencing depth (around 100x) for assembling these genomes, which depends on the specific genome and the assembler.

In addition to sequencing errors, real sequencing data is also plagued by read duplications that contribute to uneven coverage. Read duplication is typically attributed to PCR amplification bias [31, 32]. The presence of highly duplicated reads complicates the task for assemblers when they contain sequencing errors; if unique it would be easy to detect and remove them. As the coverage increases, the probability of an overlap that involves duplicated reads agreeing to each other due to sequencing errors becomes higher and higher. These new overlaps can induce spurious contigs (typically short) or prevent

the creation of longer contigs. In turns, this manifests in a degradation of the assembly quality (N50, number of mis-assemblies, portion of the target genome covered, etc.) We also suspect that the removal of bubbles/bulges from the de Bruijn graph (for details on bubbles/bulges see, e.g[14] or [16]) is significantly harder with ultra-deep sequencing data.

Since sequencing errors are the source of the problem, one could attempt to correct them before the assembly. Several stand-alone methods have been proposed in the literature (see [33] for a recent survey), and several *de novo* assemblers (e.g., SPADES [16]) employ a preprocessing step for correcting errors. Unfortunately, error correction is not very effective for ultra-deep sequencing data. Most error correction tools are based on *k*-mer spectrum analysis: the underlying assumption is that "rare" *k*-mers are likely to contain sequencing errors. As the depth of sequencing of cover-age increases, so does the number of occurrences of any *k*-mer, including the ones that contain sequencing errors. In [29] and the current manuscript, we have collected experimental evidence of the inefficacy of error-correction methods on the assembly of ultra-deep sequencing data.

An alternative approach to deal with excessive sequencing data is down-sampling. The idea of down-sampling is to disregard a fraction of the input reads, according to some predetermined strategy. The simplest approach is to randomly sample the input and only assemble a fraction of the reads. Although coverage reduction has been primarily used for unbalanced data [34], we have shown in [29] that in the presence of ultra-deep sequencing data, the assembly of a random sample of the input reads only marginally improves the assembly quality compared to the assembly of entire dataset. DIGINORM

[34] and NEATFREQ [35] are two examples of down-sampling methods aimed to produce a more uniform cover-age. They both reduce coverage by selecting representative reads binned by their median *k*-mer frequency. In general, downsampling is not a satisfactory technique to deal with large datasets, unless it is expected to remove most of the "bad" reads and none of the "good" reads. Otherwise, it has the undesirable effect of re-moving "critical" reads, i.e., rare but error-free reads that can help bridge or fill assembly gaps.

In this chapter we address the question of how to create high quality assemblies when an ultra-deep dataset is available. We propose a meta-assembly method called SLICEMBLER that, unlike down-sampling techniques, takes the advantage of the whole input dataset. SLICEMBLER uses a divide-and-conquer approach: it "slices" a large input into smaller sets of reads, assembles each set individually (using a standard assembler), and then merges the individual assemblies. Our experimental results on real and synthetic data shows that SLICEMBLER can produce higher quality assemblies than the regular assembly of entire dataset (before or after error correction), as well as better assemblies compared to the assembly of random samples of the reads. The assemblies produced by SLICEMBLER demonstrate that, when an ultra-deep coverage dataset is available, it is possible to create long contigs with no assembly errors. We believe these results can be considered the first step toward making "perfect assemblies". We also show that SLICEMBLER is less sensitive to sequencing error rates, which could make it desirable for third-generation sequencing data.

3.1 Methods

The availability of ultra-deep sequencing data opens the opportunity to construct assemblies from multiple independent samples of the reads and then compare them with the objective either to [21] merge them or [22] discover assembly errors and correct them. SLICEMBLER is based on majority voting: if a contig (or a fraction thereof) appears in the majority of the individual assemblies, we assume that it is safe to add that contig to the consensus assembly being built. SLICEMBLER is a meta-assembler for second-generation paired-end short reads, but its framework can be adapted to other type of sequencing data.

Figure 15 illustrates the proposed iterative algorithm. First, SLICEMBLER partitions the reads into several smaller sets (slices). In the second step, it assembles each set individually using a standard assembler (e.g., VELVET, SPADES, IDBA_UD or RAY). Third, SLICEMBLER analyzes the individual assemblies, and identifies long common contigs (or fractions thereof) supported by a majority of the assemblies. In the fourth step, it merges these common contigs (or fractions thereof) to the partially constructed (consensus) assembly being built. Before repeating steps 2, 3 and 4, any read that maps to the consensus assembly is removed from the input.

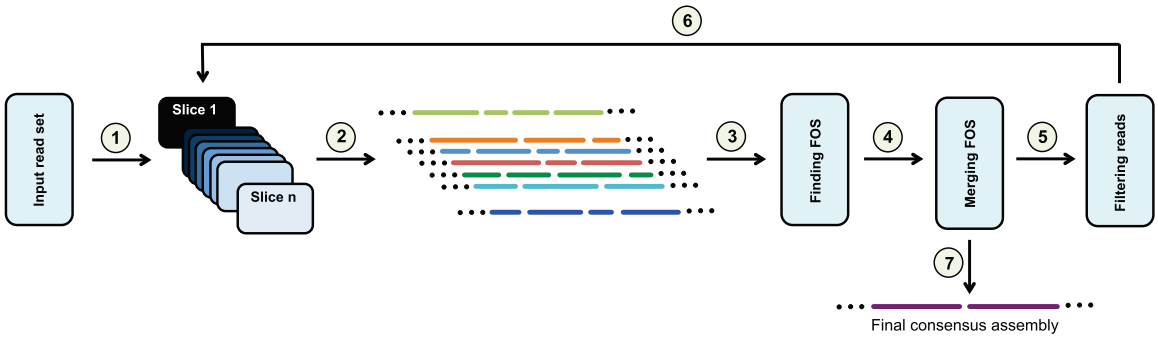

Figure 15. SLICEMBLER's pipeline: First, the input reads are partitioned into smaller slices (1). Each slice is assembled individually (2), and the resulting assemblies are merged by a "majority voting" process (3,4). Before repeating these steps any read in the input that maps to the consensus assembly is removed (6). When no further merging is possible, the final consensus assembly is produced (7).

3.1.1. "Slicing" the input

In the first step, the set of input reads is partitioned into n distinct slices. Each paired-end read is assigned to exactly one slice, although it is also possible to assign a read to multiple slices. For simplicity, each slice contains approximately the same number of reads. The number of slices is determined from the desired depth of coverage Ds for each slice. As we discussed in [29], the coverage Ds is a critical parameter for the quality of assembly. In order to find a good value for Ds, one can run the base assembler (e.g., VELVET, SPADES, RAY, or IDBA_UD) on larger and larger samples of the input and find the coverage that maximizes the chosen assembly statistics (e.g., N50). Once the value of Ds is established, one can determine the number of slices by computing n= ⌈Dt/Ds⌉ where Dt is the depth of coverage for the whole input read set. Given the set of input reads, the slice coverage Ds and the average read length, it is straightforward to partition the reads into *n* slices with the desired coverage.

3.1.2. Assembling the slices

In the second step, each of the *n* slices is assembled independently with a standard assembler (e.g., VELVET, SPADES, RAY, or IDBA_UD), possibly with different choices of the *k*-mer values in each slice. Under the assumption that the number of reads in every slice is sufficient for a complete assembly, the ideal outcome is that each of the n assemblies covers the entire the target genome. In practice, each assembly is expected to contain a mixture of "good" and "bad" contigs due to sequencing errors, repetitive regions and imperfections in the assembly algorithms. The objective of the next step is to identify the "good portion" of each contig by taking a majority vote among the assemblies.

3.1.3. Finding frequently occurring substrings

In the third step, SLICEMBLER searches for long substrings that occur exactly in the majority of individual assemblies. The input to this step is a set of *n* assemblies $S=\{A_1,A_2,\ldots,A_n\}$ where each assembly A_i is represented as a set of contigs. Given a string *s*, we define *c(s)* as a subset $T \subseteq S$ of assemblies in which s appears exactly in at least one contig of each assembly in *T*. Given a minimum support *k* and minimum length *l*, SLICEMBLER identifies all maximal substrings *r* such that $|r| > l$ and $|c(r)| > k$, that is, *r* is longer than *l* nucleotides and it appears in at least *k* assemblies. By maximal we mean that if string *r* was extended by one extra symbol to the left or to the right, then $|c(r)|$ would decrease below threshold $k+1$. We call such substrings *r*, frequently occurring substrings (FOS). Figure 16 illustrates four FOS detected from a set of five assemblies.

FOS1 occurs in four assemblies, while FOS2 appears in three of them. FOS3 and FOS4 is a pair of overlapping substrings occurring in three assemblies.

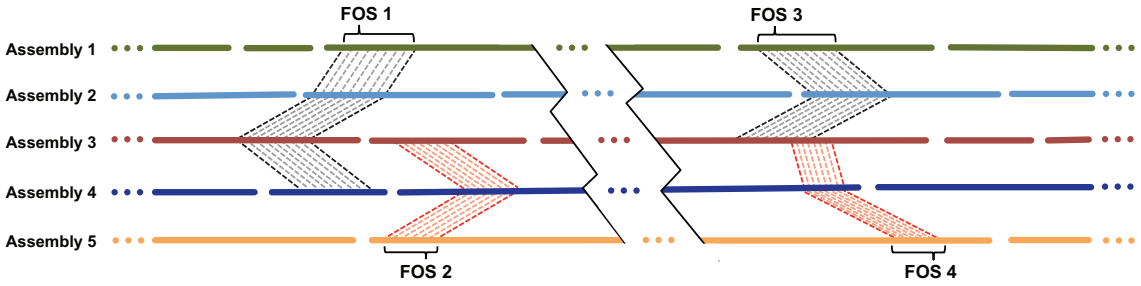

Figure 16. Examples of *frequently occurring substrings* (FOS) from five assemblies (FOS can overlap).

In order to find FOS, we build a generalized suffix tree on the contigs of n assemblies (and their reverse complement), then use a variant of the algorithm proposed in [36]. In this algorithm, each input string is assigned a distinct "color". The algorithm uses the generalized suffix tree to compute for each tree node u the number of distinct colors in the subtree rooted at node u. The algorithm computes the number of colors for each node in linear time in the length of the input strings. Algorithm [36], however, does not produce maximal substrings. Once the internal nodes have the color information, to ensure right-maximality our algorithm finds the deepest internal node u (spelling out string r, $|r|>1$) such that $|c(r)| > k$. To guarantee left-maximality we take advantage of suffix links: if node u has a suffix link to node v, and subtrees rooted at u and v have the same number of leaves and colors then the string corresponding to v is not left-maximal and should not be reported.

As we mentioned above, repetitive regions in the genome represent a major challenge for assemblers. Often a FOS includes a repetitive pattern at the end due to disagreements among assemblies on how many times that pattern should be repeated. The ends of each FOS are critical for merging, which requires a prefix-suffix overlap. Any error in these sections may prevent the algorithm from merging overlapping FOS (discussed next in Section 2.4). To avoid errors at the ends of a FOS, if a repetitive pattern is found at any of the ends, all copies (except one) of the repeated pattern are eliminated.

3.1.4. Merging frequently occurring sequences

When detected FOS are overlapping (e.g., FOS3 and FOS4 in Figure 16) they can be merged to obtain longer FOS (FOS will also be merged to the contigs in the consensus assembly being built). SLICEMBLER identifies any FOS that has an exact suffix-prefix overlap (i.e., no mismatches/indels) with another FOS (or its reverse complement), and determines the number of paired-end reads that connect each pair of such overlapping FOS. A pair of FOS is merged if either (1) the exact overlap is at least 100bp or (2) the exact overlap is 50-99bp and the number of paired end reads connecting them is at least Dt/1000 or (3) the exact overlap is 20-49bp and the number of paired end reads connecting them is at least Dt/100. This idea of using paired-end read to increase the confidence of an overlap is similar to the scaffolding step used to order and orient contigs in *de novo* assemblers or specialized scaffolding tools like [37].

