

Assembly of Large Genomes from Paired Short Reads

Benjamin G. Jackson¹, Patrick S. Schnable², and Srinivas Aluru^{1,2}

¹ Department of Electrical and Computer Engineering

² Center For Plant Genomics

Iowa State University, Ames, IA 50011, USA

Abstract. The *de novo* assembly of genomes from high-throughput short reads is an active area of research. Several promising methods have been recently developed, with applicability mainly restricted to the smaller and less complex bacterial genomes. In this paper, we present a method for assembling large genomes from high-coverage paired short reads. Our method exploits large distributed memory and parallelism available on multiprocessor systems to handle memory-intensive phases of the algorithm, effectively allowing scaling to large genomes. We present parallel algorithms to construct a bidirected string graph that is several orders of magnitude smaller than the raw sequence data and to extract features from paired reads. We also present a heuristic method that uses these features to guide the extension of partial graph traversals corresponding to large genomic contigs. In addition, we propose a simple model for error correction and derive a lower bound on the coverage needed for its use. We present a validation of our framework with short reads from *D. melanogaster* and *S. cerevisiae* synthetically generated at 300-fold coverage. Assembly of the *D. melanogaster* genome resulted in large contigs ($n_{50} > 102\text{Kb}$), accurate to 99.9% of the bases, in under 4 hours of wall clock time on a 512-node Blue Gene/L.

1 Introduction

For nearly three decades from its invention, Sanger sequencing - which produces 700 to 1000 base-pair reads - dominated the field of DNA sequencing and genome assembly. New developments in high-throughput short read sequencing are proving a disruptive technology that allows concurrent generation of millions of reads at a significantly lower per base cost, albeit with limitations on read length (25-50 bp typically, with the exception of 454, which can produce 150 bp reads). Several such platforms are available and seeing rapid adoption in the community (454 Life Sciences system [13], Illumina Solexa [1], Applied Biosystems SOLiD [17], and Helicos Biosciences Heliscope [22]).

Short-read technologies were initially aimed at resequencing individuals when a template genome of the species is known. This allows aligning the reads to the reference genome, avoiding a *de novo* assembly, and provides an easy way for biological analysis such as identifying Single Nucleotide Polymorphisms (SNPs).

The Solexa system has been used for resequencing [2][19], identifying repeats [21], and characterizing population diversity [16].

There has been considerable recent interest in the more technically challenging problem of *de novo* genome sequencing. Given the multimillion dollar expense associated with Sanger read based genome sequencing projects, high-throughput technologies offer the only hope of sequencing a much larger number of species. Also, *de novo* assembly is important in cases where significant genomic rearrangements are expected, such as when sequencing multiple inbred lines of the same plant species. In response to these needs, several short read assemblers have recently been developed – ALLPATHS [3], Euler-SR [4], SHARCGS [5], Shorty [8], an assembler by Medvedev *et al.* [14], SSAKE [20], and Velvet [23].

Traditionally, the overlap-layout-consensus paradigm has been the mainstay of Sanger read based sequencing projects, whereby long overlaps between reads provide much of the information to shape the assembly. Though graph-based methods that permit a more global view when resolving repeats have been developed (de Bruijn [9][18] and string graph models [15]), their reach has not extended beyond bacterial genomes due to significant memory requirements.

With short reads eliminating the reliability of read overlaps in predicting genomic co-location, a revival of graph-based methods has underpinned the development of short-read assemblers. However, the memory limitations seen previously are further exacerbated due to the high coverage needed to compensate for shorter read lengths. As a result, short-read *de novo* assembly has been demonstrated on relatively small genome sizes, ranging from single BACs [5][20][23] to bacterial genomes with a few million bases [4][7][14]. Perhaps the largest reported assembly (39 million bases) was produced in 2 days using a workstation with 64 GB memory [3].

In this paper, we present a short-read sequence assembly framework that can successfully assemble large genomes with high coverage. To do so, we rely on parallel algorithms and the large distributed memory afforded by high performance parallel computers for the memory-intensive phases of the assembly. We present parallel methods for constructing a bidirected de Bruijn graph of k -molecules and converting this graph into a bidirected string graph. We present a parallel method for computing features that summarize the distances between paired reads. We also create weak traversal constraints derived from k -mer frequency along each edge. We present an algorithm that finds paths corresponding to contigs by starting with unambiguous long edges as seeds, and then extending each path using heuristic rules that rely on the computed features. As the string graph is many orders of magnitude smaller than the initial sequence data, and there are only a few thousand features per edge, this final phase of assembly can be carried out sequentially for many genomes. Our method advances the state of the art in short-read assembly from the confines of prokaryotic genomes. For example, we demonstrate a 99.9% accurate assembly of the *Drosophila Melanogaster* genome in four hours, with 50% of the genome covered by contigs of length at least 102Kb.

2 Linking Coverage to Error Correction

The two most crucial technical challenges in accurate short read assembly are eliminating sequencing errors and accurately reconstructing genomic repeats. High coverage, randomness in error location, and a low error rate are helpful when dealing with errors. Paired reads obtained by sequencing both ends of fragments of known approximate size provide distance constraints that are instrumental in identifying correct walks in the graph and resolving repeats.

