Ab Initio Whole Genome Shotgun Assembly With Mated Short Reads

Paul Medvedev[1] and Michael Brudno[1,2]

[1]Department of Computer Science and
[2]Donnelly Centre for Cellular and Biomolecular Research
University of Toronto, Canada
{pashadag,brudno}@cs.toronto.edu

Abstract. Next Generation Sequencing (NGS) technologies are capable of reading millions of short DNA sequences both quickly and cheaply. While these technologies are already being used for resequencing individuals once a reference genome exists, it has not been shown if it is possible to use them for *ab initio* genome assembly. In this paper, we give a novel network flow-based algorithm that, by taking advantage of the high coverage provided by NGS, accurately estimates the copy counts of repeats in a genome. We also give a second algorithm that combines the predicted copy-counts with mate-pair data in order to assemble the reads into contigs. We run our algorithms on simulated read data from *E. Coli* and predict copy-counts with extremely high accuracy, while assembling long contigs.

1 Introduction

The problem of genome assembly has perhaps been more controversial than any other topic within computational biology, leading to alternative approaches to genome sequencing and the creation of two human genomes. Initially, a BAC-by-BAC approach to genome sequencing was favored for constructing longer genomes. While this approach was much more expensive than the alternate whole genome shotgun method, it was considered unlikely that *ab initio* whole genome shotgun assembly was feasible. The development of effective shotgun assembly algorithms capable of assembling a mammalian genome, such as the Celera assembler[16] and Arachne[3], has revolutionized sequence assembly, allowing large genomes to be sequenced much cheaper than was previously thought possible. Currently, the field of genome sequencing is undergoing another major change, with the development of Next Generation Sequencing (NGS) technologies, such as Solexa, 454 and AB SOLiD. While the new technologies can currently yield reads only 25-200 basepairs long, they dramatically reduce the cost of sequencing per nucleotide and significantly speed up data acquisition, with nearly 1 billion nucleotides sequenced in one run (2-3 days) on a Solexa machine. While the novel technologies have already made great improvements to the problem of resequencing (the determination of the genomes of various individuals once the initial, reference, genome has been built), it has not been shown whether very short reads can be used for *ab initio* genome sequencing – the determination of a completely unknown genome.

1.1 Background

One of the original approaches to genome assembly was to find the shortest common superstring of the reads, that is, to assemble a genome with minimal length. The problem of modeling genome assembly in this way is that most genomes have repeats – multiple identical, or nearly identical, stretches of DNA – while the shortest solution would include each of these repeats only once in the assembled genome. This problem is known as over-collapsing the repeats. One way of addressing this problem is to build representative strings or structures for each repeat and allow the assembly algorithm to use these multiple times. This intuition led to the development of graph-theoretic methods for sequence assembly, where the edges of the graph "spell" some string, and by walking the edges of the graph it is possible to recreate the genome.

In their EULER assembler[19], Pevzner, Tang and Waterman had the insight that by dividing the reads into shorter k-long stretches (called k-mers), all of the instances of a repeat collapse into a single set of vertices. They represent each read as a walk on a de Bruijn graph, and search for a superwalk that contains all the reads. This approach was later expanded to use A-Bruijn graphs [18], where the initial subdivision into k-mers is not necessary. Myers introduced an alternative model of sequence assembly, using a string graph [15]. Instead of dividing the reads into k-mers, the algorithm starts by building an overlap graph – a graph where vertices correspond to reads and edges correspond to overlaps. Through the process of removing redundant edges, he is able to classify all edges as either unique, required or optional, and the goal of the assembly is to find the shortest walk which respects all the edge constraints.

Because walks on graphs can be elegantly defined using the concept of balance around vertices (each vertex must be entered and left an equal number of times), network flow methods have been suggested for genome assembly. Though network flow alone is not able to resolve the problem of long repeats, it is able to estimate the number of times a read appears in the genome (its copy-count). In the de Bruijn graph formulation, Pevzner and Tang [17] formulate the problem of determining copy-counts as the minimum cost circulation problem. Myers suggests a similar method to determine the copy-counts in the context of a string graph [15]. However, he augments Pevzner and Tang's approach by placing constraints on the copy-counts prior to solving the flow. As in the Celera assembler, Myers determines whether a contig (represented by an edge) is present uniquely in the genome by modeling the reads on a contig as a Poisson arrival process and calculating the probability that the arrival rate for an edge is twice as high as for the genome as a whole. If this probability is low ($p < 10^{-6}$), the edge (contig) is labeled unique, and the flow through this edge is set to be one. Another kind of constraint is placed on every edge that has an interior vertex. Since it must be traversed at least once if the read corresponding to the interior vertex is to take part in the reconstruction of the genome, the flow is constrained to be at least one on this edge.