3.1.5. SLICEMBLER algorithm

As said, SLICEMBLER is an iterative meta-assembler. The main steps of slicing/assembling/merging are executed iteratively until a predetermined condition is met. Table 9 shows a sketch of our algorithm. As described in Section 2.1, the number of slices is calculated from the chosen slice coverage (D_S). The input read set is partitioned into *n* slices (line 2). The rest of the algorithm is performed iteratively (line 3-19) until the total length of the consensus assembly *F* meets or exceeds the target genome size no sufficiently long FOS can be found. At the beginning of a new iteration, SLICEMBLER assembles the reads in each slice individually (lines 4-6). Next, a generalized suffix tree *T* is created from the contigs in the individual assemblies (both forward and reverse complement) (line 7). Using the suffix tree, SLICEMBLER produces the set of maximal substrings longer than *l* bases that occur in at least *k* assemblies (out of n, line 11). The FOS set could contain any number of strings (including none). Then, SLICEMBLER checks whether FOS overlapping with the current consensus assembly meet the conditions described in Section 2.4 and merges them (line 12). The parameter *k* is set to *n* initially, so SLICEMBLER first tries to determine if there is any FOS that appears in all the assemblies. If no new FOS is found, the support *k* is decreased (by one) and the loop is repeated. The parameter *k* is decreased until at least one FOS is detected or k becomes smaller than *n*/2. If k becomes smaller than *n*/2, the minimum length *l* is halved and *k* is initialized again to n. We selected *n*/2 as the "turning point" because we would not trust any common substring that appears in the minority of the assemblies. The initial value for *l* is one fifth of the size of the target; based on our observations using a larger value for

the initial value of l is unlikely to improve the results, but makes SLICEMBLER slower. The iterative process stops when l drops be-low l_{min}, which is desired minimum contig length in the final assembly (l_{min} is user-defined, typically 200-500 base pairs). If l is below l_{min} and no new FOS have been identified in the current iteration (line 17), SLICEMBLER's iterative process is terminated and the consensus assembly is reported.

Table 9. A sketch of SLICEMBLER's algorithm

Inputs	Input reads (S), slice coverage (D_S), min contig length (l_{min}), size of the target genome (l_{target})
Output	Set of contigs (F)

```
1   F ← ∅
2   Partition S into n slices S1,S2, ..., Sn each of which has coverage DS
3   while ( |F| < ltarget ) do
4         A ← ∅
5         for i ←1 to n do
6               A ← A ∪ Assemble(Si)
7         T ← GeneralizedSuffixTree(A, ReverseComplement(A))
8         k ← n
9         l ← ltarget / 5
10        while (l>lmin)
11              FOS ← FindFOS(T, k, l)
12              if (FOS ≠ ∅) then F ← MergeFOS(FOS, F)
13                                    break
14                              else if (k > n/2)  then k ← k – 1
15                                                 else l ← l/2
16                                                      k ← n
17        if (l <= lmin) and (FOS=∅) then break
18        for i ← 1 to n do
19              Si ← FindUnmappedReads(F, Si)
20  return F
```

Before starting a new iteration, all the reads in each slice are mapped to all detected FOS in the current consensus assembly. Each paired end read that maps exactly to any contig in the current assembly is removed (line 18-19) and only the remaining reads are assembled in the next iteration. Note that we do not repartition the read sets after this step, because although the number of reads de-creases, so does the size of the target we are supposed to reconstruct. In other words, the desired slice coverage Ds is maintained at every iteration. There is one exception to this strategy of read elimination. Recall that in order to be able to merge the FOS set with the current assembly, the strings have to overlap a minimum number of bases. To make sure that this will be possible in future iterations, reads that are mapped close to the ends of contigs of the current assembly are not eliminated.

Like any other assembly pipeline, gap filling and scaffolding can be applied at the end of the process to improve the quality of final assembly. In this case, gap filling is easier than usual because of the high quality of contigs produced by SLICEMBLER and the very large number of reads available for filling the gaps. Also, the number of gaps to be filled at the end is relatively small since SLICEMBLER fills some of the gaps during the merging process (see Figure 18 for an example). The merging step uses small FOS identified in the later iterations to "glue" adjacent contigs.

3.2 Experimental Results

We implemented SLICEMBLER in Python. Our tool can be accessed at http:// SLICEMBLER.cs.ucr.edu/. As said, SLICEMBLER is a meta-assembler; its performance

directly depends on the base assembler. In the following experiments we used VELVET as the base assembler, unless stated otherwise. The performance of SLICEMBLER using other base assemblers (IDBA_UD, RAY and SPADES) is presented in Section 3.3. The number of slices and the sequencing error rate for the input reads are other factors that critically influence the quality of the final assembly. We study these issues in Section 3.4 and Section 3.5.

Recall that at the end of every iteration, all reads are mapped to the partially constructed assembly in order to detect bridge reads (to be used later in the merging step) and to eliminate reads that are already represented in the assembly. SLICEMBLER uses BWA [38] to find perfect alignments (no mismatches, no gaps) for this purpose. We used a minimum contig length l_{min} = 200 (which is the default parameter for SLICEMBLER). We did not use any gap filling or scaffolding tool on the final assemblies. All experiments were carried on a Linux server with twenty computing cores and 194 GB of RAM.

3.2.1. Ultra-deep sequencing of barley BACs

In order to carry out experiments on real ultra-deep data, we sequenced a set of 16 bacterial artificial chromosome (BAC) genomic clones of barley (*Hordeum vulgare L.*) on an Illumina HiS-eq2000 at UC Riverside at a depth of coverage 8,000x-15,000x. The average read length was about 88 bases after quality trimming; reads were paired-end with an average insert size of 275 bases. Another set of 52 barley BACs was sequenced by the Department of Energy Joint Genome Institute (JGI) using Sanger sequencing. Since the primary DNA sequences for each of these 52 BACs were assembled in one solid contig (details in [29]), we assumed these Sanger-based assemblies to be the

"ground truth" or "reference". Five ultra-deep sequenced BACs had such a reference, so we used them to objectively evaluate the performance of SLICEMBLER. In order to have an equal-sized input dataset for all BACs, we used only 8,000x worth of coverage. These five barley BAC clones, hereafter referred as BAC 1-5 have the following lengths: 131,747bp, 108,261bp, 110,772bp, 111,748bp, and 102,968bp, respectively. We should remind the reader that the barley genome is highly repetitive. Approximately 84% of the genome consists of mobile elements or other repeat structures [15].

3.2.2. Quality of SLICEMBLER assemblies

SLICEMBLER divided each of the five ultra-deep BAC inputs in-to ten slices (Ds = 800x coverage). We showed in [29], that such coverage is expected to provide a "good" assembly in terms of N50, longest contig, number of mis-assemblies, and percentage of the target genome covered. We compared the performance of SLICEMBLER to three alternative methods, namely A) assemble all reads (8,000x coverage) with the same assembler used in SLICEMBLER, B) run error-correction (using RACER [39]) on all reads (8,000x coverage) then assemble the corrected reads with the same assembler used in SLICEMBLER, C) assemble each of the slices (800x coverage) individually and consider the average statistics over the ten slices (down-sampling).

Figure 17 summarizes the assembly statistics collected with QUAST [40] for SLICEMBLER compared to methods A, B, C described above. The base assembler was VELVET (hash value 69). Several observations on Figure 17 are in order. First, note that for most of the BACs, down-sampling at 800x leads to better quality assemblies than the assembly of all the reads at 8,000x. This is consistent with our previous results [29].

Second, error correction increases the quality of assemblies for most of the BACs. However, this step can also introduce additional assembly errors probably due to newly introduced errors in the error-correction phase (see last panel of Figure 17). We suspect that ultra-deep coverage in the input dataset prevents RACER from detecting and correcting sequencing errors effectively.

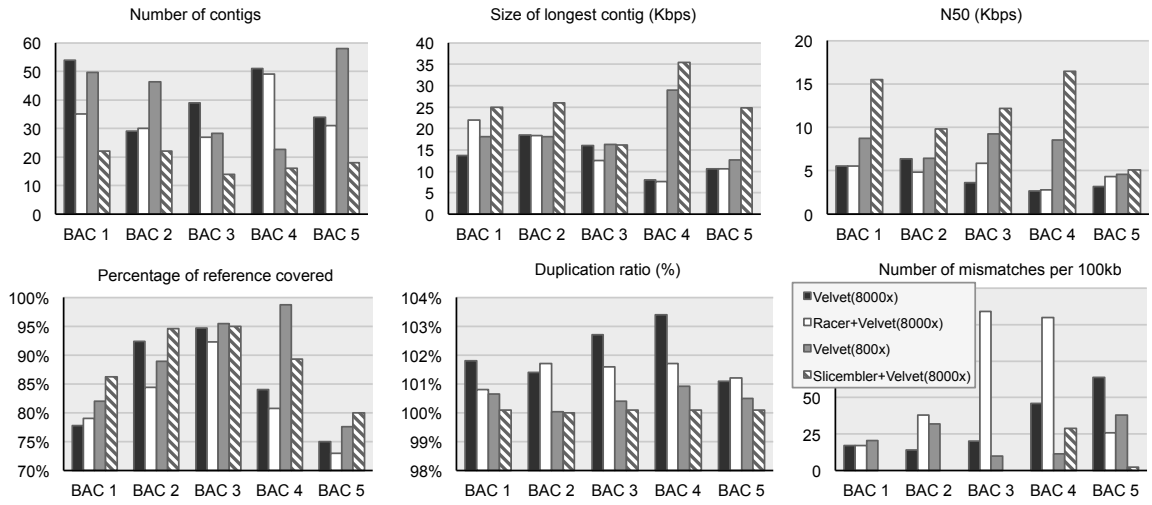

Figure 17. Summary of assembly statistics on five barley BACs sequenced at 8,000x. We compared SLICEMBLER (using VELVET) to three alternative methods: VELVET on the entire dataset, RACER+VELVET on the entire dataset, and the average performance of VELVET on the slices of 800x each (see legend). Ground truth was based on Sanger-based assemblies. Statistics were collected with QUAST for contigs longer than 500 bps.

Finally and more importantly, observe that in the majority of cases, SLICEMBLER generates the highest quality assemblies. Its assemblies are less fragmented, which is reflected by a smaller number of contigs, longer longest contigs, and higher N50. Also, SLICEMBLER's assemblies cover a higher fraction of the target genome and they have a much smaller number of mis-assembly errors compared to the other approaches. In fact SLICEMBLER's assemblies are almost error-free. BAC 4 is the only exception: although SLICEMBLER's assembly of BAC 4 has fewer mis-assemblies than the assembly of all the

reads before or after error correction, it contains more errors than the average downsampling-based assembly. The slightly higher number of assembly errors for SLICEMBLER is due to the merging step, which could be made more conservative. Finally, note that SLICEMBLER's assemblies are less inflated than the other approaches. The assembly of all the reads, with or without error correction, has quite large duplication ratio.

To illustrate the progress during SLICEMBLER's iterative refinements, Figure 18 shows the status of the consensus assembly created for BACs 1, 2 and 3 every five iterations. Each box represents a perfect alignment of a SLICEMBLER's contig to the reference genome (no insertion/deletion/mismatches allowed). Observe that in the last iteration 85%-95% of the target genome is covered by the error-free contigs. In the first iterations, most of the target ge-nome is covered by large FOS. In later iterations, FOS are smaller but they can connect adjacent contigs or extend them (see red circles). Most of the small gaps between the contigs are composed by repetitive patterns. These gaps are induced by the "trimming" step of the algorithm, which eliminates repetitive patterns from the ends of FOS to avoid false overlaps. A gap-filling tool can easily close these small gaps during the finishing step.

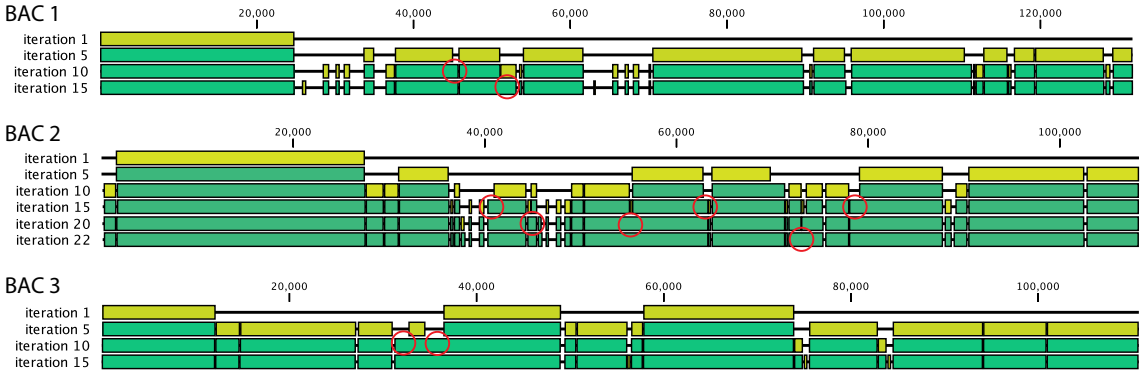

Figure 18. An illustration of SLICEMBLER's progressive construction of the consensus assembly for BACs 1, 2 and 3 ("snapshots" are taken every five iterations). Each box represents a perfect alignment between that contig and the reference. Light green boxes indicate a new FOS compared to the previous snapshot. Circles point to gaps closed or contig extended via the merging process (picture created with CLC sequence viewer).