As with other methods, the method we propose is sensitive to errors unless they are identified prior to assembly. In a manner similar to that proposed by Pevzner *et al.* [18], we deal with errors by analyzing k -mer frequency. We wish to find a threshold such that, with high probability, all k -mers with frequency below this threshold are artifacts due to sequencing errors. Correspondingly, all real k -mers should occur at a rate above this threshold. We make the simplifying assumption that the sampling of the genome and the sequencing error are independent, uniform, random processes. For the initial analysis, we ignore the increased frequencies of certain k -mers due to the presence of repeats, but address this subsequently.

Let g the length of the genome, l be the length of each read, r be the substitution error rate per base, and c be the coverage rate per base. We describe a k -mer as a tuple $s = \langle p_1, p_2, \dots, p_k \rangle$. An *edit profile* is the corresponding tuple $\langle c_1, c_2, \dots, c_k \rangle$ with $c_i \in [0, 3]$. The probability $P(c_i = 0)$ is $1 - r$ (i.e., base called correctly), while the probabilities $P(c_i = 1) = P(c_i = 2) = P(c_i = 3) = \frac{r}{3}$ (corresponding to each of the three incorrect base call possibilities).

We are interested in two classes of edit profiles: the identity profile, which has probability $(1 - r)^k$; and the profiles corresponding to a single edit, each with probability $\frac{r}{3}(1 - r)^{k-1}$. We ignore other profiles given their low probabilities for practical values of r . The expected rate at which the identity profile occurs at some location in the genome is $\lambda_p = \left(\frac{c(l-k)}{l}\right) (1 - r)^k$. The expected rate at which a single error edit profile occurs is $\lambda_d = \left(\frac{c(l-k)}{l}\right) \left(\frac{r}{3}\right) (1 - r)^{k-1}$.

The number of times a particular k -mer (the identity profile at a particular position) is seen in the data is a Poisson process, with the expected number of k -mers seen exactly t times given by the equation:

$$\mathcal{C}_t = \left(\frac{\lambda_p^t e^{-\lambda_p}}{t!}\right) g$$

The expected number of times a single base k -mer edit is seen exactly t times is defined similarly:

$$\mathcal{E}_t = \left(\frac{\lambda_d^t e^{-\lambda_d}}{t!}\right) 3kg$$

Given a genome of length g and an error rate r , we wish to find coverage c such that good k -mers can be separated from bad k -mers by some threshold τ with high probability. We create a 3-dimensional plot of \mathcal{C}_t and \mathcal{E}_t given c and

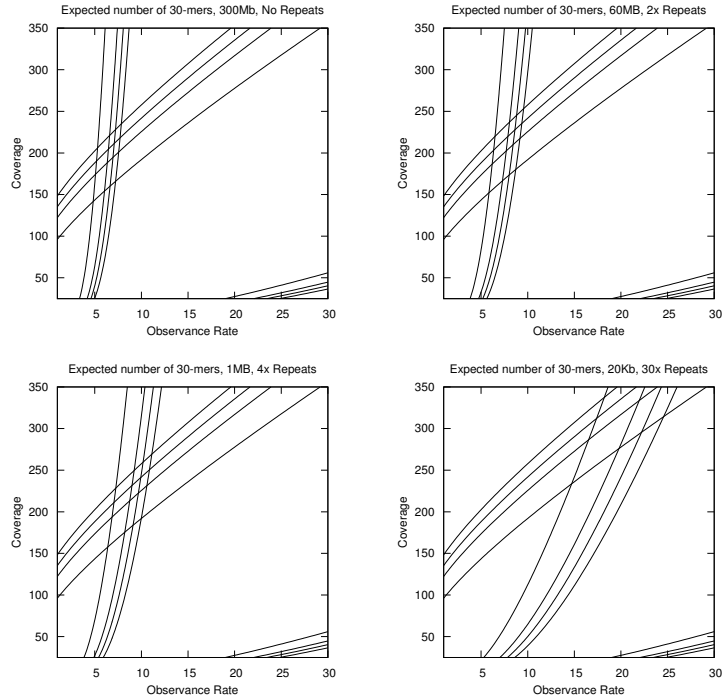


Fig. 1. Contour lines for \mathcal{C}_t and \mathcal{E}_t when plotted against c and t . We plot $\log_{10} \mathcal{C}_t = \{-2, -1, 0, \text{ and } 2\}$ for genome length 300Mb, read length of 40bp, 1% error, and $k=30$. We also plot $\log_{10} \mathcal{E}_t = \{-2, -1, 0, \text{ and } 2\}$ for a hypothetical genome repeat decomposition, superimposed against \mathcal{C}_t . We show in the upper left a plot of \mathcal{C}_t for 300Mb of unique sequence. We show in the upper right a plot for 60Mb of sequence repeated twice. We show in the lower left a plot for 1Mb of sequence repeated 4 times. Finally, we show in the lower right a plot for 20Kb repeated 30 times. These plots indicate that with 1% sequencing error rate, 30-mers can likely be differentiated using a simple threshold method at 250-fold to 300-fold coverage.