Network flow techniques alone are insufficient to assemble a genome in the presence of long repeats which are not spanned by any single read. One of the key pieces that has allowed for whole genome shotgun assembly of mammalian genomes are matepairs – pairs of reads which come from opposite strands, at an approximately known distance in the source genome. Matepairs can be generated by taking a piece of DNA of a known size (called an insert) and generating reads from its two ends. Matepairs allow for the spanning of repeats, allowing the assembler to join together long genomic regions even in the presence of a repeat which is not spanned by any read. The typical approach is to build initial contigs (chains of edges in the overlap graph), and then attempt to join them using information from the matepairs. An alternative approach was demonstrated by Pevzner and Tang in the double-barreled version of the EULER program (EULER-DB[17]). They search for all paths in the de Bruijn graph connecting the two reads of a matepair. If there exists only one with a length approximately equal to the length of the insert, it is replaced by a direct edge. This approach has the disadvantage that it requires an algorithm to find all paths of (approximately) a given length between two nodes, which is a difficult computational problem, and scales poorly with the size of the de Bruijn graph.

Sequence assembly using NGS data is a rapidly developing area. Several methods have been recently suggested for *ab initio* sequencing using short reads; many of these appeared after this paper was submitted. We briefly describe these here. SSAKE [20] is an assembler that uses a simple algorithm for building contigs by greedily extending existing overlaps. VCAKE [12] extended SSAKE to work with error-prone, rather than perfect, data. Another approach based on elongating existing contigs is the SHARCGS assembler [6]. The Shorty assembler [11] uses a de Bruijn graph approach in combination with matepairs to assemble a small bacteria – the 600 Kb *Mycoplasma genitalium.* Chaisson and Pevzner [5] have adapted EULER to use short reads. Their approach shows high accuracy and contig sizes for the slightly longer (120 bp) reads generated by the 454 sequencers. They also use matepair information in a manner identical to the EULER-DB algorithm. Another promising, though yet unpublished, tool is the Velvet assembler [21].

All of the previous work on genome assembly shares a major assumption: the goal of the assembly problem is to minimize the length of the genome. While parsimony is usually used to justify this assumption, it is well-known that repeats are ubiquitous in eukaryotic genomes, and even bacterial genomes have sections that are present multiple times. Because of over-collapsing, any repeating region of length longer than the read length may be underrepresented in the assembled genome. For read lengths of 25 nucleotides, which is what we study in this paper, the number of such repeats is very large. We therefore propose an alternate optimization criteria, as we describe below.

1.2 Contributions

In this paper, we introduce two new methods of genome assembly that are tailored specifically to short read data. First, we believe that the overall goal of an assembler should be not to minimize the length of the genome, but to maximize the likelihood that the genome was the source of the various reads. Unlike the case of sequencing by hybridization, where the only available information is whether a certain k-mer is present in the genome, whole genome shotgun sequencing samples the genome, and hence k-mers that are present more often in the genome are more likely to be sampled. For an individual read, however, Sanger style sequencing does not have sufficient coverage to take full advantage of these frequencies. The greatest advantage that the NGS technologies give is high coverage – on a single run a bacterial genome can get as much as 250x coverage. This number makes it possible to not only determine whether a particular read is present in a genome, but also to statistically estimate its copy-count. We formulate the problem of genome assembly as maximizing the likelihood of the observed read frequencies, rather than minimizing the length of the genome. This problem can be formulated as a minimum cost bidirected flow (biflow) problem with convex costs, and we show that it can be effectively solved with a generic flow solver for the case of bacterial genomes, achieving copy-counts that are accurate more than 99.99% of the time.

Second, to improve the lengths of the assembled contigs, we introduce a novel technique for taking advantage of matepair information. Our method is based on the simple Dijkstra's shortest path algorithm. In contrast to EULER-DB, we do not search for all the paths between mated reads, but rather, we search only for the existence of short paths between some pairs of reads. Because the paths we search for are bounded by a small length that is independent of the genome size (the maximum variation in the insert size), our algorithm scales extremely well for large genomes and high coverage.

2 Methods

In Sections 2.1 through 2.4, we present the steps of our copy-count prediction algorithm. In Section 2.5, we give our algorithm for repeat resolution using matepairs.

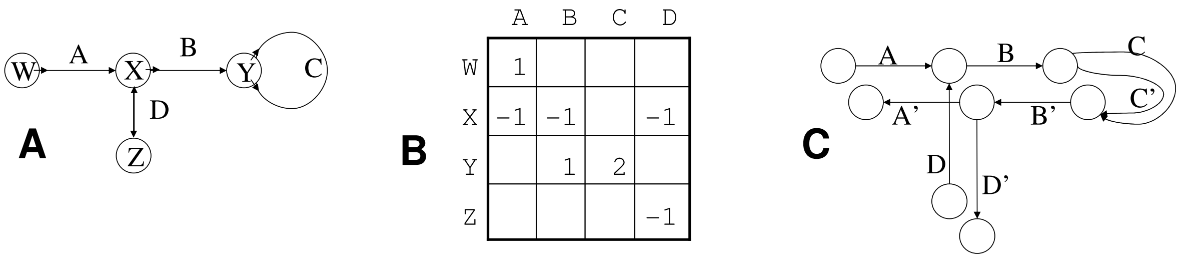
2.1 Building the Transitively Reduced Bidirected Overlap Graph

