ABSTRACT

Motivation: Current genomic sequence assemblers assume that the input data is derived from a single, homogeneous source. However, recent whole-genome shotgun sequencing projects have violated this assumption, resulting in input fragments covering the same region of the genome whose sequences differ due to polymorphic variation in the population. While single-nucleotide polymorphisms (SNPs) do not pose a significant problem to state-of-the-art assembly methods, these methods do not handle insertion/deletion (indel) polymorphisms of more than a few bases.

Results: This paper describes an efficient method for detecting sequence discrepancies due to polymorphism that avoids resorting to global use of more costly, less stringent affine sequence alignments. Instead, the algorithm uses graph-based methods to determine the small set of fragments involved in each polymorphism and performs more sophisticated alignments only among fragments in that set. Results from the incorporation of this method into the Celera Assembler are reported for the D. melanogaster, H. sapiens, and M. musculus genomes.

Availability: The method described herein does not constitute a stand-alone software application, but is laid out in sufficient detail to be implemented as a component of any genomic sequence assembler.

Contact: daniel.fasulo@celera.com

Keywords: whole-genome assembly; shotgun sequencing; polymorphism.

INTRODUCTION

Most algorithms for large-scale DNA assembly make the simplifying assumption that the input data is derived from a single homogeneous source. Although these algorithms can be extended easily to take Single Nucleotide Polymorphisms (SNPs) into account, modifying them to deal with larger variations is more difficult. Such variations can occur in regions where there is a high incidence of point mutations, or where there are longer contiguous discrepancies due to polymorphisms such as variable-length microsatellites or block insertions and deletions (indels).

The first step in DNA assembly is generally to calculate the pairwise overlaps of all of the fragments. Discrepancies due to polymorphisms will cause false negatives in the overlap relation since two fragments that are from the same position in the genome will contain different sequences, and thus will not be found to overlap. Particularly in the case of whole-genome assembly, decreasing the stringency of the overlap test to avoid these false negatives will cause an unacceptable explosion in the number of false positives and a significant increase in computation time. Hence, a more sophisticated algorithmic approach to detecting and correcting incomplete assembly due to polymorphism is needed.

This paper describes such an algorithm as an extension of the graph-theoretic methods for fragment assembly described by Myers (1995). As such, it assumes that the overlaps between all fragments have been pre-computed and placed into a fragment overlap graph, which is then used as the fundamental data structure for detecting polymorphisms. Variations of this general framework form the basis for many modern assemblers, such as the TIGR assembler (Sutton et al., 1995), Phrap, and the Celera Assembler (Myers et al., 1995).

The method has two major phases. The first phase identifies potential polymorphisms by analysing the fragment overlap graph. The second phase attempts to generate the overlaps necessary to obtain a consistent layout of all of the fragments in a polymorphic region. The method has been implemented as a subroutine of the Celera Assembler, the software which was used to produce the published consensus genomic sequences of D. melanogaster (Adams et al., 2000) and H. sapiens (Venter et al., 2001).
in \( \Pi \) is a record in which \( \pi.A \) and \( \pi.B \) denote the two fragments involved in the overlap, and \( \pi.A[\pi.s_A, \pi.e_A] \) and \( \pi.B[\pi.s_B, \pi.e_B] \) denote the overlapping substrings of \( \pi.A \) and \( \pi.B \), respectively. Overlaps are divided into two general classes. If \( \pi.B = \pi.B[\pi.s_B, \pi.e_B] \) then \( \pi.B \) is contained within \( \pi.A \), and \( \pi \) is called a containment overlap. Otherwise, the overlapping substrings of \( \pi.A \) and \( \pi.B \) must be either a prefix or suffix of their respective fragments, and the overlap is called a dovetail overlap.