τ , as shown in the upper left quadrant of Figure 1. Observe that if the genome is unique, a good separation of 30-mers can be achieved with 200-fold coverage of length 40bp reads with 1% error.

We can update this analysis given the presence of sampling bias and repeats by observing that both can be modeled as non-uniform coverage of some genome with only unique k -mers. We can analyze the k -mers by separating this genome into sets of k -mers with similar coverage. Our task is to find a single threshold that separates real k -mers from errors in all sets simultaneously. As shown in Figure 1, for a genome of length 300Mb, a 1% error rate, and an average read length of 40, we can expect that 300-fold coverage will achieve separation of 30-mers for many repeats.

3 Parallel Bidirected String Graph Construction

Our target architecture is the distributed memory high performance computer; all processors have direct access to local memory, but access to other processors' memory is mediated by an interconnection network. We have previously described a method for constructing a bidirected string graph in parallel[10][11], which we review.

In a bidirected graph, each edge has two directions, one for each endpoint. There are four possible ways of connecting edges for an ordered pair of nodes (u, v) : $u \triangleright \triangleright v$, $u \triangleleft \triangleleft v$, $u \triangleright \triangleleft v$, and $u \triangleleft \triangleright v$. A valid graph traversal requires that endpoint directions be satisfied when passing through a node; if we enter a node on an in arrow, we must exit on an out arrow, and vice versa.

Consider each k -molecule in the data (by considering both strands associated with each k -mer). We call the lexicographically larger of the two strands the positive (or representative) strand, and the other strand the negative strand. Consider a de Bruijn graph, where each k -molecule corresponds to a node in the graph, and two nodes are connected if they share a common $(k - 1)$ -molecule. There are four ways of connecting edges in the de Bruijn graph, and we use a bidirected edge to capture these options, as shown in Figure 3. If the positive strand of the underlying molecule moves into the edge, the corresponding arrowhead points away from the node. Similarly, if the negative strand of the underlying molecule moves into the edge, the corresponding arrowhead points into the node.

We construct the de Bruijn graph edge-by-edge, by looking at all $(k + 1)$ -molecules in the input data. The representative strand of each molecule is stored as a $4(k + 1)$ bit integer in a distributed array. To deal with memory constraints, the sequences are read in stages, updating this representative list at the end of each phase with the frequency of each molecule in the data, using parallel sort [6] as the supporting operation.

Also because of limited memory, data is processed in two phases. In the first phase, we record the frequency of all observed $(k + 1)$ -molecules. After enough data has been considered, all real $(k + 1)$ -molecules will have been observed at least once with high probability. At this point, we continue with the second phase, in which only the frequencies of previously seen $(k + 1)$ -molecules are updated. After all data has been considered, molecules with frequency below the threshold indicated in the previous section are discarded.

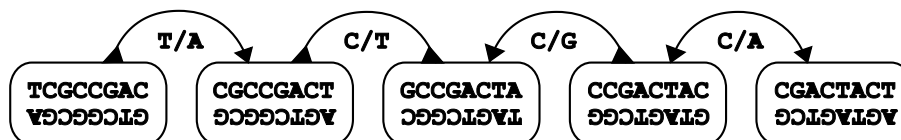


Fig. 2. The four ways in which k -molecule nodes can be connected in the bidirected de Bruijn graph of the data, with the corresponding edge labels that will be used in creating the string graph.

Once all true $(k + 1)$ -molecules are found, the graph can be determined directly by constructing an edge from each representative. This graph is nearly a de Bruijn graph, but there can exist nodes in this graph that are not connected but would be in a true de Bruijn graph, due to the existence of maximal $k - 1$ length repeats in the underlying genome [10]. We will refer to the graph constructed in this manner as a de Bruijn graph despite this subtle difference.

We can label the edges in this graph with two characters, each corresponding to the first base from each strand of the $(k + 1)$ -molecule. We are interested in compacting chains in the de Bruijn graph, as we have described extensively in [11]. We solve this problem by transforming the chain compaction problem to the parallel undirected list ranking problem. During edge compaction, we also concatenate the edge labels, to produce assembled contigs.

The result of this process is a bidirected string graph [15]. The bidirected string graph we create is the lowest order graph homeomorphic with the de Bruijn graph. Each edge in this graph is labeled by some genomic sequence, which may or may not be repeated multiple times in the genome. We arbitrarily assign each edge in the graph a forward direction which will be used during the processing of paired reads. Some traversal of this graph corresponds to the entire assembled genome, although there are many possible traversals.

4 Paired Read Processing

The reads in the sequence data come in pairs, each read pair coming from the two ends of a sheared DNA fragment. Typically fragment lengths fall into a specified range. Our method allows for the consideration of multiple such fragment ranges, which we call fragment types, and assumes that reads are classified accordingly.

In the bidirected string graph $G = \{E, V\}$, edges e_i and e_j correspond to genomic sequences s_i and s_j , each sequence occurring one or more times in the genome. If the genomic distance between these two sequences falls within some fragment range, there will be evidence of these two sequences' relative location in the sequence data. We will summarize this information as a set of features we term *partial $(k + 1)$ -pair clusters*, and use these features to finish assembly.