Our algorithm models the double-stranded nature of DNA during genome assembly by using the elegant bidirected graph framework. Bidirected graphs are a generalization of directed graphs that were introduced by Edmonds in [7]. A bidirected graph is different from a directed graph in that the edges have orientations on each of the ends, rather than on the whole edge. This leads to three types of edges:

- edges with one arrow pointing into its vertex and the other pointing out of it vertex.
- edges with both arrows pointing out of their respective vertices.
- edges with both arrows pointing into their respective vertices.

A walk in a bidirected graph is defined as a a sequence $x_1, e_1, \ldots, e_{k-1}, x_k$ where e_i is an edge incident to vertices x_i and x_{i+1}, and for all $2 \leq i \leq k-1$, e_{i-1} and e_i have opposite orientations at x_i. Informally, if a walk enters a node on an in-edge, then it must exit on an out-edge, and if it enters an on out-edge, then it must exist and an in-edge (see Figure 1A for an example).

Bidirected overlap graphs were first introduced by Kececiouglu [13]. An overlap graph is a graph where each vertex corresponds to a read and each edge corresponds to an overlap between reads. In a bidirected overlap graph, each vertex corresponds to a double-stranded read (the read and its reverse complement), and each edge corresponds to one of the three ways that double-stranded reads can overlap each other. Any walk can be traversed in either of two directions, so just like a walk in a directed overlap graph spells a string that contains each of the reads, a walk in a bidirected overlap graph spells a double-stranded string



	A	B	C	D
W	1			
X	-1	-1		-1
Y		1	2	
Z				-1

Fig. 1. A. This is an example of a bidirected graph. The sequence $W, A, X, B, Y, C, Y, B, X, D, Z$ is a walk, while W, A, X, D, Z is not. **B.** The corresponding edge incidence matrix, with the zero entries ommited. **C.** This is the associated monotonized directed graph.

that contains each of the double-stranded reads. Thus, the original double-stranded genome corresponds to a walk in the bidirected overlap graph that visits every vertex at least once (assuming error-free reads and complete coverage). For a more extended discussion of bidirected graphs in general and bidirected overlap graphs in particular, we refer the reader to [14].

The first step of our assembly algorithm is to build a bidirected overlap graph. We add an edge between two reads if they overlap by at least o_{min} characters, where o_{min} is a parameter to our algorithm. We then perform transitive edge reduction, where we remove any overlap that is spelled by two shorter overlaps. This procedure is identical to the one described in [15]. While the set of possible double-stranded strings spelled by the graph remains unchanged, the reduction drastically reduces the number of edges. The result is what we refer to as the transitively reduced bidirected overlap graph.

2.2 Convex Min-Cost Biflow

Given the (transitively reduced bidirected) overlap graph as constructed above, we now describe how to use convex min-cost biflow to estimate the copy-counts of each read. The copy-count of a read is the number of times it appears in the original genome.

Let $G = (V, E)$ be a bidirected graph. Let $l : E \to \mathbb{N}$ and $u : E \to \mathbb{N}$ be lower and upper bounds associated with the edges. The function $f : E \to \mathbb{N}$ is called a flow if for every edge, $l(e) \leq f(e) \leq u(e)$, and for every vertex, the flow along the in-edges is the same as the flow along the out-edges. Given a bidirected graph with lower and upper bounds on the edges and a cost $c_e \in \mathbb{R}$ associated with each edge, the (linear) min-cost biflow problem is to find a flow that minimizes $\sum c_e f(e)$. We discuss algorithms for this problem in Section 2.4.

We take advantage of two additional variations on the min-cost biflow problem. The first allows for having lower and upper bounds for the flow going through each vertex, as well as adding a cost function on the vertices as well as the edges. Such a problem can be reduced to the min-cost biflow problem as follows. Take every vertex v and split it into two vertices v^+ and v^-. Reconnect any edge that was pointing into v to be pointing into v^-. Similarly, reconnect any edge that was pointing out of v to be pointing out of v^+. As a final step, add an edge from v^- to v^+. Now, assign any lower/upper bounds, as well as any costs, associated with v to the edge from v^- to v^+. After repeating this procedure for every vertex, a flow on the transformed graph corresponds to a flow on the original graph, and vice-versa. This transformation is based on a similar transformation on directed graphs [1].