As mentioned above, at the end of each iteration SLICEMBLER maps the current set of input reads to the consensus assembly: any read that is mapped exactly is discarded. This allows SLICEMBLER (and its base assembler) to "focus" on the parts of the genome/BAC that are still missing from the consensus assembly. Because FOS in early iterations are "safer" to be added to the consensus assembly, the set of reads discarded in early iterations are expected to be of higher quality. To this end, we determined the percentage of reads at each iteration of SLICEMBLER that could mapped exactly (i.e., no mismatches/indels) to the reference genome. Figure 19 shows these percentages for the first fifteen iterations in the assembly of the five BACs. Observe that the percent-age of high quality reads is about 85% in early iteration.

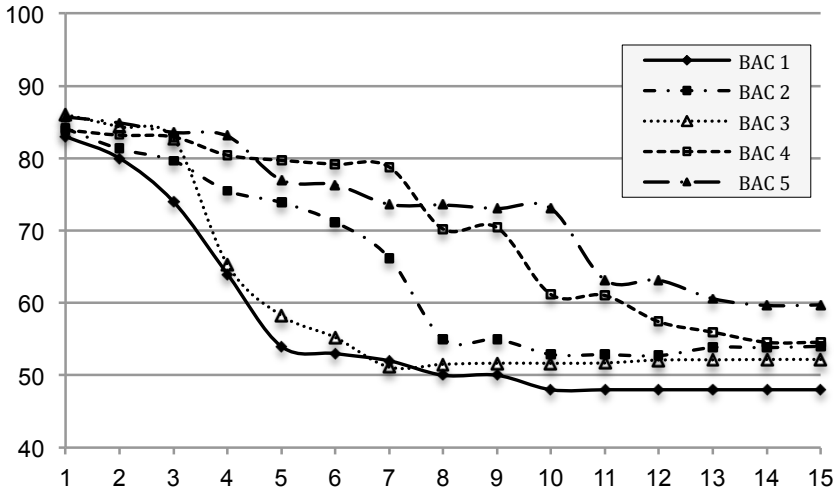

Figure 19. The percentage of reads (y-axis) at each iteration of SLICEMBLER (x-axis) that map exactly (i.e., zero mismatches/indels) to the reference on the five ultra-deep sequenced BACs.

As the number of iterations increases, the percentage of high quality reads in the input monotonically decreases. In the last few iterations the percentage stays somewhat flat because later FOS are shorter, so the additional number of high quality reads mapped to these FOS is also small.

3.2.3. The choice of the base assembler

As said, SLICEMBLER is a meta-assembler, and its performance depends on the performance on the base assembler. To evaluate the influence of base assembler on the assembly quality, we compared several assemblers, namely VELVET [14], SPADES [16], RAY [41] and IDBA_UD [17].

Experimental results for BAC 3 are shown below in Table 10. We compared the assembly produced by VELVET, SPADES, RAY and IDBA_UD all the reads (8,000x) against the assemblies created by SLICEMBLER in conjunction with the corresponding

base assembler. SLICEMBLER was run on ten slices (800x each). The k-mer used was 69 for VELVET and RAY. For IDBA_UD and SPADES the reported assembly was based on three different *k*-mers (29, 49 and 69).

Table 10. Comparing BAC assemblies produced with IDBA_UD, VELVET, SPADES and Ray to the assemblies produced by SLICEMBLER in conjunction with the same assembler. Statistics were collected with QUAST for contigs longer than 500 bps.

Method	Number of contigs	% ref covered	Duplication ratio	Mismatches per 100Kbp	N50	Longest contig
IDBA (8,000x)	34	97.0%	**1.010**	**0.93**	7,335	13,889
SLICEMBLER + IDBA (10 slices of 800x)	**13**	**97.0%**	**1.010**	1.1	**16,121**	**31,161**
Velvet (8,000x)	39	94.7%	1.027	20.0	3,649	16,048
SLICEMBLER + Velvet (10 slices of 800x)	**14**	**95.1%**	**1.001**	**0**	**12,178**	**16,128**
SPAdes (8,000x)	49	95.7%	**1.006**	**0.94**	9,129	21,872
SLICEMBLER + SPAdes (10 slices of 800x)	**11**	**96.9%**	1.024	1.2	**27,685**	**31,158**
Ray (8,000x)	35	80.0%	1.003	**0**	3,996	7,186
SLICEMBLER + Ray (10 slices of 800x)	**24**	**88.0%**	**1.000**	**0**	**7,192**	**12,842**

Observe that among the stand-alone assemblers, IDBA_UD and SPADES created higher quality assemblies compared to VELVET and RAY. However, regardless of the choice of the base assembler, SLICEMBLER improved the quality of the assemblies.

The only "negative" statistics for SLICEMBLER is that it introduced a few more errors in the assemblies created using IDBA_UD and SPADES. We determined that these additional errors were due to incorrect merging in later iterations. Also, SLICEMBLER had slightly higher duplication ratio than SPADES. Other than these, SLICEMBLER significantly improved all other statistics. In fact, similar results were observed on the

other four BACs (data not shown). In general, SLICEMBLER created higher quality assemblies when used in conjunction with IDBA_UD and SPADES.

3.2.4. The choice of depth of coverage for each slice

As said, the depth of coverage in each slice is critical to optimize on the quality of the assemblies. If the depth of coverage is too low, the assembly of each slice will be fragmented, which will be reflected in shorter FOS. On the other hand, more slices can increase the confidence in choosing FOS due to more "votes" available. For this reason, we decided to use simulations to study the tradeoffs of the depth of coverage in each slice. To this end, we used wgsim to generate synthetic datasets with 500x, 1,000x, 2,500x, 5,000x, 7,500x and 10,000x reads at 1% sequencing error rate (no indels) based on the reference sequence of BAC 3. Each dataset was assembled with SLICEMBLER using VELVET as the base assembler by dividing the input into ten slices, so that the coverage in each slide was 50x, 100x, 250x, 500x, 750x and 1,000x.

Table 11 shows the usual quality statistics for the assemblies on simulated reads. Observe that SLICEMBLER's best performance is observed when slices are in the 100x-500x coverage range. When the slice coverage is lower than 100x, assemblies are more fragmented due to insufficient coverage. When the slice coverage is higher than 500x, we experience the negative effects of ultra-deep sequencing data on the quality of the individual assemblies: FOS become smaller and the final assembly is more fragmented. Note that despite the 1% sequencing error rate, SLICEMBLER was able to create error free contigs for all cases.

Table 11. Quality statistics for SLICEMBLER's assemblies for simulated reads with different depth of coverage. We used ten slices in all experiments (i.e., the coverage for each slice was 50x, 100x, 250x, 500x, 750x, and 1,000x). Statistics were collected with QUAST for contigs longer than 500 bps.

	500x	1,000x	2,500x	5,000x	7,500x	10,000x
Number of contigs	20	12	11	**10**	18	38
Longest contig	27,364	31,823	31,946	**31,950**	21,865	9,425
N50	6,707	26,275	**26,288**	26,267	12,428	3,643
Percent Refer. Covered	90.6%	88.7%	**94%**	93.9%	92.9%	84.7%
Duplication ratio	1	1	1	1	1	1
Mismatches per 100kbp	0	0	0	0	0	0

3.2.5. Effect of sequencing error rate in the reads

De novo assemblers are quite sensitive to sequencing error rate in the input reads. Even assemblers that have a preprocessing step for error correction (e.g., SPADES), has difficulties handling errors when the depth of coverage is very high [29]. Since SLICEMBLER relies on majority voting for common contigs in the slice assemblies, we wondered whether it would be more resilient compared to its base assembler. To this end, we used wgsim to generate data sets at 3,000x coverage with increasing sequencing error rate, namely 0% (errorless), 0.5%, 1% and 2% error rate based on BAC 3. We assembled each set with SLICEMBLER + VELVET using six slices of 500x coverage each. Results are reported in Figure 20.

First, note that SLICEMBLER was not able to improve the quality of assembly when the reads are error-free. This is consistent with the results in [29] for error-free reads. VELVET and other *de novo* assemblers are capable of producing high quality assemblies when reads are error-free since there are no imperfections in the de Bruijn graph. More importantly, observe that as the sequencing error rate increases, the performance of

VELVET quickly degrades, while the performance of SLICEMBLER is unaffected (despite using VELVET as the base assembler). Particularly impressive is the number of mismatches per 100Kbp, which stays at zero for SLICEMBLER for any error rate.

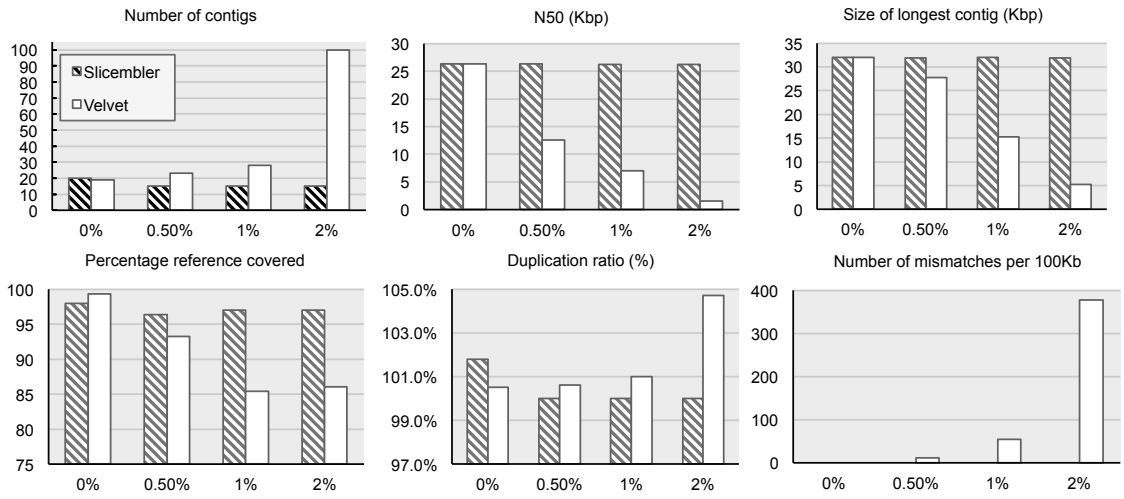

Figure 20. The effect of increasing sequencing error rates on the quality of assemblies created by VELVET and SLICEMBLER+VELVET. Input paired-end reads were generated using wgsim with a coverage of 3,000x using BAC 3 as a reference. For SLICEMBLER, simulated read sets were divided into six slices. Statistics were collected with QUAST for contigs longer than 500 bps.

3.3 Discussion

Advancement in sequencing technologies has been reducing sequencing costs exponentially fast. Ultra-deep sequencing is now feasible, especially for smaller genomes and clones. We expect that in the near future life scientists will sequence "as much as they want" because the sequencing cost will be a minor component of total project costs. This explosion of data will create new algorithmic challenges. We have shown previously that popular modern *de novo* assemblers are unable to take advantage of ultra-deep cover-

age, and the quality of assemblies starts degrading after a certain depth of coverage. SLICEMBLER is an iterative meta-assembler that solves this problem: it takes advantage of the whole dataset, and significantly improves the final quality of the assembly. The strength of SLICEMBLER is based on the majority voting scheme: frequently occurring substrings identified by SLICEMBLER in the slice assemblies never contain errors, with the exception of FOS belonging to the very ends of the target genome which are not as reliable because coverage tends to be lower. SLICEMBLER extracts high-quality contigs from the slice assemblies, and it prevents contigs containing mis-joins and calling errors to be included in the final assembly.

Experiments on a set of ultra-deep barley BACs and simulated data shows that our proposed method leads to higher quality assemblies than the corresponding base assembler. We also demonstrated that SLICEMBLER is more resilient to high sequencing error rates than its base assembler.

This iterative assembly approach can be adapted to solve other problems in this area, like assembly of single cell sequencing data. The quality of single cell assemblies usually suffers from biased coverage. It has been shown that the coverage in single-cell sequencing is distributed randomly [42]. To reduce the effect of uneven coverage, one can sequence multiple copy of a cell and assemble the merged sequencing data. In this case, regions with lower coverage in one sample can be covered sufficiently by the other samples. SLICEMBLER can be adapted to create high quality assemblies for multi-cell sequencing data. In this case, each of the sequenced cells can be considered as a “slice” for the SLICEMBLER algorithm.