Definition 1. A **position** in the bidirected graph G is a tuple of the form $p = \langle e, f \rangle$, with $e \in E$, $f \in \mathbb{N}$, $0 \leq f < \|e\|$. The field f corresponds to a position along the edge in its forward direction, indexed from zero.

Note 1. By construction of the string graph, there is a bijection between valid $(k + 1)$ -molecules in the input and the set of all positions in the graph. Therefore we use $p(m)$ to denote the position corresponding to $(k + 1)$ -molecule m , and $p(m).e$ and $p(m).f$ to denote the corresponding fields.

Definition 2. A **read pair** is a tuple of the form $\langle R_1, R_2, z \rangle$, where R_1 and R_2 are the reads and z is the fragment type.

Definition 3. A **$(k + 1)$ -pair** is a tuple of the form $\pi = \langle m_1, m_2, z \rangle$, where m_1 and m_2 are molecules.

Note 2. We use $\lceil z \rceil$ to denote the largest possible distance between observed $(k+1)$ -molecules when reading the ends of a fragment of type z , and $\lfloor z \rfloor$ to denote the smallest possible distance. If z_{min} is the minimum length of fragment type z , z_{max} the maximum fragment length, and l the maximum read length, $\lfloor z \rfloor = z_{min} - 2l + (k+1)$, and $\lceil z \rceil = z_{max} - (k+1)$.

Definition 4. *The set of all $(k+1)$ -pairs in the input is the set $\Pi = \{\pi_1, \pi_2, \dots, \pi_M\}$, with $\langle m_1, m_2, z \rangle \in \Pi$ if and only if there exists some read pair $\langle R_1, R_2, z \rangle$ with m_1 a sub-molecule of R_1 and m_2 a sub-molecule of R_2 .*

Note 3. When we allow duplicates, $M = O(N(l-k)^2)$, where N is the number of reads and l the maximum read length.

Definition 5. *An edge traversal is a tuple of the form $t = \langle e, d \rangle$, with $e \in E$ and $d \in \{F, R\}$, with F corresponding to traversing the edge in the forward direction, and R in the reverse direction.*

Definition 6. *A path is a sequence of edge traversals: $T = \langle t_1, t_2, \dots, t_l \rangle$.*

Note 4. In general, edges in the graph can be traversed multiple times, so there could exist t_i and t_j , $i \neq j$ and $e_i = e_j$. We always assume that paths being discussed are valid walks in the string graph, as described in the previous section.

Consider some $\pi_x = \langle m_{1x}, m_{2x}, z_x \rangle$ and traversal T . Let $\mathcal{L}_x = \{t_i | e_i = p(m_{1x}).e\}$ be the set of edge traversals in T to which m_{1x} maps. Let $\mathcal{R}_x = \{t_j | e_j = p(m_{2x}).e\}$ be the set of all edge traversals to which m_{2x} maps.

Definition 7. *For each $\langle t_i, t_j \rangle \in \mathcal{L}_x \times \mathcal{R}_x$, $i < j$, the observed distance of π_x is:*

$$d(\pi_x, t_i, t_j) = \sum_{h=i+1}^{j-1} \|e_h\| + \sigma_i + \sigma_j$$

$$\sigma_i = \begin{cases} p(m_1).f & \text{if } d_i = R \\ \|p(m_1).e\| - p(m_1).f & \text{if } d_i = F \end{cases}$$

$$\sigma_j = \begin{cases} p(m_2).f & \text{if } d_j = F \\ \|p(m_2).e\| - p(m_2).f & \text{if } d_j = R \end{cases}$$

Definition 8. π_x **supports** T using t_i and t_j if and only if $\lfloor z_x \rfloor \leq d(\pi_x, t_i, t_j)$ and $\lceil z_x \rceil \geq d(\pi_x, t_i, t_j)$.

Definition 9. t_i and t_j are **supported by** Π if and only if there exists some π_x that supports the path using t_i and t_j .

Note 5. We call this support *weak* because the genomic distance between t_i and t_j can differ from the path distance by as much as $\lceil z_x \rceil - \lfloor z_x \rfloor$.

Definition 10. *The maximum distance expectation for t_i and t_j and some fragment type z , denoted by $\lceil (t_i, t_j, z) \rceil$, is calculated as $\min(\lceil z \rceil, \sum_{h=i}^j \|e_h\|)$.*

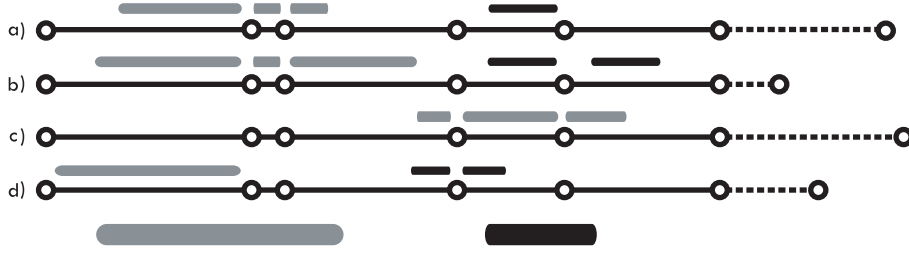


Fig. 3. Some examples of path extension candidates, with $(k + 1)$ -pair clusters for two fragment types shown. In a), we show prototypical strong cluster support. In b), we show $(k + 1)$ -pair cluster support for the extension of a repeat that will occur twice in quick succession in the path. In c), we show an obvious example of lack of support. Finally, in d) we show an example of lack of strong support for the extension, even though the cluster overlaps with expected distance constraints.