A fragment that is contained within any other fragment is called a contained fragment. Most contained fragments are not useful for polymorphism detection and can be removed from the process. Specifically, suppose \( c \) is a fragment contained in another fragment \( f \). If for all other fragments \( f' \) overlapping \( c \), \( f' \) overlaps \( f \) with coordinates consistent with its overlap with \( c \) then \( c \) is a consistently contained fragment and can be eliminated from consideration during the polymorphism detection process along with its associated overlaps.

**Fragment graphs**

Prior work on graph-theoretic approaches to fragment assembly have constructed an overlap graph in which the vertices represent the fragments and the set \( \Pi \) specifies the edges. However, the standard graph model does not contain enough information to perform fragment layout. Therefore the overlap graph is generally a hybrid model with extra information and constraints encoded in the edges, and unfortunately its complex nature makes it difficult to analyse with traditional graph methods. For efficiency in polymorphism detection, it is useful and possible to abstract away most of this complexity.

Suppose for a moment that the true layout of the input fragments were known and specified as an arrangement of intervals on the number line. Define the abstract fragment graph \( G \) as follows: \( G \) contains a vertex for every interval and a directed edge for every dovetail interval overlap, with the edge directed from the interval with the leftmost endpoint to the interval with the rightmost endpoint. Note that this graph is acyclic, a property that is key in making the polymorphism detection algorithms efficient. For convenience, it is also assumed that the set of edges has been dechorded. That is, if \( (u, v) \) is an edge in the dechorded graph, then there is no vertex \( v' \) such that \( (u, v') \) and \( (v', v) \) are edges in the graph. Since \( G \) is ideally derived from an interval graph, in general dechording removes all of the transitive edges from the graph, although for real data this is not always the case.

Of course, the abstract fragment graph is not truly known. Instead, it can be approximated by a simple analysis of the overlap relation which is similar in nature to a depth-first search. An unplaced fragment \( f \) is selected and its left endpoint arbitrary placed at position 0. Its first overlap \( \pi \) is selected. Without loss of generality, assume that \( f \) is \( \pi.A \); then, \( \pi.B \) can be placed via the positional information in \( \pi \) and an edge between \( \pi.A \) and \( \pi.B \) can be generated and oriented appropriately. As with depth-first search, the fragment \( \pi.B \) is then processed recursively, and so on. Whenever an overlap is encountered between two fragments which have already been positioned, it is either retained as an edge or discarded, depending on whether or not the overlap is consistent with the positions of the fragments. In general, the overlap does not have to agree with the exact coordinates which have been assigned to the fragments; rather, it merely needs to have the correct orientation. Of course this approximation of the abstract fragment graph is not accurate on a global scale, but locally fragments are arranged correctly relative to one another in unique areas of the genome. In practice, this is sufficient for the polymorphism removal procedure to be run.

**Bubbles**

Assume that the input fragments were derived from a heterogeneous source. Then, scattered throughout the assembly will be polymorphic loci. Assume that when two fragments contain different alleles at a locus within their overlapping region, then with some probability the overlap is not detected and its corresponding edge is not present in the graph. In terms of \( G \), these polymorphic regions would have distinct signatures. A couple of examples are given in Figure 1. Figure 1(a) shows the simplest case, in which a single overlap is missing due to variation between two fragments. The difference is denoted as an ‘X’ in one fragment and an ‘O’ in the other. The left side of the figure shows the arrangement of fragments and the site at which two fragments differ. The right side shows the corresponding region of the abstract fragment graph. Part (b) of the figure shows a slightly more complex scenario involving multiple fragments per allele. Notice in particular that \( Y \) is contained by fragment \( X \), but is
not consistently contained because \( X \) does not overlap \( V \). Also, note that there is no edge between \( Y \) and \( X \) in the graph, because the graph contains only dovetail edges.

Such polymorphism-induced formations in the graph are called *bubbles*. Intuitively, a path through the abstract fragment graph represents a layout of fragments. A bubble represents a region in which multiple paths exist, each containing at least one unique vertex, flanked by non-ambiguous regions. Hence, bubbles are regions where there are multiple correct but mutually-exclusive fragment layouts. The next several paragraphs will develop a more precise definition of a bubble.