The second equivalent variation is called the convex min-cost biflow problem. Here, the cost c_e associated with an edge e is no longer a real number but rather a convex function $c_e : \mathbb{N} \to \mathbb{R}$, and the goal is to minimize

$\sum_e c_e(f(e))$. Such a minimization function is called *separable convex* because it is a sum of convex functions on each of the variables, independently. In the directed case, this problem is polynomially equivalent to the linear min-cost flow problem by modeling each convex function with piecewise-linear approximations. The same reduction holds in the bidirected case.

Before defining our flow problem, we make a modification to the overlap graph by adding a supersource and supersink to the graph. This is the standard way to convert from a flow to a circulation problem.For a thorough discussion of this method, as well as for descriptions and proofs of the above reductions, we refer the reader to a text on network flow, e.g. [1].

In our assembly algorithm, we define a convex min-cost biflow problem on the modified transitively reduced bidirected overlap graph, with bounds and costs on both the edges and the vertices. Each vertex has a lower bound of 1 since it represents a read that must be present in the genome at least once. All other lower bounds are 0 and all upper bounds are infinity. We specify convex costs for the vertices, which we describe in the next subsection, and add prohibitively large costs to the edges from/to the supersource/sink so that their usage is minimized. Next, we solve for the flow (which we describe in detail in Section 2.4). Since any flow can be decomposed into a collection of walks, our flow represents a (non-contiguous) assembly of the genome, and the flow going through each vertex represents the number of time the read is present in the assembly.

2.3 Maximizing the Global Read-Count Likelihood

Let G be a circular genome of length $N(G)$, and let g_i denote the number of times the k-mer i appears in G. Probabilistically, the dataset of n reads corresponds to a set of outcomes from n independent trials. In each trial, a position is uniformly sampled from G and the outcome of the trial is the k-mer beginning at that position. For a given i, the probability that the outcome of a single trial is i is simply $\frac{g_i}{N(G)}$. Let the random variable X_i denote the number of trials whose outcome is i. There are 4^k such variables, and when considered independently of each other, they each follow the binomial distribution. When taken together, their joint distribution is exactly the multinomial distribution, given by

$$\mathrm{P}[X_1 = x_1,\, X_2 = x_2,\, \ldots,\, \text{and } X_{4^k} = x_{4^k}] \;= \frac{n!}{\prod x_i} \prod_i \left(\frac{g_i}{N(G)} \right)^{x_i}$$

For the assembly problem, G is not known but the results of the n trials are known. Thus, we can consider the likelihood of the parameters of the distribution (g_i) given the outcome of the trials (x_i), which we call the **global read-count likelihood**:

$$\mathrm{L}[g_1, \ldots, g_{4^k} | x_1, \ldots, x_{4^k}] = \frac{n!}{\prod x_i} \prod \left(\frac{g_i}{N(G)} \right)^{x_i}$$

In our approach, we attempt to assemble the genome with the maximum global read-count likelihood. Equivalently, we minimize the negative log of this likelihood. Within the biflow framework, g_i corresponds to the flow through a vertex (k-mer) in the overlap graph, and we want to find a flow that minimizes $-\log L$. In order to formulate this as a convex min-cost biflow problem, we need $-\log L$ to be a separable convex function. That is, we need to find convex functions c_i such that $-\log L = \sum c_i(g_i)$. Unfortunately, since the multinomial distribution has the constraint that $N(G) = \sum g_i$, this is not possible.

However, as the number of trials goes to infinity, the X_i random variables become independent. Because the number of trials (sampled k-mers) is typically large, we can approximate the multinomial distribution as the product of the individual binomial distributions of each X_i. Since in the binomial approximation the length of the genome $N(G)$ is a constant that is independent of each g_i, we can replace it by N, which is the length of the actual genome from which the reads were sampled. The approximate length of the actual genome can be ascertained through one of a number of biological experiments, or through an Expectation-Maximization type approach. For our experiments, we assume that the genome size is known.

The resulting approximation for L is thus

$$\mathrm{L}[g_1, \dots, g_{4^k} | x_1, \dots, x_{4^k}] \approx \prod \mathrm{P}[X_i = x_i] = \prod \binom{n}{x_i} \left(\frac{g_i}{N}\right)^{x_i} \left(1 - \frac{g_i}{N}\right)^{n-x_i}$$

Now we can write $-\log L = K \cdot \sum c_i(g_i)$, where K is some positive constant independent of all g_i, and

$$c_i(g_i) = -(x_i \log g_i) - (n - x_i) \log(N - g_i)$$

We let c_i be the convex cost functions for the vertices of our min-cost biflow problem, and reduce it to a linear min-cost biflow problem by approximating the convex function, as described in the previous section. We will now describe our approach for solving the resulting linear min-cost biflow.