Our proposed algorithm is expected to work for genomes of any length, but the current implementation of SLICEMBLER has been tested only on relatively small genomic target sequences for which real ultra-deep coverage is now available. In order for SLICEMBLER to scale to larger genomes its efficiency must be improved. Most of the computational effort in SLICEMBLER is spent in finding FOS (this required construction of the generalized suffix tree), merging FOS (this requires computing exact prefix-suffix overlaps) and mapping the reads (this requires running BWA) at every iteration. One way to increase the algorithm speed would to process the slices in parallel. Another possible improvement would be to map the reads to each slice assembly only once and process the alignment file to determine which reads should be passed to the following iteration, instead of mapping the reads to the slice assembly from scratch in every iteration. We are also working on improving the merging step, in order to prevent mis-joins. More advanced approaches for merging contigs, like methods proposed for merging draft assemblies [43-45], may improve the quality of SLICEMBLER results. We plan to release soon an improved version of SLICEMBLER implemented in C++.

To conclude, the results presented in this chapter indicate the possibility of having (almost) perfect assemblies when the depth of coverage is very high. Although there is more work to be done to achieve a perfect assembly, we believe that SLICEMBLER represents a significant step forward in this direction.

Chapter 4: Other projects

4.1 SNP detection and anchoring on cowpea genome

Cowpea (*Vigna unguiculata*) is the main source of protein for people living in Sub-Saharan Africa (SSA). This legume is native to Africa, but is grown in Asia, Latin America, and in the southern United States. There is a high collinearity between cowpea genome and its close relatives, soybean and common bean [46]. However, cowpea is more drought and heat tolerant than them. More countries around the word are facing drought due to the global warming, therefore this legume is extremely important for the future of human food, especially in developing countries. Although cowpea is an important source of food, researchers still suffer from the lack of a high quality published reference genome for this plant.

Cowpea has a diploid genome with size ~620 MB. It has 2×11 = 22 chromosomes. We maintain a rich and diverse germplasm at the UC Riverside campus, provided by three major germplasm collections in Africa [International Institute of Tropical Agriculture (IITA), Nigeria], the USDA repository in Griffin, Georgia, and the University of California (UC) cowpea germplasm collection.

Different cowpea cultivars show different levels of ability to survive drought stress[47], so the breeders can incorporate drought tolerance into improved varieties. Our cowpea team, including UC Riverside and a network of breeders in Burkina Faso, Ghana, Mozambique, Nigeria (IITA) and Senegal, has developed new resources for cowpea in

recent years. Finding new markers for the cowpea genome to assist breeding efforts has been a main goal for the team.

Quantitative trait locus (QTL) analysis is a statistical method that allows researchers to link certain complex phenotypes to specific regions of chromosomes. Many of the cowpea drought and heat resistance traits have been genetically mapped using QTL approaches [48]. A consensus genetic map positions of the more important QTLs for cowpea is available based on the 1536-SNP GoldenGate assay for genotyping. Our team has started introgressing some of these traits into breeding lines in the African partner countries. The relatively low SNP resolution of the trait determinant haplotypes is a challenge for the progress of the project.

To address this issue, we designed a pipeline to find around 60k high quality SNPs for cowpea genome at UC Riverside. Illumina designed an Infinium iSelect custom genotyping platform based on the discovered SNPs. The designed genotyping chip is available in the market for all researchers. The provided higher resolution SNP map for this genome can be used for several proposes including pedigree validation, germplasm characterization and marker-assisted breeding of cowpea. Also, our team is currently working on the problem of ordering and merging the sequenced cowpea BACs, using the discovered SNPs.

4.1.1. Cowpea reference genome

The African cowpea cultivar IT97K-499-35 has been attracted the most attention among the other cowpea accessions, mainly because it is resilient against the parasitic weed *Striga gesnerioides* (cowpea witchweed). There are multiple genome resources

available for this accession, mainly provided by the International Institute on Tropical Agriculture (IITA): gene-rich sequences accounting for ~160 Mb of the 620 Mb genome [49]; ~29,000 EST-derived "unigene" consensus sequences available at harvest-blast.org and in the HarvEST:Cowpea (harvest.ucr.edu) software that our cowpea team developed; a BAC-based physical map (http://phymap.ucdavis.edu/cowpea/; see "Prior Results") from which we have sequenced about 4,000 BACs; and an initial whole-genome shotgun assembly that contains sequences for about 97% of all known cowpea genes but has very limited contiguity.

We assembled a set of sequencing data available for IT97K-499-35 with SOAPdenovo. The input set for the assembly consisted of on ~60x Illumina GAII short reads, one 5 kb library, about 30,000 Sanger BAC-end sequences and about 250,000 Sanger "gene space" sequences. Due to the fact that a large portion of the cowpea genome is highly repetitive, the assembly covered just ~40% of the genome and is very fragmented. It consists of from over 600 thousand scaffolds, with an n50 of ~6.3 kb.

We mapped the EST-derived consensus sequences (unigenes) from assembly P12 of HarvEST:Cowpea (harvest.ucr.edu) to the created assembly with BLAST. More than 97% of the unigenes were mapped to the assembly with a high mapping score.

We also produced an assemblies from ~4,000 minimal tiling path BACs using our sequencing protocol based on combinatorial pooling [4]. Briefly, in our sequencing protocol (i) we pool the MTP BACs according to the shifted transversal pooling design, (ii) sequence the DNA in each pool, trim/clean sequenced reads, (iii) assign reads to BACs using our tool HashFilter, (iv) assemble reads BAC-by-BAC using a standard

assembler (e.g., Velvet or SPAdes). Each BAC assembly contained on average 29 scaffolds, with an average n50 of ~14 kb, so in general, the quality of BAC assemblies are better than the whole-genome draft sequence. We believe that the BACs cover ~60% of the cowpea genome because around 60% of the unigenes were mapped to the BAC assemblies.

Although the quality of the draft cowpea genome assembly can be improved, we designed a "60k" iSelect SNP assay (Illumina, Inc., San Diego, CA) based on that. We have used this assay to genotype several cowpea accessions to support high-density map production.

A set of SNPs, discovered by Genotyping-by-sequencing (GBS) method [50] at Cornell University Institute of Genomic Diversity (IGD) was provided by UCR Plant Biology department. GBS is a SNP detection pipeline designed for efficient genotyping of large numbers of samples using next generation sequencing platforms. In this technique, genome complexity is reduced with methylation-sensitive restriction enzyme digestion. The ends of small restriction fragments are sequenced at 96- to 384-plex levels per flow channel on the Illumina HiSeq instrument. The GBS SNPs were discovered from 119 cowpea samples.

4.1.2. Sequencing, sequence alignment and SNP calling

In order to detect SNPs in cowpea we sequenced thirty-six cowpea breeding accessions from Africa, China and the United States, using the Illumina HiSeq 2500 hosted at UC Riverside. The thirty-six libraries composed of 2x100bp paired end reads

had an average coverage of 12.5X. We trimmed the reads based on quality scores and discarded reads shorter than 70bp.

As reported in the previous section, the SOAPdenovo assembly for the cowpea reference genome (cultivar ITK97-499-35) was highly fragmented. Due to the low quality of the reference genome, it was crucial to be very conservative in the analysis of candidate SNPs because of possible assembly errors in the reference sequence. For this purpose, we selected the two highest quality set of reads from ITK97-499-35 and included them in our analysis as the 37th genotype (referred as the *ITK genotype* hereafter).

Each of the thirty-seven sets of reads was individually mapped to the reference genome using BWA [51] (BWA mem with –M option to mark shorter split hits as secondary). On average about 87% of the reads were mapped uniquely to the reference genome. We excluded reads mapped to multiple locations from further analysis. Table 12 summarizes the mapping statistics for each cowpea accession.

The alignment files were merged with the software tool Picard to a single "sam" file. Reads that "hanged off" the end of the contigs in the reference sequence were clipped with Picard. Also, in order to avoid skewed variant calling result, duplicated reads were marked with Picard. Duplicated reads are mostly originated from DNA prep methods and may misguide the SNP calling tools in case they contain sequencing errors.

Table 12. Mapping statistics for 36 cowpea accessions

Accession code	Total number of reads	% reads aligned	Aligned bases	HQ aligned reads	HQ aligned Q20 bases	Mismatch rate	INDEL rate	Mean read length	%reads aligned in pairs
002	5722440	0.882552	4431379635	36695865	3220070361	0.019155	0.001092	96.829325	0.949452
003	63681647	0.877309	4912910063	40571742	3580088877	0.019149	0.001157	96.789498	0.939521
005	62384014	0.881382	4837829484	39780184	3505927785	0.019575	0.00114	96.930588	0.946913
006	68329599	0.891252	5386070218	44040654	3901724888	0.019104	0.001094	96.91561	0.946836
007	92780607	0.883712	7257596016	59704339	5311313645	0.018828	0.001109	97.084049	0.945719
008	58150744	0.877237	4484945854	36914412	3256918292	0.019234	0.001161	96.794883	0.938929
009	55878161	0.882818	4350292145	35597484	3153138800	0.019457	0.001177	96.900026	0.94112
010	42065728	0.891384	3303200232	27036688	2384043625	0.019332	0.001078	96.91001	0.948549
012	56378738	0.903137	4508400774	36646799	3247645178	0.019403	0.001047	97.008748	0.957359
013	53803268	0.883558	4192817538	34207684	3029982306	0.019516	0.00117	96.849769	0.939185
014	52945896	0.876356	4069671078	33359190	2935888072	0.020178	0.001218	96.752236	0.938388
015	54143339	0.880889	4193642216	34304871	3027344732	0.019864	0.001207	96.861697	0.940369
016	54378736	0.876953	4195085842	34803931	3072812157	0.018955	0.001146	96.785913	0.938043
017	54583876	0.915922	4405359380	37153576	3285870030	0.018454	0.000949	97.082056	0.964511
019	56842645	0.885576	4444608799	36609141	3244201895	0.018498	0.001061	96.817979	0.942064
020	59929340	0.879379	4633249502	38259034	3374668962	0.018558	0.001069	96.897608	0.94441
023	63186499	0.896783	5009675872	40464675	3581240448	0.019965	0.001098	97.008681	0.954434
024	92157135	0.875559	7119942628	58457770	5183217542	0.019142	0.001102	97.113443	0.948456
026	66690032	0.885053	5208769588	42925235	3792415772	0.018826	0.001052	96.992409	0.953134
027	56231255	0.896665	4468215430	36267215	3218952959	0.019012	0.001064	96.80786	0.946925
028	64553920	0.893551	5098723586	41774187	3697824190	0.019126	0.001078	96.817063	0.948776
029	13210443	0.88422	8101042014	66938600	597723310	0.018594	0.001098	97.225865	0.947943
030	6100513	0.875042	4611898277	38192040	3358742076	0.019758	0.001204	96.889533	0.941244
032	68917003	0.887372	5407879336	44835048	3977824887	0.018973	0.001119	96.802867	0.944729
033	56101445	0.887647	4396098259	36373452	3222902222	0.018745	0.001102	96.791847	0.94781
034	63589528	0.873437	4872484494	40182975	3538298920	0.019846	0.001195	96.830765	0.94047
035	97555059	0.883285	7633482805	62630838	5578157461	0.019404	0.001163	97.017319	0.946807
036	47343832	0.872003	3611255630	29863594	2616979355	0.020105	0.001221	96.807774	0.940391
038	46396066	0.8721	3540992506	29306022	2571249033	0.019583	0.001185	96.858473	0.938897
039	64295666	0.895606	5049367424	41967673	3688207259	0.020284	0.00113	96.918372	0.957847
040	51907594	0.884407	4028312450	33215155	2916082777	0.019264	0.001089	96.836851	0.948393
G32	57387758	0.87325	4443821204	37397022	3342623035	0.019778	0.00125	98.262768	0.949047
ZJ60	61607426	0.866673	4751798648	39764617	3570054619	0.019605	0.001246	98.278719	0.946589
ZJ282	45115430	0.879375	3541767172	29955497	2694998620	0.019354	0.001177	98.471749	0.95421
ZN016	44195979	0.885919	3506744303	29432724	2659166923	0.019687	0.001171	98.457848	0.957204

In order to call the SNPs genome-wide, we employed three software packages, namely Samtools [52], SGSautoSNP [53] and FreeBayes [54]. We could not use GATK [55] because this tool needs a set of confirmed training SNPs for the base quality score recalibration phase which we did not have for cowpea. We had 1,536 SNPs previously discovered with the Ilumina GoldenGate assay [56], but this number of SNPs was insufficient for training GATK.

Samtools has been widely used for calling SNPs for different organisms due to its simplicity and accuracy [57-59]. Several studies showed that Samtools is almost as reliable as GATK in terms of the quality of discovered SNPs, especially when there is no verified set of SNPs available in advance [60]. In total, Samtools discovered a total of 5,108,787 SNPs (using mpileup with default parameters).