Consider an arbitrary, directed, acyclic graph \( D \) with vertex set \( V_D \). Suppose that \( D \) has a special vertex \( v_0 \in V_D \) with the property that all other vertices in \( D \) are reachable from \( v_0 \). Traditionally, if \( v \) is a vertex such that all paths starting at \( v_0 \) and ending at \( v \) pass through a vertex \( v' \), then \( v' \) is said to *dominate* \( v \). The analysis of this domination relationship is a key part of many data-flow optimizations in compilers (Aho and Sethi, 1986), and the set of dominating vertices in a directed graph can be computed quickly, particularly if the graph is acyclic. In the case of the abstract fragment graph there may not be a clear choice for \( v_0 \); to overcome this problem, an artificial vertex \( v_0 \) can be logically added to \( V_D \). This vertex is then given an edge to every vertex in \( V_D \) with indegree 0. With this caveat in mind, let \( B \) be a set of vertices in \( D \), and let \( D \) be the subgraph induced by \( B \). Then \( D \) is a bubble if it has the following properties:

1. There exists a vertex \( v_t \in B \) such that every vertex in \( B \) is dominated by \( v_t \) in \( D \). The vertex \( v_t \) is called the initiation vertex. Observe that since the graph is acyclic, the initiation vertex must be unique with respect to \( B \).

2. There exists a vertex \( v_i \in B \) such that every vertex in \( B \) is dominated by \( v_i \) in \( D \), where \( R(D) \) indicates the graph generated by reversing the direction of all of the edges in \( D \). The vertex \( v_i \) is called the termination vertex, and must be unique with respect to \( B \).

3. The vertices \( v_t \) and \( v_i \) must have indegree and outdegree respectively less than or equal to 1 in \( D \).

4. Every vertex in \( D \) on a path from \( v_t \) to \( v_i \) is in \( B \).

5. There exist vertices \( u \) and \( v \) in \( B \) such that neither \( v \) nor \( u \) is reachable from the other.

The first two localization properties involving domination are important because they localize the bubble; it is apparent that all of the fragments involved in the bubble belong between \( v_t \) and \( v_i \) in the assembly. They also guarantee that the sequence path corresponding to each allele can be separately assembled, hence confirming that the bubble is likely to be caused by polymorphism and not sequencing error. The third property, uniqueness of flanking regions, reflects the notion that the bubble must exist between regions of unambiguous sequence; the in- and outdegree requirements enforce that the immediate flanking regions can be uniquely assembled. The fourth property requires completeness of the bubble, so that all fragments belonging on dovetail paths between \( v_t \) and \( v_i \) in the assembly are considered to be part of the bubble. Finally, the fifth property, divergence, specifies that the bubble must contain alternative potential assemblies using different sets of fragments, since there must be at least one pair of paths through the bubble that are disjoint at least one vertex.

As defined above, bubbles have ill-defined boundaries. For example, in Figure 2(a), either \( s \) or \( s' \) could technically be the bubble initiation node, despite the intuition that \( s' \) is more appropriate. To capture this intuition, bubble \( B \) is said to be *minimal* if it cannot be expanded or contracted by keeping one boundary vertex fixed and moving the other. More precisely, the graph \( B \) is minimal if there is no descendant \( v_t' \) of \( v_t \) such that \( v_t' \) and \( v_t \) constitute the initiation and termination nodes of a bubble, and conversely there is no ancestor \( v_i' \) of \( v_i \) such that \( v_i' \) and \( v_i \) constitute the initiation and termination nodes of a bubble. Note that a minimal bubble can still contain a smaller bubble, as demonstrated in Figure 2(b). Henceforth, the term ‘bubble’ will refer to a minimal bubble unless otherwise stated.