2.4 Efficient Algorithm for (Linear) Min-Cost Biflow

The min-cost biflow problem was formulated by Edmonds, who showed that it is equivalent to perfect b-matchings [7]. Edmonds' work was later extended by Gabow [8], who gave the fastest to-date algorithm for sparse graphs, which runs in time $O(|V|^2 \log^2(|V|))$ in the worst case. Unfortunately, no efficient implementation of this or similar algorithms exists, and the worst-case running time is prohibitive for a large graph, such as the overlap graph of a genome. In this section, we give a much faster 2-approximation algorithm that allows us to efficiently solve min-cost biflow problems with optimal results in most cases.

One of the easiest ways that directed network flow for a graph $G(V, E)$ can be solved is through a reduction to a linear program (LP). The reduction is based on building the $|V| \times |E|$ edge incidence matrix $I_{|V||E|}$ for the graph. Every column of I corresponds to $e \in E$, and every row corresponds to $v \in V$. The cell $I_{m,n}$ is 1 if the edge n is an in-edge of vertex m, it is -1 if it is an out-edge from m, and 0 if it is not incident on m. The edge incidence matrix of a graph can be viewed as the constraint matrix for an LP where the optimal LP solution corresponds to the minimum flow in the graph.

Incidence matrices based on directed graphs are Totally Unimodular (TU), leading to LPs that always have integral solutions. Because in bidirected graphs an edge may be an in-edge or an out-edge on both of its ends, the resulting incidence matrix may have two 1s (or two -1s) in a column. It is also possible to have a 2 or a -2 if it is the only non-zero entry in its column (this corresponds to a loop). Figure 1b gives an example. The resulting matrices are known as binet matrices [2], and have the property that the optimal solution of the LP is guaranteed to be half-integral (a multiple of 0.5).

Our algorithm is based on the recent result by Hochbaum [10], who demonstrates a reduction from a binet matrix to a TU matrix by monotonization: doubling the number of columns and rows. Solving the LP defined by the new TU matrix is equivalent to solving it in the original binet matrix. However the new TU matrix corresponds to a directed graph, and one can find the min-cost flow in directed graphs using algorithms that are much faster than general LP solvers. We now formulate the monotonization procedure of Hochbaum in terms of the underlying bidirected graph.

For every vertex v of the original bidirected graph we introduce two vertices v_1 and v_2 in the new directed graph. For every in-edge of v we create two directed "twin" edges, one of which points into the v_1 vertex and the other points out of the v_2 vertex. For all out-edges of v, we again create twin edges, one of which points out of v_1 and the other into v_2. An example of the transformation is given in Figure 1c. We transfer all of the bounds and costs on the original edges to the respective twin edges, and after finding the min-cost flow in the directed graph we transfer the results to the original bidirected graph by adding the flows through the pairs of twin edges and dividing by two. Because the procedure above is equivalent to the monotonization procedure of Hochbaum, it has the same provable properties, e.g. that the optimal result is half integral and that the monotonized flow is at worst a 2-approximation to the optimal integral flow. This reduction allows us to convert our bidirected flow problem into a directed flow problem, for which many efficient algorithms have been developed, e.g. the network simplex algorithm. We are also able to take advantage of off-the-shelf packages for solving network flow within our implementation.

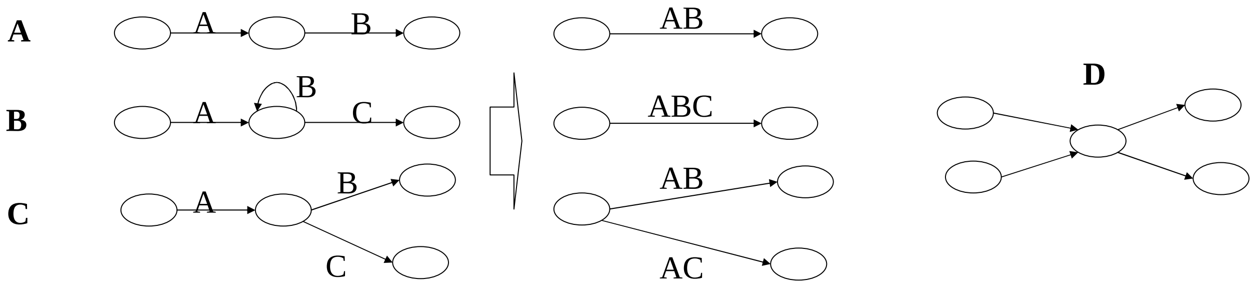


Fig. 2. A, **B**, and **C** demonstrate the three cases of graph simplification described in Section 2.5. Case A is a chain, case B a loop attached to a chain, and case C is a split vertex. A join vertex case is symmetrical and is not shown. The three simplifications are shown to the right. In all cases, the new graph can "spell" the exact same strings as the initial graph. **D.** This is a conflict node. By iterative application of cases A, B and C, we generate a graph where all remaining vertices are of type D.

2.5 From Flow to Contigs

At this point of our algorithm, we have found a flow on the overlap graph, as described above. In general, any flow can be decomposed into a collection of walks, which, in our case, correspond to the assembled contigs. Since there is an exponential number of decompositions possible, we use a heuristic to find one where the length of the walks (contigs) is large and the accuracy of the contigs is high.