Definition 11. *The minimum distance expectation for t_i and t_j and some fragment type z , denoted by $\lfloor (t_i, t_j, z) \rfloor$, is calculated as $\max(\lfloor z \rfloor, \sum_{h=i+1}^{j-1} \|e_h\|)$.*

In general, multiple $(k + 1)$ -pairs with the same fragment type can support a pair of edge traversals on a path. Moreover, if the path is correct, we would expect that, for all z_h , support for much of the range $[\lfloor (t_i, t_j, z_h) \rfloor, \lceil (t_i, t_j, z_h) \rceil]$ to be found in the data, assuming the edges are not very short. We wish to formalize this support expectation.

Definition 12. *A $(k + 1)$ -pair cluster is a set of observed distances for t_i, t_j , and z . We summarize a cluster using the range $\alpha(t_i, t_j, z) = [min, max]$, with min being the minimum observed distance in the cluster and max the maximum observed distance.*

We construct the $(k + 1)$ -pair clusters starting from all single element sets taken from Π and proceeding in two phases of merging. In the first phase, we perform single linkage clustering, merging two sets $\alpha_x(t_i, t_j, z)$ and $\alpha_y(t_i, t_j, z)$ if and only if $(max_x + R > min_y) \wedge (min_x - R < max_y)$, for some parameter R . In the second stage, we order all clusters by min , and then, considering all consecutive pairs $(\alpha_x(t_i, t_j, z), \alpha_y(t_i, t_j, z))$ in this ordered set, merge if $max_y - min_x < \lceil z \rceil$.

Definition 13. *t_i and t_j are strongly supported by a $(k + 1)$ -pair cluster $\alpha(t_i, t_j, z)$ if $\alpha_{min} < \lfloor (t_i, t_j, z) \rfloor + T$ and $\alpha_{max} > \lceil (t_i, t_j, z) \rceil - T$, with T a sensitivity parameter.*

The preceding definition is carefully chosen to allow for an edge to be strongly supported even if, for example, t_i is a repeat and occurs at multiple distances from t_j in the genome. For a visual intuition behind the strongly supported definition, see Figure 3.

In practice, we wish to be able to answer the question of whether t_i and t_j are strongly supported without having to consider the entire set II when analyzing a particular path, but instead preprocess the raw paired reads to extract the necessary features. This needs to be done without any *a priori* knowledge of the nature of the eventual traversal T . In other words, we do not know either the distance between pairs of edges or their relative orientations at the time of summarization.

We achieve this goal by calculating, for each tuple $\langle e_i, e_j, z \rangle$, the ranges of the partial sum $\sigma_i + \sigma_j$ corresponding to each $(k + 1)$ -pair cluster. As we don't know the relative orientation of the edges at the time of summation, we track the range of $\sigma_i + \sigma_j$ for all four possible orientations of edges, using ff_{min} and ff_{max} to denote this range when e_i and e_j are traversed forwards, with fr_{min} , fr_{max} , rf_{min} , rf_{max} , rr_{min} , and rr_{max} denoting the ranges of other orientations.

Definition 14. A **partial $(k + 1)$ -pair cluster** is a summarization of a set of observed partial sums $\sigma_i + \sigma_j$ for edges e_i , e_j and fragment type z , denoted as $\hat{\alpha}(e_i, e_j, z) = \langle \text{rf}_{min}, \text{rf}_{max}, \text{ff}_{min}, \text{ff}_{max} \rangle$.

Note 6. The value for rr_{min} can be calculated as $\|e_i\| + \|e_j\| - \text{ff}_{max}$, and rr_{max} , fr_{min} , and fr_{max} can be calculated similarly.

Given a traversal T with edges t_i and t_j , we can calculate the $(k + 1)$ -pair clusters $\alpha_x(t_i, t_j, z)$ corresponding to partial $(k + 1)$ -pair cluster $\hat{\alpha}_y(e_i, e_j, z)$. If $t_i.f = F$ and $t_j.f = F$, $\text{min}_x = \text{ff}_{min_y} + \sum_{h=i+1}^{j-1} \|e_h\|$ and $\text{max}_x = \text{ff}_{max_y} + \sum_{h=i+1}^{j-1} \|e_h\|$. The other orientations of t_i and t_j are handled similarly.