**METHODS**

**Bubble identification**

As in the previous section, let \( G = (V_G, E_G) \) be a directed, acyclic graph. For each vertex \( v \in V_G \), let \( \text{pred}_G(v) \) be the set of predecessors of \( v \) in \( G \); that is,

\[
\text{pred}_G(v) = \{ u \in V_G : (u, v) \in E_G \}.
\]
Now, let indegree\((v)\) and outdegree\((v)\) denote the indegree and outdegree respectively of a vertex \(v\). The vertex \(v\) is considered a potential initiation vertex (PIV) if indegree\((v)\) \(\leq 1\) and outdegree\((v)\) \(> 1\).

The first criterion helps to ensure that \(v\) occurs in a unique region of the genome (this criterion can also be augmented by tests on the sequence of the fragment represented by \(v\)). The second criterion specifies that \(v\) may begin a bubble, since at least two paths containing distinct sets of vertices can be initiated at \(v\). At the same time, the number of potential paths starting at \(v\) should be limited by the number of distinct alleles that it is possible to see in the input data. Hence, in practical implementations the additional constraint that outdegree\((v)\) \(\leq \delta\) may be imposed, where \(\delta\) is a user-specified parameter. An outdegree greater than this limit indicates the likely presence of repetitive sequence in the fragment represented by \(v\).

For each vertex \(v \in V_G\), define the set of dominating potential initiation vertices of \(v\) in \(G\), denoted \(\text{dom}_G(v)\), as follows:

\[
\text{dom}_G(v) = \begin{cases} 
\{v\} \cup \bigcap_{u \in \text{pred}_G(v)} \text{dom}_G(u) & \text{if } v \text{ is a PIV,} \\
\bigcap_{u \in \text{pred}_G(v)} \text{dom}_G(u) & \text{otherwise.}
\end{cases}
\]

Conversely, define a potential termination vertex (PTV) as a PIV in \(R(G)\), and note that the set of dominating potential termination vertices for each vertex \(v\) is simply \(\text{dom}_{R(G)}(v)\). Observe that both \(\text{dom}_G\) and \(\text{dom}_{R(G)}\) can be computed easily in time \(O(|V_G| + |E_G|)\) by performing a topological sort of \(G\) and applying the definition above on the vertices in the appropriate topological order, forward or reverse. The following theorems, whose proofs have been omitted for brevity, show how bubbles can be easily identified via dom sets.

**Lemma 1.** If \(s\) and \(s'\) are distinct potential initiation vertices, then \(\text{dom}_G(s) \neq \text{dom}_G(s')\). Similarly, if \(t\) and \(t'\) are distinct potential termination vertices, then \(\text{dom}_{R(G)}(t) \neq \text{dom}_{R(G)}(t')\).

Define an \((s, t)\)-pair to be a pair of vertices \((s, t)\) such that \(s\) is a PIV, \(t\) is a PTV, \(\text{dom}_G(s) = \text{dom}_G(t)\), and \(\text{dom}_{R(G)}(s) = \text{dom}_{R(G)}(t)\).

**Theorem 1.** Every PIV \(s\) and PTV \(t\) may participate in at most one \((s, t)\)-pair.

Now let \(B_{(s,t)} \subseteq V_G\) be the set of vertices dominated by \(s\) in \(G\) and \(t\) in \(R(G)\). That is,

\[
B_{(s,t)} = \{v : s \in \text{dom}_G(v) \land t \in \text{dom}_{R(G)}(v)\}.
\]

Let the subgraph of \(G\) induced by \(B_{(s,t)}\) be denoted by \(B_{(s,t)}\).

**Lemma 2.** Let \((s, t)\) be an \((s, t)\)-pair. Then \(B_{(s,t)}\) is a minimal bubble in \(G\).

**Lemma 3.** If \(B\) is a minimal bubble with initiation vertex \(s\) and termination vertex \(t\), then \(s\) and \(t\) form an \((s, t)\)-pair.

These results yield the key theorem for bubble identification, whose proof follows directly from Theorem 1, Lemma 2, and Lemma 3.