We tried to filter the set of primary discovered SNPs with vcfutils [61] with default parameters, which led to around three million SNPs. We compared the ratio between the number of verified SNPs deleted because of the filtering, and the number of verified SNPs remained in the set. The results did not convinced us that the filtered set was more accurate than the original set, so we moved forward with the original set.

We also independently called the SNPs with SGSautoSNP. This tool has been mainly designed for complex crop genomes (e.g., on the wheat genome[53]). SGSautoSNP does not need a reference sequence for detecting the SNPs. The reference is only used to assemble the reads, then SNPs are then called between the assembled reads. This was a desirable feature for our cowpea project because of the very fragmented reference genome. SGSautoSNPs detected a total of 2,488,797 SNPs.

Finally, we used FreeBayes to independently call the cowpea SNPs. Contrary to tools like samtools and GATK, which discover variants based on the precise alignment of the reads, FreeBayes is a haplotype-based variant detection tool. A haplotype is a set of DNA variations, or polymorphisms, that tend to be inherited together. FreeBayes considers this type of relationship between the SNPs during the calling process. FreeBayes has been used to call SNPs for potato [62], Corvina [63], Adriatic sturgeon [64] and others. FreeBayes called a total of 8,269,140 SNPs on the cowpea genome.

4.1.3. Filtering the candidate SNPs

We designed several filters to determine the most reliable subset of SNPs in cowpea. The first step was to compute the intersection between the SNPs called by Samtools, SGSautoSNP and FreeBayes. About 1.5 million SNPs were called by all the three tools.

We further filtered down the set of 1.5M SNPs based on several additional criteria, namely a) allele frequency, b) existence or absence of repeated patterns (i.e., repeated a certain number of times in the WGS assembly) in the SNP sequence, c) the size of WGS contig containing the SNPs, and d) calling scores generated by each of the tools, and other criteria. Finally, we took advantage of available gene models for evolutionary-related genomes to refine the set further.

We expected that SNPs located inside the gene bodies (or close to them) would be more useful for the downstream applications. Specifically, we took advantage of high quality genes models available for common bean (*P. vulgaris*), which is a close relative of cowpea in the *Fabaceae* family. Due to their relatively close evolutionary distance, a

high level of conservation between the genes models is expected. Over 20 thousand gene models are available for common bean.

In order to find the synthetic blocks between cowpea and common bean, we aligned the two genomes with Mummer [65]. We first removed scaffolds smaller than 100,000 bp in the *P. vulgaris* assembly and then divided the remaining scaffold into two sets based on the chromosome of each scaffolds (chromosomes 1-5 in one set and chromosomes 6-11 in the other). The reason for splitting the files was that Mummer is unable to align genomes larger than a particular size. We then aligned the cowpea genome assembly to the two files separately and merged the results. Mummer reported synthetic blocks between the two genomes. Some of these blocks included common bean gene models. The list of putative gene models in the cowpea assembly was used below in the final selection criteria.

Illumina provides a web-based service which evaluates the candidate SNP sequences based on certain constraints for designing the iSelect chip and informs the user if a particular SNP can be included to the final design or not. A score is assigned to each of the accepted candidate sequences that reflects the probably of the sequence to identify uniquely to the desired SNP (the higher, the better). In some cases, a SNP sequence is acceptable, but requires two assays on the chip (e.g., when the sequence contains a certain number of ambiguous nucleotides). We selected two subsets from the remaining candidate SNPs and focused on these subsets for the rest of the filtering process:

1) The primary list of SNPs consists of the SNPs that required one assay on the chip. The SNPs in this list had a design score between 0.500 to 1 and found either by Samtools, SGSautoSNP and Freebayes or were included in GBS, SGSautoSNP and Freebayes sets.

2) The secondary list of SNPs consists of the SNPs needed two assays on the chip with design score 0.500 to 1 or required one assay on the chip with design score 0.270 to 0.499. Again, These SNPs were found either by samtools, SGSautoSNP and Freebayes or were included in GBS, SGSautoSNP and Freebayes sets.

The final list of SNPs was selected according to the following rules, aimed at selecting the most reliable set of SNPs from the primary and secondary lists:

1. We selected a SNP from the primary list if it was inside the WGS contig boundaries matching a common bean gene model.

2. We selected a SNP from the primary list if it was located on the same WGS contig region (synthetic block) matching a common bean gene model.

3. We selected a SNP from the primary list if it was located within 3000bp of either end of the aligned region (synthetic block) on the same WGS contig as (1).

4. We selected a SNP from the primary list if it was located on any WGS contig (not necessarily the synthetic block with the highest similarity) having SNP within 3000 bp of region matching a common bean gene model.

5. We selected a second SNP from the primary list if it was from the same WGS contig as (4).

6. We selected a SNP from the secondary list for a WGS contig if rules 1-5 resulted in only one SNP for that contig**.** The additional SNP was selected from the region matching the gene model, and at least 246 bp from the first selected SNP.

7. We selected a SNP from the secondary list if only one SNP was chosen in (6) for a particular WGS contig. Again, the second SNP was chosen from the same WGS contig as the first SNP chosen. In this case, a SNP was selected near the region matching the gene model, at least 246 bp from the first SNP chosen.

8. We selected a SNP from the secondary list, for any common bean gene model that has not as yet had any SNP chosen and any WGS contig having SNP within 3000 bp of region matching a common bean gene model.

9. We selected a second SNP from same WGS contig as (8), from the secondary list.

10. For any common bean gene model that has one SNP selected, we chose a second SNP from the primary list, if it was at least 130 bp away from the first SNP in the same WGS gene model.

11. We selected from the GBS set, the SNPs in or near regions of a common bean gene models (only the SNPs which needed one assay with design score at least 0.5, minor allele frequency at least 0.25).

12. We selected a SNP from the whole list of SNPs, when only one of the 37 individuals had minor allele (only the SNPs requiring one assay with design score at least 0.5).

In addition, 1163 GoldenGate assay SNPs had a good technical score when submitted to the Illumina website and passed the other filters. These SNPs were also included to the

final set. The final set provided to Illumina for the chip design contained 56,719 SNPs (instead of 60K), due to the fact that two assays were required by some of the SNPs on the chip.

4.1.4. SNP validation

Once the iSelect chip was available, we genotyped several cowpea tissues and investigated the quality of the final list of SNPs with GenomeStudio software. GenomeStudio clusters the input samples based on the detected SNPs and visualizes the clusters. A high quality SNP generates clear clusters with sharp borders. Also, this tool helps to realize if the call rate of a specific SNP is within the expected range (Figure 21).

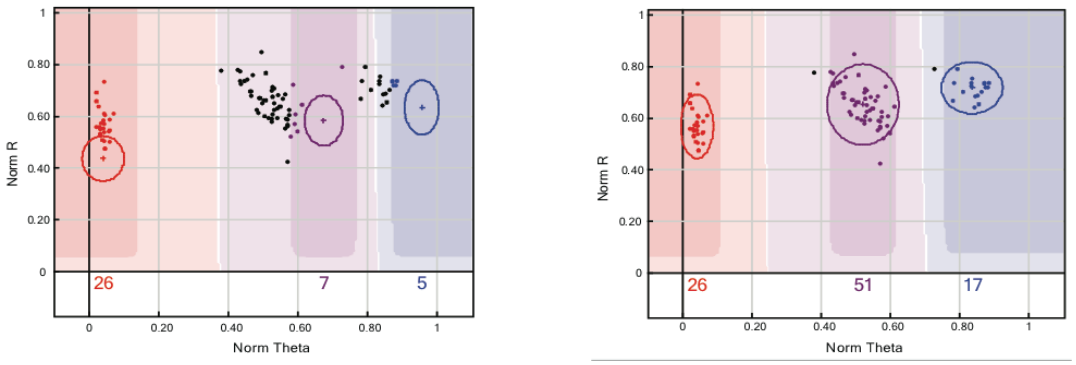

Figure 21. Left: an example of a low quality SNP, detected by GenomeStudio (Illumina). Right: An example of a high quality SNP. Borders are sharp and the samples fall close to the cluster centers.

Based on this analysis, more than 49,000 SNPs (96%) were led to "clean" clusters indicating that these SNPs are likely to be real. These SNPs can be used for allele mining and high-density mapping, and can adapted to other genotyping platforms for a range of breeding applications.

4.1.5. Ordering and anchoring the cowpea BAC

As mentioned earlier, we sequenced the cowpea cultivar IT97K-499-35 based on two different approaches:

1) BAC by BAC: We sequenced 4,353 minimal tiling path BACs, which are expected to cover around 60% of all the cowpea genome. We assembled the reads for each of the BACs separately with SPAdes [16]. Each BAC assembly contained on average 29 scaffolds, with an average n50 of about 14 kb.

2) Whole Genome Shotgun (WGS): We sequenced the entire cowpea genome with Illumina GAII at ~60x coverage. In addition, a 5 kb mate pair library, about 30,000 Sanger BAC-end sequences and about 250,000 Sanger "gene space" were assembled with SOAPdenovo. The result was a set of 644,126 scaffolds, with an n50 of about 6.3 kb.

In order to take advantage of both assemblies, we had to order and orient BAC and WGS contigs along the cowpea chromosomes. Observe that the order and orientation of the contigs created by SPAdes for each BAC are unknown. Similarly, the order and orientation for the WGS contigs and scaffolds created by SOAPdenovo are unknown.

We took advantage of the genotyping data obtained from the iSelect to create a high-density genetic map on which we could order and orient BAC contig and WGS contigs/scaffolds. We used MSTmap [66] to generate a cowpea genetic map for 14,868 markers (SNPs). We then selected 121bp around each of the 50,747b SNPs with known coordinates from the WGS contigs (60bp from the right side and 60bp from the left side of the SNP). Out of 50,747b SNP "design sequences", 49,645 didn't contain any ambiguous nucleotide. Among the 1,102 SNP sequences with ambiguous

nucleotide, 1,060 had Ns at the ends, which were trimmed. We excluded 48 SNPs with ambiguous nucleotides in the middle from further analysis. Because of the trimming process some of the SNP sequences were shorter than 121bp. By comparing the genetic map with the unambiguous SNP list, we obtained the coordinates for 37,161 SNPs (iSelect SNPs), located on 25,244 WGS scaffolds. In some cases, we observed a conflict between the genetic map and the prior knowledge about certain SNPs. For example, SNPs that were located on a particular WGS contigs according to the iSelect design but mapped to two distinct chromosomes based on the genetic map. We marked those SNPs for further investigations.

We then mapped the SNP design sequences against the BAC assemblies with BWA[38]. The result was filtered to find only perfect (exact) mappings. As a result, 12,210 SNPs were mapped uniquely to the BAC assemblies (to 2,040 unique BACs). Among them, the coordinates for 8,853 SNPs were available, which revealed the location of 1,786 BACs. According to the mapping result, 10,935 SNP sequences were mapped to two BACs. We assumed that these SNPs were located on the overlapping portion of the adjacent BACs. The coordinates for 8,058 of these SNPs were determined. Also, 2,799 SNP sequences were mapped to three BACs. Although it is possible for three MTP BACs to be overlapping, we decided to exclude them from further analysis.

We then mapped the WGS scaffolds to the BAC assemblies. After filtering and analysis, 116,378 WGS scaffolds were mapped to a single BAC and 114,032 to two BACs (overlapping section). After integrating the BACs with known position and the contigs mapped to the BACs, we found the approximate location for 46,347 more contigs.

4.2 Reference-guided assembly of heterogeneous DNA segments to improve the quality of *P.falciparum* DD2 genome

In this section, we present a method to assemble a mixture of reads and DNA segments based on a reference genome. We applied this method to improve the quality of assembly for *P. falciparum* strain DD2 based on the genome of *P. falciparum* strain 3D7. *Plasmodium falciparum*, the parasite that causes malaria in human is a major cause of mortality worldwide, infecting approximately 500 million individuals each year. It is estimated that 3,000 children under the age of five years fall victim to malaria each day. Around 40% of the worlds population is at risk. In spite of the vast investments, there is still no effective vaccine for malaria. The number of infected individuals is increasing due to increasing drug resistance and globalization. *Plasmodia* infect many organisms including birds, rodents, monkeys and human.

Four species of *Plasmodia* cause malaria in human; among them *Plasmodium falciparum* is the deadliest. *P.falciparum* has fourteen chromosomes, a mitochondria and a apicoplasts. The sequence of its mitochondrion was reported in 1995 [67] and its complete genome was published in 2002 [68]. The genome size is almost 24 million bases and very AT rich. The complete sequence of *P. falciparum* genome has enabled researchers to identify many of the genes involved in drug resistance and to understand the underlying mechanisms that control the biology of this parasite. What makes this organism especially interesting for researchers is the fact that the mechanism controlling gene regulation in *Plasmodia* appears to be different from known mechanisms of

transcriptional regulation in other organisms. Strong evidences suggest that epigenetic mechanisms play an important role in malaria parasite gene expression.