We will now describe a parallel algorithm for computing all partial $(k + 1)$ -pair clusters from II :

1. Assume that we have, for each $(k + 1)$ -molecule m , a mapping to that molecule's corresponding graph position, stored as a distributed tuple array of the form $\langle m, e_{ID}, f, l \rangle$, where l is the length of the edge identified by e_{ID} , and f is the forward position, as described previously.
2. Let C be the a distributed tuple array containing tuples of the form $\langle e_{ID_i}, e_{ID_j}, z, \text{ff}_{min}, \text{ff}_{max}, \text{rf}_{min}, \text{rf}_{max} \rangle$, corresponding to partial $(k + 1)$ -pair clusters. C is initially empty.
3. To overcome memory limitations, we process the paired reads in R stages. In each stage we process an $\frac{N}{R}$ subset of the data, $\frac{N}{Rp}$ per processor, if p is the number of processors.
 - (a) For each read pair in the data of the form $\langle R_1, R_2, z \rangle$, create $(k + 1)$ -molecule tuples $\langle m_i, m_j, z \rangle$ for all possible $(k + 1)$ -molecule combinations.
 - (b) Distribute these tuples, based on element m_i , to those processors that have the corresponding $(k + 1)$ -molecule tuple.
 - (c) Combine the tuples $\langle m_i, m_j, z \rangle$ and $\langle m_i, e_{ID_i}, f_i, l_i \rangle$ to the form the tuple $\langle m_j, z, e_{ID_i}, f_i, l_i \rangle$.
 - (d) Distribute the tuples, based on element m_j , to those processors that have the corresponding $(k + 1)$ -molecule tuple.

- (e) Combine the tuples $\langle m_j, e_{ID_j}, f_j, l_j \rangle$ and $\langle m_j, z, e_{ID_i}, f_i, l_i \rangle$ to form a partial $(k+1)$ -pair cluster $\langle e_{ID_i}, e_{ID_j}, z, \text{ff}_{min}, \text{ff}_{max}, \text{rf}_{min}, \text{rf}_{max} \rangle$ with $\text{ff}_{min} = \text{ff}_{max} = l_i - f_i + f_j$ and $\text{rf}_{min} = \text{rf}_{max} = f_i + f_j$.
 - (f) Merge these partial $(k+1)$ -pairs with the array C .
 - (g) Sort C , using e_{ID_i} as primary key, e_{ID_j} as secondary key z as tertiary key, and finally by ff_{min} , forcing all clusters with equivalent (e_{ID_i}, e_{ID_j}, z) onto the same processor.
 - (h) For each bucket of partial $(k+1)$ -pair clusters with equivalent (e_{ID_i}, e_{ID_j}, z) , merge partial $(k+1)$ -pair clusters in accordance with the single linkage merging described above, updating all minimums and maximums. As the clusters are sorted by ff_{min} , this can be done using a single pass through the array on each processor.
4. After all stages have completed, consider each bucket of partial $(k+1)$ -pair clusters with equivalent (e_{ID_i}, e_{ID_j}, z) , and merge partial $(k+1)$ -pair clusters in accordance with the phase two merging rule described above, updating all minimums and maximums. As the clusters are sorted by ff_{min} , this can be done using a single pass through the array on each processor.
 5. Write the resulting partial clusters.

4.1 Bidirected Graph Traversal

Given the partial $(k+1)$ -pair clusters, we wish to find valid walks through the bidirected string graph. We will describe the process using $(k+1)$ -pair clusters, as it is more natural to do so, and these can be derived from the partial $(k+1)$ -pair clusters for any path T .

We assign to each edge e in the graph an expected traversal bound $b(e)$ by analyzing the coverage seen along that edge. When traversing the graph, we keep track of the number of times an edge has been traversed as $c(e)$, and use this information in conjunction with the bound to choose between ambiguous options. Additionally, we restrict traversal to edges with $c(e) < 2b(e)$. Initially $c(e) = 0$ for all edges.

Our method for traversing the graph is one of path extension. Given a likely partial traversal of the graph as a path $T = \langle t_1, t_2, \dots, t_l \rangle$ with total length greater than the maximum fragment size, we can determine, by looking at the structure of the string graph, a set of possible edge traversals that can serve as extensions of this path: $E = \{t'_1, t'_2, \dots, t'_h\}$, $t'_j = \langle e_j, d_j \rangle$. We describe a heuristic method for choosing the best extension from E by choosing the candidate with the most strong support among the $(k+1)$ -pair clusters.

Specifically, consider some extension t'_j . Let T' be the path created by extending T with t'_j .

Definition 15. *The expected support for t_i, t'_j , and fragment type z_v is:*

$$\gamma_{ijv} = \begin{cases} \hat{\gamma}_{ijv} & \text{if } (\lfloor (t_i, t'_j, z_v) \rfloor < (\lceil z_v \rceil - B)) \wedge (\lceil (t_i, t'_j, z_v) \rceil > (\lfloor z_v \rfloor + B)) \\ \hat{\gamma}_{ijv} & \text{if } t_i \text{ and } t'_j \text{ are strongly supported by some } \alpha(t_i, t'_j, z_v) \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\gamma}_{ijv} = \lceil (t_i, t'_j, z_v) \rceil - \lfloor (t_i, t'_j, z_v) \rfloor$$

Where B is a parameter.