**Theorem 2.** For a DAG \(G\), the set of \((s, t)\)-pairs contained in \(G\) is uniquely determined by the dom relationship. Furthermore, there is a one-to-one correspondence between the \((s, t)\)-pairs and the minimal bubbles in \(G\).

### Implementing bubble identification

This section reviews the algorithmic steps required to identify bubbles, starting with a set of fragments, \(F\), and a pairwise overlap relationship \(\Pi\) computed over \(F\).

1. **Create the abstract fragment graph \(G\).** The first step in performing bubble detection is to create a directed, acyclic graph \(G\) using the method described previously.

2. **If necessary, dechord \(G\).** In the bubble detection implementation in the Celera Assembler, dechording is done directly on the overlap relation \(\Pi\) using methods similar to those described by Myers (1995). In this case, additional constraints must be observed to make certain that all overlaps in the triangle are mutually consistent.

3. **Compute a topological order for the vertices of \(G\).** The advantage of pre-computing the topological order is that both \(\text{dom}_G\) and \(\text{dom}_{R(G)}\) can be computed using the vertex ordering obtained from the sort, thus saving the slight extra overhead of performing multiple graph traversals.

4. **Compute \(\text{dom}_G\) and \(\text{dom}_{R(G)}\) for all vertices in \(G\).** This is done using the definition in the previous section. In the case of \(\text{dom}_G\), vertices are processed in topological order computed in Step 3, and \(G\) is used only to find the predecessors for each vertex. In the case of \(\text{dom}_{R(G)}\), the vertices are processed in reverse topological order. Instead of actually reversing the graph, \(G\) is used and \(\text{pred}(v)\) is replaced with \(\text{succ}(v)\), the set of successors to \(v\), in the definition of \(\text{dom}(v)\).

In addition, a couple of practical schemes can be added to prevent the size of the sets from becoming unwieldy. Let the **age** of a vertex \(u \in \text{dom}_G(v)\) be the length of the shortest path from \(u\) to \(v\). One can then impose a maximum size on \(\text{dom}_G(v)\), with vertices beyond this limit being discarded from the set in order of decreasing age. In addition, one can impose a practical maximum age on vertices.

5. **Find all \((s, t)\)-pairs.** To perform this task efficiently, each PIV \(s\) is placed in a hash table with the key \((\text{dom}_G(s), \text{dom}_{R(G)}(s))\). Lemma 1 shows that this key is unique for each PIV. Then, the pair...
(dom₂(t), dom₃(G(t))) for each PTV t can be looked up in the table to identify which PIV, if any, forms an (s, t)-pair with t.

6. Use traversal to identify all fragments in each bubble. By definition, each vertex v in the bubble has the property that s ∈ dom₂(v) and t ∈ dom₃(G(v)). However, performing this test upon each vertex for each bubble is excessively time consuming, especially given the small, localized nature of bubbles in real data. Instead, a depth-first traversal of the graph is started at s to find all of the fragments in the bubble. The traversal is standard except that it backtracks whenever the vertex t is encountered.

**Processing bubbles**

Once a bubble has been detected and its associated fragments identified, the next step is to smooth the bubble into a single consensus sequence. This can be accomplished in several ways. One method would be to choose a path arbitrarily through the bubble, and eliminate those fragments not on the path. Another possibility is to take every path through the bubble as a potential assembly and calculate a multiple sequence alignment over all potential assemblies. Both of these methods, however, have either theoretical or practical drawbacks. Therefore, the methodology adopted for the Celera Assembler represents a compromise between these two extremes. The Celera Assembler attempts to generate new overlap edges between fragments within the bubble using an affine overlapping procedure. A bubble B is smoothed if, after introduction of these new overlaps and subsequent dechording, it collapses into a single path. This method has the disadvantage that it cannot deal with polymorphisms whose lengths approach the length of a fragment. However, it fits well within the paradigm of graph-based assemblers, and it is reasonably efficient since the number of overlaps computed per bubble is at most quadratic in the size of the bubble and the size of the sequences involved is bounded by the length of a fragment. The steps used to smooth each bubble B are described below.