P. falciparum appears in the wild in many different variants, or *strains*: 3D7 is the most widely studied strain. DD2 is another strain known to be resilient against an anti-malaria drug called *artimisin*. The Broad Institute reported a Sanger-based draft genome sequence of *P.falciparum* DD2 strain in 2007 [69].

4.2.1. Related work

Our goal in this project is to improve the quality of DD2 Sanger-based assembly with the 2nd generation sequencing data (NGS). For this purpose, we used *P. falciparum* strain 3D7 as a reference sequence to help with the assembly of NGS reads.

We review some of the approaches proposed in the literature to improve the quality of draft contigs and advance a genome from a draft assembly to an improved or finished state. CloG [70] is a technique to close gaps in a draft assembly. It has two main steps: 1) generate a hybrid *de novo* assembly from NGS short reads and the original draft assembly. 2) close the gaps between adjacent contigs by reconciling the two assemblies. The basic idea behind reconciliation is to generate a consensus sequence by finding overlapping regions of the two assemblies. For this purpose, a hybrid assembly contig that shares common seeds with two different draft assembly contigs is detected. Seeds are specific length sequences located at a specific distance away from contig ends. Consensus sequences are constructed by stitching together appropriate fragments from the two assemblies.

Tsai *et al.* have developed an approach [71] to improve the quality of a draft assembly with local assembly of gap regions. Reads that are belong to gap sections or questionable regions are identified and reassembled locally before being incorporated back into the final assembly. In order to do this, reads are aligned against the initial assembly. The reads aligned to the contig ends, with their mates, are assembled into new contigs, which are subsequently mapped back to the initial assembly. Then, reads are aligned against the updated assembly and the whole process is repeated iteratively until the gap is closed or no new useful read is found.

It is also possible to improve the quality of their assembly by closing the gaps manually. In order to do that, they produce additional sequence referred to as *finishing reads*. Finishing reads derive from PCR, primer walking, transposon bombing, shotgun of individual clones and other techniques. Reads are extended and manually aligned to close the gaps and improve questionable regions. This manual process is labor intensive and time consuming. Furthermore, the increases in data volumes and the small contig sizes which is the consequence of using NGS reads, have increased the time and costs needed to advance a genome from a draft assembly to an improved or finished state. For example, Koren *et al.* proposed a method [72] that generates a *de novo* assembly which integrates the whole genome shotgun sequencing technique and finishing reads. The algorithm uses the sets of finishing reads and placement bounds for each set to incorporate finishing reads during the assembly process.

Another approach for finishing draft assemblies is to assemble different types of reads independently and combine the result subsequently. Casagrande *et al.* [73] proposed a

method to improve the overall quality of the genome assembly sequences by merging the sequences produced with different assembly techniques. In each step, two assemblies are combined. The user chooses one of the assemblies as the master assembly by the user. The algorithm tries to improve the quality of the master assembly using the information obtained from the other. For this purpose, the two contig sets are mapped and corresponding contigs are detected. The longer contig is reported as the final result for each contig pair. If significant differences are observed between contigs, the master assembly is chosen.

Assembly reconciliation is a method to integrate different assemblies proposed by Zimin *et al.* [74]. Again, the main goal is to extend the contigs by merging sequences coming from different assemblies. Incorrect assembled fragments are detected by exploiting mean insert size. In particular, if the distance between paired reads in a particular section diverges from the expected value, it is marked as a potential breakpoint.

A couple of methods have been suggested to incorporate mixtures of reads based on a reference-guided assembly. These methods usually combine *de novo* and reference guided assembly techniques to take advantage from both approaches.

Cattonaro *et al.* extend their idea to improve the assembly result when a reference genome is available [74]. This method first builds a *de novo* master assembly. Also, a reference guided assembly is generated. These two assemblies are combined using the technique proposed in [75]. To detect possible breakpoints and break assemblies to smaller sections, assembly reconciliation technique [14] is used.

Gnerre *et al.* [76] have proposed a method to improve low-coverage *de novo* assemblies by exploiting the genome sequence of a related organism. The method begins with *de novo* assembly of the reads, followed by mapping the reads to the reference genome. Mapped reads are grouped based on their position. Each group is assigned to a *de novo* contig (or contigs) based on the similarity between the reads in the group and the contig. Grouped reads are used to improve the quality of the contig. Scaffolding is carried out using paired-end reads. The authors applied this method to obtain whole genome sequence of four divergent *Arabidopsis thaliana* in 2010 [77].

TASR [78] is a reference guided assembly algorithm for very large NGS data sets. Sequence targets are read first. From each target, every possible 15-character word from the plus and minus strand are extracted and stored in a hash table. Next, reads from the NGS data set are processed: any read with an exact match of its first 15 bases to any of the 15-mer words from the target sequence, is retained. The identity and coverage of every base, within and beyond the user-provided target sequence, is stored in a hash table. The sequence within the bounds of the user-supplied target sequence will exactly match the target itself, but recruited sequence reads will typically extend beyond the boundaries of the target sequence, and this flanking sequence may also be included in the assembly. A consensus sequence is derived, taking exactly matching bases at each position within the target region, and extended outward, bi-directionally, to include the most represented base at positions outside the target sequence. Extension is terminated when a position is encountered that does not meet the user-specified criteria.

LOCAS [79] is another method designed for assembling short to medium sized reads either *de novo* or in a homology-guided fashion using an overlap-layout-consensus approach.

4.2.2. Methods and Excremental Results

Sequence comparison between *P. falciparum* DD2 strain and other *Plasmodia* can provide critical information about the underlying mechanisms of drug resistance in the DD2 strain. In addition, it enables researchers to carry out several other genome-wide studies, e.g., gene expression analysis via RNA-seq, epigenetic studies, etc. As said, the goal is to derive an efficient and accurate method to combine heterogeneous reads to build an improved version of DD2 genome, based on the reference 3D7 genome.

We started with a heterogeneous set of input data, including single and paired end reads of *P. falciparum* DD2 strain, Sanger contigs of DD2 strain from the Broad Institute of MIT and Harvard and a high quality assembly of *P. falciparum* 3D7. The Broad Institute (BI) generated the assemblies as a part of a project for comparing different strains of *P. falciparum*. These assemblies were built from Sanger reads, which are usually longer and cleaner than NGS reads in terms of sequencing errors. In this case, we quickly realized that the DD2 Sanger-based contigs were not very accurate. Table 13 summarizes the statistics collected for the BI contigs.

Table 13. Statistics for the DD2 Sanger contigs generated by the Broad Inst.

Num of contigs	Min contig len	Mean contig len	N50	Max contig len	Total length
4,511	201	4,311	11,610	79,198	19.5 M

P. falciparum has a very AT-rich and repetitive genome; this make *de novo* assembly of the genome very challenging. We decided not to rely on *de novo* assembly of the sequencing data because (1) the genome of *P. falciparum* is very repetitive genome, (2) we had to deal with a diverse set of the input data, and (3) the quality of the NGS reads was relatively low. We decided to use a reference genome as a guide to assemble the reads. Figure 22 shows the pipeline we designed to assemble the genome of *P.falciparum* DD2 from the set of heterogeneous sequencing data. This pipeline, partially based on a method proposed in [77], contains the following steps:

a) *Trimming the short reads*: The process began with the trimming of single and paired end reads. The quality of both end of the NGS reads tend to be lower. Also, NGS reads contain at the beginning a DNA barcode and an adapter which has applications in the sequencing process. NGS instruments have internal software modules for removing adapters and low quality bases, but sometimes it is necessary to re-trim the reads before assembling them.

Trimming started with removing the low quality nucleotides (with score lower than 25 for the paired end and 10 for the single end reads) from the both ends. Removing the reads with several low quality nucleotides in the middle was the second step. Reads with more than 20% low quality bases or "N"s were removed in this step. To help the mapper to align the reads more accurately, base pairs with quality score lower than 15 were replaced with "N". Finally, six base pairs at the beginning of all the reads were removed based on the quality profile of the reads. We eliminated all the reads shorter than 15 bases

(18 bases for single end reads). Table 14 contains the statistics collected about the three input sets, before and after trimming.

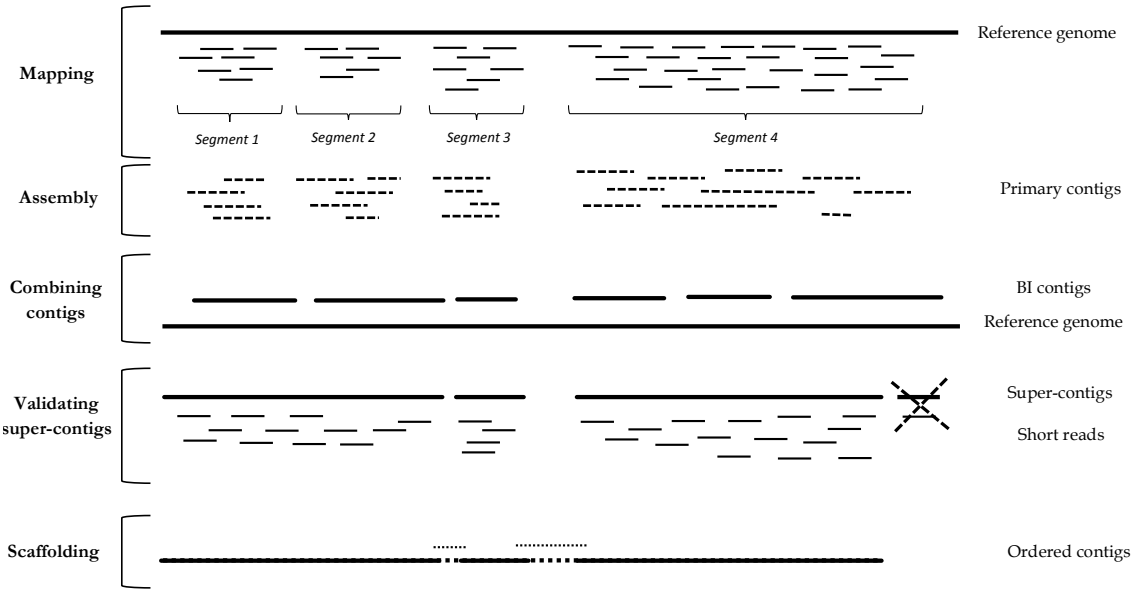

Figure 22. The reference-guided assembly of heterogeneous DNA segments pipeline

b) *Mapping and clustering the reads*: because of the similarity between the reference and the target genome, we expected the reads to cover most of the reference genome. As expected, some parts of the reference genome remained uncovered (or poorly covered) due to mutations and structural variations.

Table 14. Statistics for the input datasets, before and after trimming

	Reads before trimming	Reads after trimming	Unmatched reads after trimming	CG% (before | after trimming)	Max length before trimming	Min length after trimming
Single end	37,376,378	29,926,656	-	30% | 28%	76	18
Paired 1 (C)	41,735,228 pairs	29,233,333 pairs	15,789,230 (total)	18% | 17%	51 (each pair)	15
Paired 2 (G)	36,592,078 pairs	29,741,545 pairs	15,789,230 (total)	37% | 36%	51 (each pair)	15

We mapped a set of single end reads, two sets of paired end reads against the 3D7 genome separately with BWA. For each read, the maximum edit distance was set to 5% of the read length, including at most 3% of gap opens. We prevented BWA from mapping the reads with long gaps. Table 15 shows the result of mapping DD2 reads to the 3D7 genome.

Table 15. Mapping statistics

Name	Alignment percentage	Unique mapped	Multiple mapped
SE reads	76%	68%	8%
Unmated reads	68%	62%	6%
PE read set 1	59%	55%	4%
PE read set 2	30%	26%	4%

We expected a significant deletion near the beginning of chromosome 2 in the DD2 genome. We observed the deletion by visualizing the mapping result, as shown in the left panel of Figure 23.

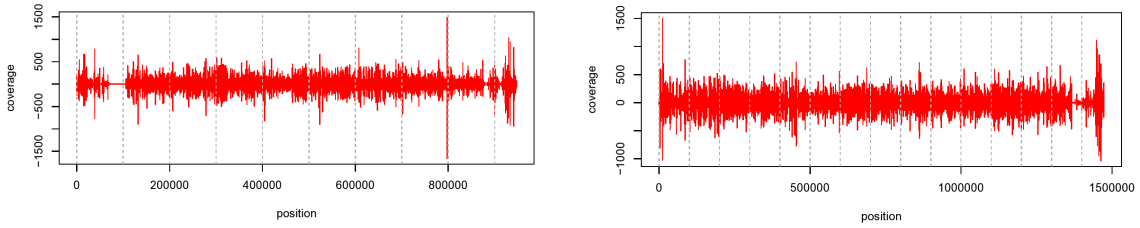

Figure 23. Left: coverage profile of chromosome 2, including a deletion. b) coverage profile of chromosomes 8.