Graph simplification: In many cases, it can be inferred that certain walks will appear as a subwalk in any decomposition. First, we remove all edges with flow zero from the overlap graph. Next, by applying the following the three rules to every vertex v, we can greatly simplify the overlap graph (see Figure 2):

Case A. There is exactly one edge going into v and exactly one edge going out of v. The flow on both edges is the same. We can merge the two edges and remove v from the graph.
Case B. There are exactly two edges going out of v and two edges going into v, and exactly one of the edges going out of v is also going into v (a loop). The flow on all three edges is the same. We can merge the three edges and remove v from the graph.
Case C. There is exactly one edge going into v and $m > 1$ edges going out of v (v is a split vertex), or there is exactly one edge going out of v and $m > 1$ edges going into v (v is a join vertex). The flow on the in(out) edge is equal to the sum of the flows on the out(in) edge. We can split the in(out) edge into m copies, merge each one with one of the out(in) edges, and remove v from the graph.

We call a vertex v removable if it falls into one of the above cases, and a **conflict vertex** otherwise. For every removable vertex in the graph, we perform one of the three operations above. It can be shown that after at most $2|V|$ operations, all the remaining vertices are conflict vertices. In practice, this process reduced the number of edges in the overlap graph by over 10^5 fold.

Conflict node resolution algorithm: Once the graph contains only conflict vertices, we attempt to resolve each in turn by finding pairs of edges that are incident on the vertex with opposite orientations and are supported by matepairs. For each vertex we do a breadth first search in both the in and out directions, recording all of the reads that are within a specified distance threshold. We skip any read that was initially on an edge that had been split (during case C of the previous step), as it no longer has a unique position in the overlap graph. We now have two sets of vertices, L and R, corresponding to reads that were observed on the in side of a vertex and out side of a vertex respectively (see Figure 3). The high coverage provided for by NGS methods allows us to concentrate our analysis on reads only a short distance away from the conflict vertex. For each of the reads found, we locate their matepairs in the graph (treating the forward and reverse matepairs separately) and run an all-pairs bounded shortest path algorithm from all the mates of L to all

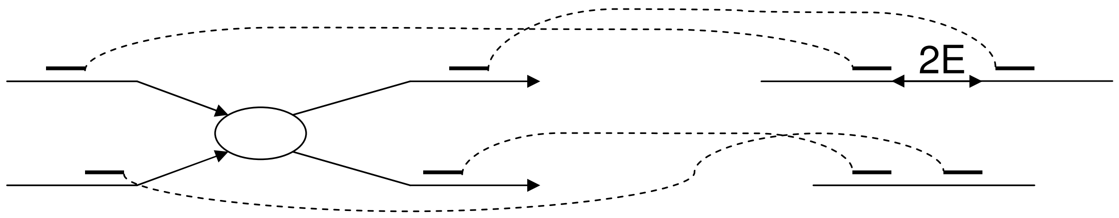


Fig. 3. Resolution of a conflict vertex: we find all reads within a pre-specified distance from the conflict vertex, and locate their mates on the graph. Because the reads are close together, if they spell the same path we expect their mates to also be close together. If the distance between the mates is less than twice the error of the insert size ($2E$) we consider the two mates to support each other. In the example the top left edge will be joined with the top right, and bottom left with the bottom right.

the mates of R. Because the overlap graph is sparse, the most efficient algorithm for all-pairs shortest path is to run Dijkstra's algorithm from every vertex. Furthermore, we terminate Dijkstra's algorithm when all vertices within the bounding distance have been explored: if we expect that the true size of the insert will vary by at most E from the expected size, than the bounding distance is $2E$.

To resolve conflict vertices we implement a simple greedy matching algorithm. All of the edges incident on a particular vertex are separated into two classes depending on their direction at the node – *in* or *out*. For every pair of (in, out) edges, we compute the number of mates that are within the bounding distance from each other. If a significant fraction of one edge's matepairs are within this distance from the matepairs of another edge on the opposite side (a matching condition), the two edges are joined into a common edge. We handle any half-integral edges by allowing either of the edges to get matched to the integral edge incident to the conflict vertex. The process is repeated until no more pairs of edges that satisfy the matching condition are found at the current vertex.

After every conflict vertex has been considered, the graph simplification steps described in the previous section are run again, as new removable vertices may be created during the matching process. The matching procedure is then iterated for a set number of steps, or until convergence.

3 Results

We implemented a prototype assembly algorithm for short reads using the algorithms described above. We have experimented with both CPLEX and CS2 [9] for solving network flow, and found that while the running times are comparable, CS2 uses less memory; consequently we used it for all of the experiments below. To simplify the implementation, we model the convex cost function using a three-piece linear approximation (see [1] for details). The overall running time of our algorithm is approximately 1 hour on a single machine.