1. **Compute the adjacency array for B and compute its transitive closure.** In the adjacency array, vertices are ordered first by their topological order, then by their left endpoint position as determined by the abstract fragment graph. Let B′ be the DAG resulting from transitive closure.

2. **For each pair of vertices (u, v) determine if an overlap path exists between u and v.** Since the fragments are arranged positionally within the adjacency array, by iterating outward from the diagonal it is possible to heuristically process those pairs of vertices which are most likely to overlap first. If u is already reachable from v or vice versa, then no further work needs to be done. Otherwise, u is overlapped with v using the fast affine overlap procedure described below. If an overlap is found, then the adjacency array is updated. Let B′′ denote the graph after all possible overlaps have been added.

3. **Find the longest path in the resulting graph.** Once all possible overlaps between the fragments have been found, the bubble is considered smooth if it can be reduced to a path through all vertices. If this is not the case, then the extra overlaps computed above are not introduced into the graph. The reason for this is that the affine overlaps are more generous and hence more likely to be spurious. To make this determination, the longest path through B′′ is found. In almost all cases B′′ is a DAG, for which the longest path can be found in linear time using a slight modification of the topological sort algorithm. If B′′ is not a DAG then there is some inconsistency introduced by the new overlaps, and hence they should be discarded.

One additional risk in the bubble smoothing procedure is that the bubble is not a polymorphism at all, but rather represents divergence in two otherwise identical copies of a repeated sequence. Many of these cases can be identified prior to bubble smoothing by using the Celera Assembler's A-statistic (Myers et al., 1995) to determine if the region containing the bubble seems repetitive. In this case, the hypothetical assembly achieved by combining all of the fragments in the bubble along with those in its uniquely constructable flanking regions is used as input to the statistic. Given this input, the length of the genome, and the total number of fragments, the statistic then gives an estimate of whether the region is unique. Regions that are not unique according to this statistic are deemed unsuitable for bubble smoothing.

**A fast approximate overlap procedure**

Searching for overlaps between two fragments f₀ and f₁ that are in the same bubble, but have no path between them in the initial graph, requires a special ‘gapped’ overlap test in the presence of polymorphisms. The Celera Assembler employs the following procedure, which achieves nearly the sensitivity of the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970) at a fraction of the computational cost.

First, good local alignments are found by extending exact matches that exceed a certain length, in the spirit of the widely-used BLAST (Altschul et al., 1990) programs. If no local alignment is longer than some minimum length and has a mismatch rate below a maximum error tolerance, then a failure is reported. Otherwise, a chain of local alignments is found that gives an optimal approximate overlap. Finally, a global overlap alignment is constructed by tiling together pieces of the selected local alignments. Specifically, if two local alignments that are adjacent in the chain overlap one another on one or both fragments, then their ends are trimmed enough to eliminate the overlap. Additionally, block mismatches are inserted to get from one local alignment to the next. Computation may stop
after any of the three stages if the results to that point are not sufficiently good. Otherwise, \( f_0 \) and \( f_1 \) are considered to overlap according to the resulting alignment.

**RESULTS**

An implementation of the bubble smoothing algorithm has been incorporated into the Celera Assembler and used routinely since summer, 2001. This section reports on the results of running this implementation on three representative genomic assembly problems. The input genomes are described in the list below.

1. *Drosophila melanogaster*. The Celera Assembler was run on a data set approximately equivalent to that used in the production of the published version of the *Drosophila* genome (Adams et al., 2000; Myers et al., 1995).

2. *Mus musculus*. The Celera Assembler was used to produce a consensus assembly of the mouse genome. A total of approximately 5x coverage in fragment reads was generated from three distinct strains and one sub-strain of mouse.