We then separated the mapped reads into several group based on the depth of coverage. We defined a *segment* as a section on the reference genome between two adjacent not covered by reads. Segments and their corresponding reads were detected

with software HORE [80]. We excluded from the analysis segments shorter than 200 bps. Mitochondrion and apicoplast were considered as a single segment each. These segments covered roughly 93% of the 3D7 genome. Figure 24 shows the length distribution of the created segments, genome-wid.

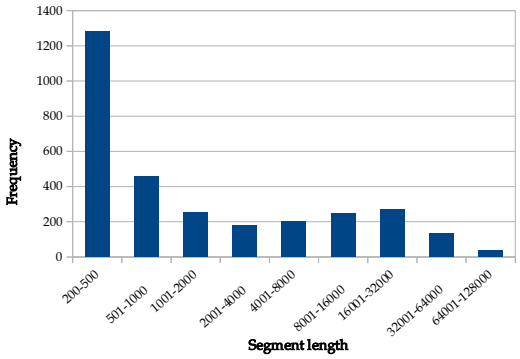

Figure 24. Length distribution of the created segments

c) *Assembly of the reads in each segment*: After partitioning the mapped reads, we carried out a local *de novo* assembly in each segment. In this case, the assembler handled fewer reads and smaller targets, thus simplifying the assembly process.

De novo assembly is highly sensitive to parameters like the *k*-mer size. Choosing an appropriate *k*-mer is not trivial as it is a trade off between specificity and sensitivity. Longer *k*-mers decrease mis-assembly errors because the detected overlaps are more reliable. Shorter k-mers increase contiguity in the assembly. We used VELVET [14]to assemble the reads assigned to each segment. VELVET was run twelve times for each read set using twelve different *k*-mer sizes. We called these local assemblies *primary contigs*.

d) *Merging the contigs*: We mapped the primary contigs to the reference genome again in order to detect overlaps and merge them. We obtained several sets of connected contigs, called *supercontigs*. We merged the created contigs using the assembler

AMOScmp [81] which generated a set of non-redundant supercontigs. The BI contigs mapped to the target chromosome were added to the local assemblies in this step.

We then mapped the draft contigs against the reference genome using BLAST [82] , which enabled us to find the corresponding chromosome for each draft contig. Mapping the draft contig with the primary contigs to their corresponding chromosome (and not the entire genome) produced more accurate results. 4,216 of the BI contigs were aligned to 3D7 genome, including 3,829 uniquely mapped contigs. The total length of the mapped contigs (including the repeated contigs) was 20,280,254 bps.

Table 16 summarizes the main statistics about the BI contigs and the result of combining the contigs generated in this step and the BI contigs. Figure 25 shows the total length of the two contig sets and N50 for each chromosome. Observe that our contigs have larger N50 and they significantly longer than BI contigs for most of the chromosomes.

Table 16. Comparison of the BI contigs and the contigs produced by our pipeline

	Our contigs							BI contigs						
	n	min	median	mean	N:50	max	sum	n	min	median	mean	N:50	max	sum
Chr 1	290	200	1,377	2,467	**4,342**	23,629	533,030	346	233	1,249	1,996	2,203	23,600	**690,853**
Chr 2	348	201	1,435	3,014	**6,612**	66,171	816,945	366	204	1,251	2,571	5,422	59,641	**941,118**
Chr 3	368	204	1,480	3,309	**8,390**	50,081	976,393	376	203	1,246	2,656	7,019	31,357	**998,850**
Chr 4	498	201	1,256	2,699	5,608	40,788	996,199	429	203	1,242	2,603	**7,440**	43,280	**1,116,813**
Chr 5	559	202	1,040	2,685	**7,390**	53,549	**1,272,902**	409	218	1,270	2,813	7,320	38,536	1,150,919
Chr 6	487	200	1,289	3,154	7,340	63,633	1,252,322	531	204	1,239	2,681	**7,916**	57,538	**1,423,991**
Chr 7	490	200	1,254	3,212	**10,283**	43,095	1,262,415	446	229	1,265	2,900	8,957	40,513	**1,293,403**
Chr 8	578	200	1,302	2,815	5,963	30,606	**1,300,960**	466	211	1,239	2,642	**7,069**	42,645	1,231,384
Chr 9	568	202	1,262	3,218	**9,636**	65,466	**1,416,117**	454	221	1,300	2,844	7,817	33,203	1,291,368
Chr 10	658	200	1,257	2,929	7,569	42,100	**1,552,881**	567	209	1,289	2,719	**7,938**	39,372	1,542,120
Chr 11	623	200	1,534	3,602	**10,550**	39,247	**1,920,340**	657	180	1,295	2,891	8,272	34,994	1,899,808
Chr 12	758	200	1,392	3,310	**8,358**	52,954	**2,069,271**	599	222	1,344	2,915	7,684	27,118	1,746,543
Chr 13	873	200	1,468	3,669	**10,504**	50,108	**2,759,147**	733	210	1,300	3,317	9,170	33,326	2,431,576
Chr 14	800	202	1,770	4,581	**12,839**	88,013	**3,156,696**	650	206	1,512	3,744	9,252	36,634	2,433,612

e) *Quality check and validation of the contigs*: Because of the repeats in the target genome and sequencing errors in the reads, invalid contigs could be generated. Therefore, a validation process was necessary. In order to do this, we mapped the single and paired end reads to the supercontigs. We then eliminated supercontigs with insufficient coverage.

f) *Scaffolding*: The order of supercontigs was still undetermined at the beginning of this step. Neighbor supercontigs were determined with the paired reads aligned to the two separate supercontigs.

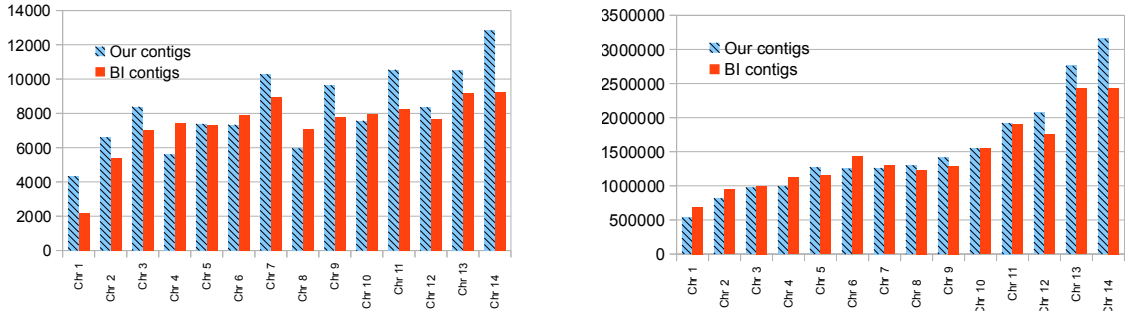

Figure 25. Left: Comparison of sum of the length of BI contigs and the improved contigs. Right: Comparison of N50 for BI contigs and the improverd contigs

g) *Building new contigs and patching them*: After mapping single and paired end reads against the reference genome, some reads were still remained unmapped. The unmapped and the orphan reads (paired end reads with one mate mapped and the other unmapped) could be distinctive parts of the DD2 genome that are not present in the 3D7 genome, or they could be due to contamination. Since *P. faciparum* parasites are grown in human blood, we expected that a fraction of the short reads was from the human genome. To identify these reads, we aligned all reads not mapped to the 3D7 genome

against the human genome and removed the mapped reads. We mapped the leftover reads to the genome of every known organism with BOWTIE [83] (no mismatch and gap). Table 17 summarizes statistics of aligning the reads not mapped to the 3D7 genome to the human genome.

To build unique (distinctive) blocks of the target genome, the remaining unmapped reads were assembled *de novo* with VELVET. We repeated the assembly with 12 different *k*-mers. As we assembled short reads without any reference in this step, the quality of the created contigs were not as high as BI contigs which were not mapped to the chromosomes.

Table 17. Aligning the reads not mapped to the 3D7 against the human genome

	Unmapped SE reads	Orphan reads	Unmapped PE reads
Number of reads	12,341,533	929,012	21,217,665
Unmapped to the HG	7,956,583 (64%)	670,081 (72%)	12,095,895 (57%)

4.2.3. Methods and Excremental Results

We presented a method that enabled the assembling of heterogeneous DNA segments of *P. falciparum* strain DD2 based on a reference genome (3D7). We followed a pipeline consist of trimming, mapping, partitioning, *de-novo* assembly, combining contigs, validating and scaffolding. Contigs obtained with our pipeline were longer on average than the BI original contigs. However, the BI contigs not mapped to the 3D7 genome were longer in average than the contigs we generated with unmapped reads

Chapter 5: Conclusions

Advancement in sequencing technologies has been reducing sequencing costs at an astonishing rate. The estimated cost of the human genome project was about 3 billion dollars. At the time of writing the cost of sequencing one human genome is below $1,000. For the same reason, ultra-deep sequencing is also now feasible, especially for smaller genomes and clones. As it becomes more and more common, ultra-deep sequencing data is expected to create new algorithmic challenges in the analysis pipeline. In this dissertation, we focused on two of these challenges: the accuracy of decoding reads and quality of *de novo* assemblies created from the ultra deep sequencing data.

In hierarchical genome sequencing approach, a genome is sequenced into long DNA fragments (i.e., BAC clones). To take advantage of the throughput of modern sequencing instruments, BACs are usually pooled before sequencing. Each reads then has be assigned to its original BAC after the sequencing process. Our experiments showed that decoding "slices" of the input reads instead of the whole dataset may increase the accuracy of the decoding process. We presented an effective 'divide and conquer' solution: we 'slice' the data in subsamples, decode each slice independently, then merge the results. In order to handle conflicts in the BAC assignments (i.e., reads that appear in multiple slices that are decoded to different sets of BACs), we devised a simple set of voting rules.

We also showed that popular modern *de novo* assemblers are unable to take advantage of ultra-deep coverage, and the quality of assemblies starts degrading after a certain depth

of coverage. We proposed an iterative approach to solves this problem, which significantly improves the final quality of the assembly. Experiments on a set of ultra-deep barley BACs and simulated data shows that our proposed method leads to high quality assemblies. We also demonstrated that this approach is more resilient to high sequencing error rates than the other methods.

Additionally, we reported on a protocol to discover high quality SNPs for complex and repetitive genomes like plant genome. The SNPs discovered for the cowpea genome using the protocol were used to design an Illumina "60k" iSelect genotyping chip. The pipeline for ordering and orienting the previously sequenced cowpea BACs and WGS contigs, using the discovered SNPs, was reported in this dissertation as well.

Finally, a computational pipeline for assembly of heterogeneous sequencing data was described and the quality of assemblies created for a drug-resistant malaria strain based on the pipeline was investigated.

To conclude, new algorithms and methods will be required to handle deeper and more heterogeneous sequencing data as we witness advances in sequencing technology. This dissertation is a step toward this goal.

Bibliography

1. The Genome International Sequencing Consortium, *Initial sequencing and analysis of the human genome.* Nature, 2001. **409**: p. 860-921.

2. Willis, J.C. and G.M. Lord, *Immune biomarkers: the promises and pitfalls of personalized medicine.* Nat Rev Immunol, 2015. **15**(5): p. 323-9.

3. Waterman, M.S., *Introduction to computational biology: maps, sequences and genomes*. 1995: CRC Press.

4. Lonardi, S. and others, *Combinatorial Pooling Enables Selective Sequencing of the Barley Gene Space.* PLoS Comput Biol, 2013. **9**(4): p. e1003010.

5. Stein, N. and others, *A physical, genetic and functional sequence assembly of the barley genome.* Nature, 2012. **491**(7426): p. 711--716.

6. Munoz-Amatriain, M., et al., *Sequencing of 15 622 gene-bearing BACs clarifies the gene-dense regions of the barley genome.* Plant J, 2015.

7. Yu, Y. and others, *A bacterial artificial chromosome library for barley (\emphHordeum vulgare L.) and the identification of clones containing putative resistance genes.* Theoretical and Applied Genetics, 2000. **101**(7): p. 1093-1099.

8. Luo, M.-C. and others, *High-throughput fingerprinting of bacterial artificial chromosomes using the snapshot labeling kit and sizing of restriction fragments by capillary electrophoresis.* Genomics, 2003. **82**(3): p. 378--389.

9. Bozdag, S. and others, *A compartmentalized approach to the assembly of physical maps*, in *Proceedings of IEEE International Symposium on Bioinformatics \& Bioengineering (BIBE'07)*. 2007. p. 218--225.