3.1 Description of the Dataset

Because we were unable to obtain any real data from the Solexa system, we generated synthetic read data from the *E. coli* genome, which has a total length of 4.6 megabytes. We uniformly sampled the genome to find the location of the first read of a matepair, and then sampled the second read at a distance within 10% of the expected insert size, also uniformly. The reads generated were always of length 25 and error-free (the importance of the assumptions of error-free reads and uniform coverage is elaborated upon in the Discussion). The coverage rate used was varied from 50 to 100X, though we also tried one test with 200x coverage (a single run of the Solexa system generates 1Gb of data, or greater than 200x coverage for *E. coli*). The minimum overlap length (o_{min}) was varied from 17 to 21. The exact datasets used are summarized in Table 1.

Table 1. The first four columns describe the datasets used in the evaluation of the assembly algorithm. The insert size was simulated with a uniform error of up to 10%. The right side shows the deviations of the predicted read copy-counts from their true values. While half-integral flows were observed with some parameter settings (too low coverage, too low o_{min}), the flow was always integral over all runs with the parameters shown.

Dataset	Coverage	Insert Size	o_{min}	-2	-1	0	+1	+2	+3
50x3k	50	3000	17	4	397	3937038	170	18	6
75x3k	75	3000	19	0	9	4324061	28	3	0
75x6k	75	6000	19	0	7	4324665	22	0	0
100x3k	100	3000	21	0	2	4466328	6	0	0
100x6k	100	6000	21	0	2	4466636	23	0	0
200x6k	200	6000	19	0	0	4547426	4	0	0

Table 2. Evaluation of the assembly quality for the various datasets. Length is the N50/N90 score, number is the number of contigs longer than the N50/N90 length. The errors are computed as the total length of errors over the total size of all contigs.

	N50 Contig			N90 Contig			
Data set	length(kb)	number	1 error per	length(kb)	number	1 error per	Total contigs
50x3k	23.4	53	165k	7.1	189	148k	803
75x3k	25.8	48	177k	7.8	174	154k	731
75x6k	25.1	49	93k	7.9	176	96k	732
100x3k	27.9	48	178k	7.9	171	159k	700
100x6k	25.8	49	106k	7.9	174	109k	736
200x6k	25.8	48	154k	8.0	174	158k	727

3.2 Read Count Results

To evaluate the accuracy of our maximum likelihood flow solving algorithm we compared the flow going through every vertex in the overlap graph to the number of times that the corresponding read appears in the original genome. The results are presented in Table 1. For the vast majority of the reads we correctly predicted their copy count in the genome, with the fraction of misestimated counts varying between 10^{-4} and 10^{-6}, depending on the coverage. When our algorithm mispredicted the number of occurrences, the error was typically small compared to the true frequency of the read. We also note that the results show only slight improvement past 75x coverage.

3.3 Overall Assembly Results

In order to estimate the quality of the assembly resulting from matepair information, we take every edge of the graph after the conflict node resolution and generate the sequence which it spells. As per convention, we compute the N50 and N90 scores as the length of the shortest contig such that 50 and 90 percent of the original genome is in longer contigs, and the number of such contigs. To check for the presence of errors in the assembly, each contig was aligned to the reference *E. coli* genome. The number of errors in a contig was computed as the number of local alignments that is required to completely tile it minus one. The results are summarized in Table 2.

Overall the length of the contigs which contained 50 and 90 percent of the genome varied between 23-28k and 7-8k, respectively, while the error rate was about one error every 100-180k in the longer N50 contigs, and one error every 100-160k for the N90 contigs. These errors illustrate a weakness of a greedy matching algorithm, which may be mislead by two well-matched edges that contradict many other good matchings. While the contig sizes are short by the standard of whole genome assembly with Sanger reads, they compare favourably with the results that Chaisson et al. [4] obtained on *Neisseria meningitis* (genome length 2.2Mb) with 70 nucleotide reads, albeit without matepairs: in their experiments they required 344 contigs to achieve 95% coverage of the genome, while our algorithm required 206 contigs to cover 95% of *E. coli*, a genome which is twice as long as *N. meningitis*. These results demonstrate the power of matepair information in resolving the proper layout of the genome, even in the case of very short reads.

4 Discussion

In this paper we explore the potential for *ab initio* whole genome shotgun sequencing with very short, mated reads, similar to those that are produced by novel sequencing technologies such as Solexa or the AB SOLiD system. We demonstrate that 25 nucleotide reads, while a significant challenge to assemble, can in fact be used to construct contigs of a reasonable length by using a combination of a network flow algorithm that is able to accurately capture the frequencies at which the various reads occur in the genome and the further use of matepairs to resolve the paths in the resulting graph. While the current algorithm does not yet allow for the sequencing of an unknown bacterial genome using NGS reads, it does indicate that *ab initio* whole genome shotgun sequencing is indeed possible with read data even as short as 25 nucleotides, in the presence of matepair information. Two potential avenues towards this goal may include the use of matepair information to connect the various contigs into supercontigs, and improvements to the algorithm used to resolve the conflict nodes. At the same time, we believe that the general approach of using convex network flow to estimate frequencies of strings in a genome is a more general technique with other applications in computational biology, such as building repeat libraries for newly-sequenced genomes.