3. *Homo sapiens*. The Celera Assembler has been used to produce a consensus assembly of the human genome. The assembly described in this paper is based upon a data set distinct from that described by Venter et al. (2001). The data presented herein encompasses only the 27.7 million sequencing reads generated by Celera’s sequencing facility, and represents an approximate 5x coverage of the genome. The data were derived from clone libraries generated from a sample of 5 ethnically diverse individuals. The library from a single individual was used to sequence approximately 3.6x coverage in reads, with the other individuals’ libraries contributing approximately 0.3 to 0.5x coverage each.

**Summary**

An overview of the results obtained by running the Celera Assembler on these data sets is presented in Figure 3. Recall that a scaffold is a set of ordered and oriented contigs, potentially with sequence gaps between them. Each gap has an estimated size, thus yielding an estimate of the overall scaffold length. In the figure, the size of the assembled genome is considered to be the portion covered in scaffolds of size greater than 100 kbp, and the total number of base pairs in the assembly is the number of actual consensus sequence base pairs within these scaffolds. As evidence of feasibility, the method’s running time is reported as approximate wall-clock running time on a Compaq machine with 32Gb of RAM and four 667 MHz Alpha processors. The capacity of the machine is excessive; the bubble smoothing implementation requires significantly less than 32 Gb, and is not multithreaded. The remainder of the rows are self-explanatory.

As expected, the heavily-inbred *Drosophila* genome has a much lower frequency of bubble occurrence—approximately 1 bubble for every 47 kbp of sequence—than either the mouse or human assemblies. The human population shows a fair amount of variability and the multiple strains of mice represented in the mouse assembly contribute to a high rate of variability, nearly one bubble every 8.6kbp on average. The bubble smoothing algorithm was particularly beneficial in the case of the mouse genome, not only because it detects polymorphisms, but because it allowed the assembly of these regions. Without bubble smoothing, many of the contigs would have been broken at these polymorphic loci.

In all cases, the number of affine overlaps attempted appears to be linearly proportional to the number of fragments found in bubbles, thus reducing the computational cost of dealing with polymorphisms.

**Distribution of polymorphism sizes**

For the *Drosophila* and human genomes, statistics were collected about the sizes of bubbles. A histogram of the number of fragments in each bubble detected and each bubble smoothed is shown, along with a histogram of the largest block indel between any pair of fragments in each bubble. A value of 0 for the block size indicates a block of smaller than four bases, which is not counted as an identifiable indel by the alignment method. The results for the *Drosophila* genome appear in Figure 5, while those for the human genome are in Figure 6. In both cases, the number of fragments in each bubble is generally very small, although the distribution has a long tail. The mean block indel size of the bubbles in the human genome is nearly twice that found in the *Drosophila* genome, once again suggesting a more heterogeneous population of human donors. In both cases, the block size indicates that overlaps between the fragments could not have been found during the overlapping phase of the assembly without either allowing an unacceptably high error rate, or using a more costly affine overlapping procedure to overlap all fragments.

**Validation by resequencing**

As a final validation of the bubble smoothing method on real data, a set of ten putative polymorphisms from the human genome were submitted for resequencing. The ten examples were hand-selected so that there were at least two fragments showing each variation found in the assembly. Bubbles that were caused by variable-length simple repeats were also discounted. The ten chosen bubbles were then resequenced across the set of five
D. Fasulo et al.