10. Soderlund, C. and others, *Contigs Built with Fingerprints, Markers, and FPC V4.7.* Genome Research, 2000. **10**(11): p. 1772-1787.

11. Bozdag, S. and others, *A Graph-Theoretical Approach to the Selection of the Minimum Tiling Path from a Physical Map.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2013. **10**(2): p. 352-360.

12. Thierry-Mieg, N., *A new pooling strategy for high-throughput screening: the shifted transversal design.* BMC Bioinformatics, 2006. **7**(28).

13. Ding, Y. and others, *Five-Color-Based High-Information-Content Fingerprinting of Bacterial Artificial Chromosome Clones Using Type IIS Restriction Endonucleases.* Genomics, 2001. **74**(2): p. 142--154.

14. Zerbino, D.R. and E. Birney, *Velvet: algorithms for de novo short read assembly using de Bruijn graphs.* Genome Res, 2008. **18**(5): p. 821-9.

15. International Barley Genome Sequencing, C., et al., *A physical, genetic and functional sequence assembly of the barley genome.* Nature, 2012. **491**(7426): p. 711-6.

16. Bankevich, A., et al., *SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing.* Journal of Computational Biology, 2012. **19**(5): p. 455-77.

17. Peng, Y., et al., *IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth.* Bioinformatics, 2012. **28**(11): p. 1420-8.

18. Salzberg, S.L. and others, *GAGE: A critical evaluation of genome assemblies and assembly algorithms.* Genome Research, 2011.

19. Lander, E.S. and M.S. Waterman, *Genomic mapping by fingerprinting random clones: a mathematical analysis.* Genomics, 1988. **2**(3): p. 231--239.

20. Roach, J.C., et al., *Pairwise end sequencing: a unified approach to genomic mapping and sequencing.* Genomics, 1995. **26**: p. 345-353.

21. Ekblom, R., L. Smeds, and H. Ellegren, *Patterns of sequencing coverage bias revealed by ultra-deep sequencing of vertebrate mitochondria.* BMC Genomics, 2014. **15**: p. 467.

22. Eid, J., et al., *Real-time DNA sequencing from single polymerase molecules.* Science, 2009. **323**(5910): p. 133-138.

23. Clarke, J., et al., *Continuous base identification for single-molecule nanopore DNA sequencing.* Nat Nanotechnol, 2009. **4**(4): p. 265-70.

24. English, A.C., et al., *Mind the gap: upgrading genomes with Pacific Biosciences RS long-read sequencing technology.* PLoS One, 2012. **7**(11): p. e47768.

25. Spence, J.M., J.P. Spence, and W.R. Burack, *Quantification of intraclonal diversity in follicular lymphoma.* Modern Pathology, 2012. **25**: p. 372A-372A.

26. Campbell, P.J., et al., *Subclonal phylogenetic structures in cancer revealed by ultra-deep sequencing.* Proceedings of the National Academy of Sciences of the United States of America, 2008. **105**(35): p. 13081-13086.

27. Widasari, D.I., et al., *A deep-sequencing method detects drug-resistant mutations in the hepatitis B virus in indonesians.* Intervirology, 2014. **57**(6): p. 384-392.

28. Beerenwinkel, N. and O. Zagordi, *Ultra-deep sequencing for the analysis of viral populations.* Current Opinion in Virology, 2011. **1**(5): p. 413-418.

29. Lonardi, S., et al., *When less is more: 'slicing' sequencing data improves read decoding accuracy and de novo assembly quality.* Bioinformatics, 2015. **31**(18): p. 2972-80.

30. Desai, A., et al., *Identification of optimum sequencing depth especially for de novo genome assembly of small genomes using next generation sequencing data.* PLoS One, 2013. **8**(4): p. e60204.

31. Zhou, W., et al., *Bias from removing read duplication in ultra-deep sequencing experiments.* Bioinformatics, 2014. **30**(8): p. 1073-1080.

32. Aird, D., et al., *Analyzing and minimizing PCR amplification bias in Illumina sequencing libraries.* Genome Biology, 2011. **12**(2): p. R18.

33. Yang, X., S.P. Chockalingam, and S. Aluru, *A survey of error-correction methods for next-generation sequencing.* Briefings in Bioinformatics, 2013. **14**(1): p. 56-66.

34. Brown, C.T., et al., *A reference-free algorithm for computational normalization of shotgun sequencing data*, in *arXiv preprint arXiv:1203.4802*. 2012.

35. McCorrison, J.M., et al., *NeatFreq: reference-free data reduction and coverage normalization for de novo sequence assembly.* BMC Bioinformatics, 2014. **15**(1): p. 357.

36. Hui, L.C.K., *Color Set Size Problem with Applications to String Matching.* Combinatorial Pattern Matching, 1992. **644**: p. 230-243.

37. Pop, M., D.S. Kosack, and S.L. Salzberg, *Hierarchical scaffolding with Bambus.* Genome Research, 2004. **14**(1): p. 149-159.

38. Li, H. and R. Durbin, *Fast and accurate short read alignment with Burrows-Wheeler transform.* Bioinformatics, 2009. **25**(14): p. 1754-60.

39. Ilie, L. and M. Molnar, *RACER: Rapid and accurate correction of errors in reads.* Bioinformatics, 2013. **29**(19): p. 2490-3.

40. Gurevich, A., et al., *QUAST: quality assessment tool for genome assemblies.* Bioinformatics, 2013. **29**(8): p. 1072-5.

41. Boisvert, S., F. Laviolette, and J. Corbeil, *Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies.* Journal of Computational Biology, 2010. **17**(11): p. 1519-33.

42. Raghunathan, A., et al., *Genomic DNA amplification from a single bacterium.* Appl Environ Microbiol, 2005. **71**(6): p. 3342-7.

43. Soueidan, H., et al., *Finishing bacterial genome assemblies with Mix.* BMC Bioinformatics, 2013. **14**: p. S16.

44. Nijkamp, J., et al., *Integrating genome assemblies with MAIA.* Bioinformatics, 2010. **26**(18): p. i433-9.

45. Vicedomini, R., et al., *GAM-NGS: genomic assemblies merger for next generation sequencing.* BMC Bioinformatics, 2013. **14**: p. S6.

46. Turk, K.J., A.E. Hall, and C.W. Asbell, *Drought adaptation of cowpea. I. Influence of drought on seed yield.* Agronomy Journal, 1980. **72**(3): p. 413-420.

47. Mai-Kodomi, Y., B.B. Singh, and O. Myers, *Two mechanisms of drought tolerance in cowpea.* Indian Journal of Genetics & Plant Breeding, 1999. **59**(3): p. 309-316.

48. Pottorff, M.O., et al., *Genetic mapping, synteny, and physical location of two loci for f. sp. race 4 resistance in cowpea [(L.) Walp].* Mol Breed, 2014. **33**: p. 779-791.

49. Timko, M.P., et al., *Sequencing and analysis of the gene-rich space of cowpea.* BMC Genomics, 2008. **9**: p. 103.

50. Elshire, R.J., et al., *A robust, simple genotyping-by-sequencing (GBS) approach for high diversity species.* PLoS One, 2011. **6**(5): p. e19379.

51. Li, H. and R. Durbin, *Fast and accurate short read alignment with Burrows-Wheeler transform.* Bioinformatics, 2009. **25**(14): p. 1754-1760.

52. Li, H., et al., *The Sequence Alignment/Map format and SAMtools.* Bioinformatics, 2009. **25**(16): p. 2078-9.

53. Lorenc, M.T., et al., *Discovery of Single Nucleotide Polymorphisms in Complex Genomes Using SGSautoSNP.* Biology (Basel), 2012. **1**(2): p. 370-82.

54. Garrison, E. and G. Marth, *Haplotype-based variant detection from short-read sequencing.* arXiv preprint arXiv:1207.3907, 2012.

55. McKenna, A., et al., *The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data.* Genome Res, 2010. **20**(9): p. 1297-303.

56. Lucas, M.R., et al., *Cowpea–soybean synteny clarified through an improved genetic map.* The Plant Genome, 2011. **4**(3): p. 218-225.

57. Arai-Kichise, Y., et al., *Discovery of genome-wide DNA polymorphisms in a landrace cultivar of Japonica rice by whole-genome sequencing.* Plant Cell Physiol, 2011. **52**(2): p. 274-82.

58. You, F.M., et al., *Annotation-based genome-wide SNP discovery in the large and complex Aegilops tauschii genome using next-generation sequencing without a reference genome sequence.* BMC Genomics, 2011. **12**: p. 59.

59. le Roex, N., et al., *Novel SNP Discovery in African Buffalo, Syncerus caffer, using high-throughput Sequencing.* PLoS One, 2012. **7**(11): p. e48792.

60. Pabinger, S., et al., *A survey of tools for variant analysis of next-generation genome sequencing data.* Brief Bioinform, 2014. **15**(2): p. 256-78.

61. Danecek, P., et al., *The variant call format and VCFtools.* Bioinformatics, 2011. **27**(15): p. 2156-8.

62. Uitdewilligen, J.G., et al., *A next-generation sequencing method for genotyping-by-sequencing of highly heterozygous autotetraploid potato.* PLoS One, 2013. **8**(5): p. e62355.

63. Venturini, L., et al., *De novo transcriptome characterization of Vitis vinifera cv. Corvina unveils varietal diversity.* BMC Genomics, 2013. **14**: p. 41.

64. Vidotto, M., et al., *Transcriptome sequencing and de novo annotation of the critically endangered Adriatic sturgeon.* BMC Genomics, 2013. **14**: p. 407.

65. Delcher, A.L., S.L. Salzberg, and A.M. Phillippy, *Using MUMmer to identify similar regions in large sequence sets.* Curr Protoc Bioinformatics, 2003. **Chapter 10**: p. Unit 10 3.

66. Wu, Y., et al., *Efficient and accurate construction of genetic linkage maps from the minimum spanning tree of a graph.* PLoS Genet, 2008. **4**(10): p. e1000212.

67. Wilson, R.J., et al., *Complete gene map of the plastid-like DNA of the malaria parasite Plasmodium falciparum.* J Mol Biol, 1996. **261**(2): p. 155-72.

68. Gardner, M.J., et al., *Genome sequence of the human malaria parasite Plasmodium falciparum.* Nature, 2002. **419**(6906): p. 498-511.

69. Volkman, S.K., et al., *A genome-wide map of diversity in Plasmodium falciparum.* Nat Genet, 2007. **39**(1): p. 113-9.

70. Yang, X., et al. *CloG: A pipeline for closing gaps in a draft assembly using short reads*. in *Computational Advances in Bio and Medical Sciences (ICCABS), 2011 IEEE 1st International Conference on*. 2011. IEEE.

71. Tsai, I.J., T.D. Otto, and M. Berriman, *Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps.* Genome Biol, 2010. **11**(4): p. R41.

72. Koren, S., et al., *An algorithm for automated closure during assembly.* BMC Bioinformatics, 2010. **11**: p. 457.

73. Vicedomini, R., et al., *GAM-NGS: genomic assemblies merger for next generation sequencing.* BMC Bioinformatics, 2013. **14 Suppl 7**: p. S6.

74. Zimin, A.V. and others, *Assembly reconciliation.* Bioinformatics, 2008. **24**(1): p. 42-45.

75. Policriti, A., et al., *GAM: Genomic Assemblies Merger.* EMBnet. journal, 2012. **18**(A): p. pp. 50-51.

76. Gnerre, S., et al., *Assisted assembly: how to improve a de novo genome assembly by using related species.* Genome Biol, 2009. **10**(8): p. R88.

77. Schneeberger, K., et al., *Reference-guided assembly of four diverse Arabidopsis thaliana genomes.* Proc Natl Acad Sci U S A, 2011. **108**(25): p. 10249-54.

78. Warren, R.L. and R.A. Holt, *Targeted assembly of short sequence reads.* PLoS One, 2011. **6**(5): p. e19816.

79. Klein, J.D., et al., *LOCAS--a low coverage assembly tool for resequencing projects.* PLoS One, 2011. **6**(8): p. e23455.

80. Ossowski, S., et al., *Sequencing of natural strains of Arabidopsis thaliana with short reads.* Genome Res, 2008. **18**(12): p. 2024-33.

81. Pop, M., et al., *Comparative genome assembly.* Brief Bioinform, 2004. **5**(3): p. 237-48.

82. Altschul, S.F., et al., *Basic local alignment search tool.* J Mol Biol, 1990. **215**(3): p. 403-10.

83. Langmead, B., *Aligning short sequencing reads with Bowtie.* Curr Protoc Bioinformatics, 2010. **Chapter 11**: p. Unit 11 7.