In our experiments we make two major assumptions – that the reads are error-free, and that the genome sequencing rate is uniform. We believe that the first of these assumptions is not fundamental, and a limited amount of error in the reads can be overcome using methods similar to the ones developed for the EULER assembler [19]. Moreover, the high coverage rate should improve the correction accuracy of these methods. The second assumption, however, is more essential to the accuracy of our algorithm. In the case of non-uniform coverage of certain areas in the genome (in particular it is suspected that the Solexa machines may under-sample homo-polymer runs) our algorithm may be less accurate at predicting the copy-counts, which may have significant effects in downstream analyses. We believe that these effects can be neutralized if the biases of the sequencing apparatus are known. For example, each read's observed frequency can be adjusted depending on its sequence. The exploration of the exact biases of the NGS platforms and the correction for these is an important avenue for future research.

5 Acknowledgments

We are grateful to Gene Myers and Pavel Pevzner for useful discussion during the development of these algorithms. This work was funded by an NSERC Discovery Grant, an equipment grant from Canada Foundation for Innovation, and NIH Grant RO1-GM81080.

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
2. Gautam Appa and Balázs Kotnyek. A bidirected generalization of network matrices. *Networks*, 47(4):185–198, 2006.
3. Serafim Batzoglou, David B Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, Evan Mauceli, Bonnie Berger, Jill P Mesirov, and Eric S Lander. Arachne: a whole-genome shotgun assembler. *Genome Res*, 12(1):177–89, Jan 2002.
4. Mark Chaisson, Pavel A. Pevzner, and Haixu Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–2074, 2004.
5. M.J. Chaisson and P.A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res*, pages Published online before print, DOI: 10.1101/gr.7088808, 2007.

6. 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 Res.*, 17:1697–1706, Nov 2007.
7. J. Edmonds. An introduction to matching. Notes of engineering summer conference, University of Michigan, Ann Arbor, 1967.
8. Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC*, pages 448–456, 1983.
9. Andrew V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 22(1):1–29, 1997.
10. Dorit S. Hochbaum. Monotonizing linear programs with up to two nonzeroes per column. *Oper. Res. Lett.*, 32(1):49–58, 2004.
11. Chen J and Skiena S. Assembly For Double-Ended Short-Read Sequencing Technologies. In Mardis E. and Kim S. and Tang H., editor, *"Advances in Genome Sequencing Technology and Algorithms"*, pages 123–141. Artech House Publishers, 2007.
12. W.R. Jeck, J.A. Reinhardt, D.A. Baltrus, M.T. Hickenbotham, V. Magrini, E.R. Mardis, J.L. Dangl, and C.D. Jones. Extending assembly of short DNA sequences to handle error. *Bioinformatics*, 23:2942–2944, Nov 2007.
13. John Dimitri Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, University of Arizona, Tucson, AZ, USA, 1992.
14. Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *WABI*, pages 289–301, 2007.
15. Eugene W. Myers. The fragment assembly string graph. In *ECCB/JBI*, page 85, 2005.
16. Eugene W. Myers, Granger G. Sutton, Art L. Delcher, Ian M. Dew, Dan P. Fasulo, Michael J. Flanigan, Saul A. Kravitz, Clark M. Mobarry, Knut H. J. Reinert, Karin A. Remington, Eric L. Anson, Randall A. Bolanos, Hui-Hsien Chou, Catherine M. Jordan, Aaron L. Halpern, Stefano Lonardi, Ellen M. Beasley, Rhonda C. Brandon, Lin Chen, Patrick J. Dunn, Zhongwu Lai, Yong Liang, Deborah R. Nusskern, Ming Zhan, Qing Zhang, Xiangqun Zheng, Gerald M. Rubin, Mark D. Adams, and J. Craig Venter. A Whole-Genome Assembly of Drosophila. *Science*, 287(5461):2196–2204, 2000.
17. Pavel A. Pevzner and Haixu Tang. Fragment assembly with double-barreled data. In *ISMB (Supplement of Bioinformatics)*, pages 225–233, 2001.
18. Pavel A. Pevzner, Haixu Tang, and Glenn Tesler. *De novo* repeat classification and fragment assembly. In *RECOMB*, pages 213–222, 2004.
19. Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98:9748–9753, 2001.
20. R.L. Warren, G.G. Sutton, S.J. Jones, and R.A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23:500–501, Feb 2007.
21. http://www.ebi.ac.uk/∼zerbino/velvet/.