<table>
<thead>
<tr>
<th></th>
<th>D. melanogaster</th>
<th>M. musculus</th>
<th>H. sapiens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembled genome size</td>
<td>123.1Mbp</td>
<td>2.48Gbp</td>
<td>2.79Gbp</td>
</tr>
<tr>
<td>Assembled base pairs</td>
<td>120.8Mbp</td>
<td>2.39Gbp</td>
<td>2.67Gbp</td>
</tr>
<tr>
<td>Number of reads</td>
<td>3.2M</td>
<td>42.1M</td>
<td>27.7M</td>
</tr>
<tr>
<td>Bubbles found</td>
<td>2,784</td>
<td>275,923</td>
<td>70,297</td>
</tr>
<tr>
<td>Bubbles smoothed</td>
<td>2,277</td>
<td>256,746</td>
<td>61,889</td>
</tr>
<tr>
<td>Pct. Bubbles smoothed</td>
<td>82%</td>
<td>93%</td>
<td>88%</td>
</tr>
<tr>
<td>Bubbles rejected by A-stat</td>
<td>415</td>
<td>15,658</td>
<td>8,004</td>
</tr>
<tr>
<td>Reads in bubbles</td>
<td>19,277</td>
<td>2,394,873</td>
<td>481,635</td>
</tr>
<tr>
<td>Reads in smoothed bubbles</td>
<td>12,899</td>
<td>2,211,230</td>
<td>388,944</td>
</tr>
<tr>
<td>Overlaps attempted</td>
<td>19,067</td>
<td>3,591,869</td>
<td>352,008</td>
</tr>
<tr>
<td>Overlaps found</td>
<td>11,087</td>
<td>2,930,485</td>
<td>302,476</td>
</tr>
<tr>
<td>Overlaps retained</td>
<td>9,084</td>
<td>2,837,137</td>
<td>295,382</td>
</tr>
<tr>
<td>Running time (h:m:s)</td>
<td>00:03:35</td>
<td>13:27:39</td>
<td>01:05:09</td>
</tr>
</tbody>
</table>

**Fig. 3.** Bubble smoothing results on three large eukaryotic genomes.

**Experiment 1**

(1) TCCC.........................AGAC  
(2) TCCC.........................AGAC  
(3) TCCC.........................AGAC  
(4) TOCTGGAATGCAAGCCACGGCCAGTCAGGAC  
(5) TOCCNNN...

**Experiment 2**

(1) TAAGCAGTGGTCCACTAAGTGTAGTGTGGCAGACCTTAATG  
(2) TARRNN...  
(3) TA..................................TG  
(4) TA..................................TG  
(5) TAAGCAGTGGTCCACCTTA..................TG

**Fig. 4.** Alignments of resequenced polymorphisms from five donors.

Donors using PCR to select the region of interest. Of the ten samples, two failed to amplify due to nearby large tandem repeats. The other eight all clearly showed the presence of polymorphism. Specifically, for each of the eight successful experiments, at least one subject showed a homozygous sequence trace, not all subjects had identical traces, and at least one subject showed a heterozygous trace.

Two examples of the results of this experiment are shown in Figure 4. In the first example, the fourth donor is homozygous with a 26bp insertion and the fifth is heterozygous. In the second example, the second donor is heterozygous, the third and fourth lack the insertion, the first has a 36bp insertion, and the fifth has half of the insertion. The biological relevance of these polymorphisms has not been studied.

**CONCLUSIONS**

This paper has presented a new method for assembling a consensus genomic sequence from DNA fragments derived from heterogeneous sources. This method is very efficient, with all operations performed on the whole fragment graph being linear-time, and relies entirely on algorithms which are provably optimal.

However, there are still several open questions about this method. Although the algorithms provably solve the abstract bubble finding problem, it is not clear exactly how well the bubble model conforms to polymorphism in real data. In particular, it is difficult to assess how many polymorphisms may have been ignored by the algorithm because they did not produce the expected formation in the graph. Also, as currently implemented, the method for processing bubbles is not general enough. It is sufficient only for handling block insertions, block deletions, and tandem repeat variations that are shorter than the length of a single sequencing read. Although they may be detected in the fragment graph, larger insertions and deletions
Efficiently detecting polymorphisms during the fragment assembly process

Fig. 5. Distributions of bubble characteristics from the *Drosophila* genome.

(for example, of large transposable elements) cannot be smoothed by the current implementation.

Nevertheless, the algorithm in its current form has been shown to be sufficiently efficient to be run on mammalian-sized genomes and to enhance the quality of the assemblies produced, particularly when the data is derived from diverse sources. It is the authors’ hope that these methods will contribute not only to better assemblies, but also to the identification of biologically-relevant polymorphisms during the assembly process.

REFERENCES