Definition 16. The **observed support** for t_i, t'_j and fragment type z_v is:

$$\omega_{ijv} = \begin{cases} \hat{\omega}_{ijv} & \text{if } t_i \text{ and } t'_j \text{ are strongly supported by some } \alpha(t_i, t'_j, z_v) \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\omega}_{ijv} = \min(\lceil(t'_i, t'_j, z_v)\rceil, \alpha.max) - \max(\lfloor(t'_i, t'_j, z_v)\rfloor, \alpha.min)$$

Definition 17. The **extension score** of a candidate t'_j (for $\sum_{i,v} \gamma_{ijv} > 0$) is defined as:

$$0 \leq S(t'_j) = \frac{\sum_{i,v} \omega_{ijv}}{\sum_{i,v} \gamma_{ijv}} \leq 1$$

We say that an extension t'_j is *unambiguous* if $S(t'_j) > 0$ and, for all w with $0 < w \leq h$ and $w \neq j$, $\frac{S(t'_w)}{S(t'_j)} < D$ (for some specificity parameter D). If there exists an extension t'_j that is unambiguous, then we append t'_j to the path and continue with traversal. If t'_j does not exist, we consider only those edges with $b(t'_j) > c(t'_j)$, and look for the existence of an unambiguous extension \hat{t}'_j . If neither t'_j nor \hat{t}'_j exist, we stop extension.

All that remains is to describe how we seed the paths. All edges e with $\|e\|$ greater than the maximum fragment size and $c(e) = 0$ can serve as a seed. As it is not obvious, for example, that really long edges are better than moderately long edges as starting points, we simply choose from candidate starting points in increasing order of edge identifier, until none remain.

5 Experimental Results

We experimentally evaluated the proposed method using synthetic data generated from previously assembled genomes, downloaded from the NCBI FTP server. For each genome, we cleaned any ambiguities from the data and concatenated any contigs from the same chromosome, in the order presented in the finished FASTA file. In this way, we generate a contiguous sequence for each chromosome even though the actual data may contain a large number of contigs, possibly scaffolded. From the input chromosomes, we then generated fragments and sampled 30bp to 50bp paired short reads from the ends of the fragments. We used a 0.9% substitution rate and 0.1% deletion rate, for a total error rate of 1%. Fragment sizes were based upon two hypothetical experimental protocols. Protocol I consisted of two fragment types, $\{900 \pm 100, 4300 \pm 600\}$. Protocol II consisted of five fragment types $\{330 \pm 30, 660 \pm 60, 1100 \pm 100, 2200 \pm 200, 4400 \pm 400\}$. All genomes were sampled at 300-fold coverage.

For testing the assembler, data was generated on a workstation, using the parameters described above. This data was then transferred to a 512-node Blue Gene/L system with 512 MB memory per node, at which point the parallel

Organism	Size	Max	n50	n75	n90	Count	Mis	Cov
<i>E. coli</i>	5.4	224	85	43	10	91	0.2	90.0
<i>S. cerevisiae</i>	12.2	225	71	34	11	225	0.8	90.1
<i>C. pneumoniae</i>	1.0	867	867	867	132	2	0.0	99.9
<i>S. pneumoniae</i>	2.1	321	137	92	77	19	0.0	95.5
<i>E. coli</i>	5.4	378	231	104	42	42	0.5	94.0
<i>S. cerevisiae</i>	12.2	290	107	75	25	148	0.8	94.1
<i>D. melanogaster</i>	120.3	855	102	43	12	1,687	1.5	91.2

Table 1. Assembly quality for five organisms. The first group shows results for sequences using protocol I, as described in the text. The second group was assembled from data matching protocol II. In order, we show the size of the genome in megabases; the maximum, $n50$, $n75$, and $n90$ lengths, all in kilobases; the number of contigs with length $> 10\text{Kb}$, the number of missassemblies per megabase, and percentage of the genome covered by these contigs at $> 99.9\%$ identity.

phases of the software were run to produce the intermediate string graph and features from paired reads. These results were transferred back to a workstation for production of contigs using bidirected graph traversal. For *Drosophila*, this process took ~ 50 minutes for data transfer (depending on network congestion), ~ 100 minutes for the parallel phases, and ~ 20 minutes for the remaining. In the local processing phase of the algorithm, the running time was dominated by file I/O.

Producing long and correctly assembled contigs that cover most of the genome is the hallmark of a good assembler. We use the nX length measure, which is the maximum length l such that X percent of the genome is covered by contigs with length at least l . For validation, we used MUMmer 3.20 [12] to align the finished contigs back to the reference. The results are presented in Table 5. We present results on three bacterial genomes as a reference point for comparison with other work on short read assembly. In addition, we present results on *S. cerevisiae* and *D. melanogaster*. For each genome, we present the length of the maximum contig generated, along with $n50$, $n75$ and $n90$ lengths. We also present the number of contigs with length $> 10\text{Kb}$, the percentage of the genome that is covered by these contigs with at least 99.9% identity, and the number of large-scale missassemblies per megabase. Generally, four out of five assembled contigs aligned perfectly to the reference.

6 Discussion and Conclusions

In this paper, we presented a graph-based method for assembly of large genomes from short reads. Our method can scale to large genomes using distributed memory multiprocessors. It can naturally handle reads of multiple lengths (short reads from one or more platforms, Sanger reads, or a mixture), and multiple fragment

sizes used in paired read generation. Experimental results show that our method produces large ($N50 > 100\text{Kb}$), high quality ($>99.9\%$ correct) contigs from synthetic data sets generated with 1% error in a few hours of wall clock time using a 512-node Blue Gene/L. We have validated our method by generating synthetic short sequence data from *Drosophila* (120Mb), Yeast (12Mb), and a number of bacterial genomes.

Error discovery is an essential part of any *de novo* short read assembly. While we have identified errors by counting k -mer frequency for relatively large k , other promising methods for finding errors have been proposed. Zerbino *et al.* [23] describe motifs in the graph that are likely to be due to errors, such as alternate edges between two nodes, with one edge at much lower coverage. Chaisson *et al.* [4] described methods for preserving sequences with errors by finding good edits. We are interested in exploring ways to achieve robust error correction using parallel computers, as this will be necessary to achieve good results at lower sequence coverage rates.

There is good potential for using the large edges in the graph to decompose the graph traversal into multiple independent problems. We plan to develop parallel algorithms for this final assembly stage, leading to the creation of a fully parallel assembly pipeline. This may be necessary for scaling to very large genomes such as human, mouse and maize. Using our assembler, it has become easy to experiment with the number and types of paired read lengths. A parameterized study with multiple experimental protocols, and different sampling errors and coverage variance, can be conducted to help plan future *de novo* genome sequencing projects.

Acknowledgements

We thank Chad Brewbaker, Scott Emrich, Xiao Yang, and Jaroslaw Zola for their input and feedback. This project was supported in part by the National Science Foundation under CNS-0521568, DBI-0527192, and CCF-0431140, and by the Plant Sciences Institute Innovative Research Grants program.

References

1. S. Bennet. Solexa ltd. *Pharmacogenomics*, 5(4):433–438, 2004.
2. D.R. Bentley, S. Balasubramanian, H.P. Swerdlow, and G.P. Smith. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456:53–59, 2008.
3. J. Butler, I. MacCallum, M. Kleber, I.A. Shlyakhter, M.K. Belmonte, E.S. Lander, C.N. Nusbaum, and D.B. Jaffe. ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Research*, 18:810–820, 2008.
4. M.J. Chaisson and P.A. Pevzner. Short fragment assembly of bacterial genomes. *Genome Research*, pages 18:324–330, 2008.
5. J.C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research*, 17:1697–1706, 2007.

6. D.R. Helman, J. Ja'Ja', and D.A. Bader. A new deterministic parallel sorting algorithm with an experimental evaluation. *Journal of Experimental Algorithms*, 3:4, 1998.
7. D. Hernandez, P. Francois, L. Farinelli, M. Osteras, and J. Schrenzel. De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*, 18:802–809, 2008.
8. S. Hossain, N. Azimi, and S. Skiena. Crystallizing short-read assemblies around lone Sanger reads. *Bioinformatics*, 2009.
9. R.M. Idury and M.S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2:291 – 306, 1995.
10. B.G. Jackson and S. Aluru. Parallel construction of bidirected string graphs for genome assembly. In *Proceedings of the International Conference on Parallel Processing*, pages 346–353, 2008.
11. B.G. Jackson, P.S. Schanble, and S. Aluru. Parallel short sequence assembly of transcriptomes. *BMC Bioinformatics*, 10:S14, 2009.
12. S. Kurtz, A. Phillippy, A.L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S.L. Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5, 2004.
13. M. Margulies and M. Egholm. Genome sequencing in open microfabricated high density picoliter reactors. *Nature*, 437(7054):376–380, 2005.
14. P. Medvedev and M. Brudno. Ab initio whole genome shotgun assembly with mated short reads. In *Lecture Notes in Computer Science*, volume 4955, pages 50–64, 2008.
15. E.W. Myers. The fragment assembly string graph. *Bioinformatics*, 21:ii79–ii85, 2005.
16. S. Ossowski¹, K. Schneeberger¹, R.M. Clark, C. Lanz, N. Warthmann, and D. Weigel. Sequencing of natural strains of *arabidopsis thaliana* with short reads. *Genome Research*, preprint, 2008.
17. V. Pandey, R.C. Nutter, and E. Prediger. *Applied Biosystems SOLiD System: Ligation-Based Sequencing*. Wiley, 2008.
18. P.A. Pevzner, H. Tang, and M.S. Waterman. Fragment assembly with double-barreled data. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.
19. J. Wang, W. Wang, R. Li, and Y. Li. The diploid genome sequence of an Asian individual. *Nature*, 456:60–65, 2008.
20. R.L. Warren, G.G. Sutton, S.J.M. Jones, and R.A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23:500–501, 2007.
21. T. Wicker, A. Narechania, F. Sabot, G.T.H. Vu, A. Graner, D. Ware, and N. Stein. Low-pass shotgun sequencing of the barley genome facilitates rapid identification of genes, conserved non-coding sequences and novel repeats. *BMC Genomics*, 9:518, 2008.
22. Business Wire. Helicos biosciences enters molecular diagnostics collaboration with renowned research center to sequence cancer-associated genes. *Genetic Engineering and Biotechnology News*, 2008.
23. D. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18:821–829, 2008.