BIOINFORMATICS

Vol. 17 Suppl. 1 2001
Pages S132–S139

Design of a compartmentalized shotgun assembler for the human genome

Daniel H. Huson, Knut Reinert, Saul A. Kravitz, Karin A. Remington, Art L. Delcher, Ian M. Dew, Mike Flanigan, Aaron L. Halpern, Zhongwu Lai, Clark M. Mobarry, Granger G. Sutton and Eugene W. Myers

Informatics Research, Celera Genomics, 45 West Gude Drive, Rockville, 20850, USA

Received on February 5, 2001; revised and accepted on March 30, 2001

ABSTRACT

Two different strategies for determining the human genome are currently being pursued: one is the “clone-by-clone” approach, employed by the publicly funded project, and the other is the “whole genome shotgun assembler” approach, favored by researchers at Celera Genomics. An interim strategy employed at Celera, called *compartmentalized* shotgun assembly, makes use of preliminary data produced by both approaches. In this paper we describe the design, implementation and operation of the “compartmentalized shotgun assembler”.

Contact: Knut.Reinert@celera.com

INTRODUCTION

Although current technology for DNA sequencing is highly automated and can determine large numbers of base-pairs very quickly, only at most approximately 900 *consecutive* base-pairs can be reliably read at a time (Sanger *et al.*, 1977). Thus, a larger stretch of consecutive DNA can only be determined by “assembling” it from such short fragments.

Two different strategies for assembling the human genome are currently being pursued: one is the “clone-by-clone” approach, employed by the publicly funded project (PFP) (U.S. Dep. of Energy *et al.*, 1997), and the other is the “whole genome shotgun assembler” (WGA) approach, favored by researchers at Celera Genomics (Webber & Myers, 1997). Both efforts are well under way, and first assemblies of the genome have been published (International Human Genome Sequencing Consortium, 2001; Venter *et al.*, 2001). Below we give a brief description of both approaches.

To best leverage the early stages of both efforts, we designed and implemented a “compartmentalized shotgun assembler” that makes use of preliminary data from both assembly projects and produces a draft of the genome that is more complete than was obtainable from either source separately. Together with Celera’s WGA method, our compartmentalized shotgun assembler was instrumental in computing Celera’s first assembly of the human genome, which was announced in June 2000 (Marshall, 2000), and will be helpful in verifying Celera’s ultimate WGA assembly. The aim of this paper is to discuss the design, implementation and operation of our compartmentalized shotgun assembler from a software engineering point-of-view, and to give a brief summary of the results obtained. In (Huson *et al.*, 2001), we discuss some of the algorithmic issues in detail.

Clone-By-Clone Assembly

In the PFP’s clone-by-clone approach, one first constructs a tiling of the genome by overlapping pieces, each typically of length up to 150k base pairs (bp), and then concentrates on determining the sequence of each such piece. We will call these pieces *BAC clones* or simply *BACs*, since they are usually cloned using “Bacterial Artificial Chromosome” vectors.

The sequence of a BAC is determined using *shotgun sequencing* (Sanger *et al.*, 1992): the BAC is randomly broken into many small fragments which are then individually cloned and sequenced. For a successful assembly, this has to be done with a sufficient amount of oversampling. Statistical calculations (Lander & Waterman, 1988) and systematic experimental studies (Myers *et al.*, 2000) suggest that the average number of fragments covering any given site in the BAC should be at least 7. The sequenced fragments are then run through an assembly program that attempts to construct the full sequence of the BAC from them, by determining how these fragments of sequence overlap with each other. For the purposes of this paper, a BAC is a collection of *bactigs*, i.e. pieces of DNA sequence that are obtained from a common “source region” of approximately 150k bp of contiguous DNA in the human genome, using a shotgun sequencing and assembly process.

The BACs sequenced by the PFP are submitted to

Table 1. Listed by phase, we report the number of BACs, and the average number and length of bactigs in GenBank on September 1, 2000.

Phase	# BACs	avg. # btgs	avg. length (bp)
0	3067	91.5	784
1/2	20960	19.8	8102
3	9494	1.0	94309

Table 2. Fragments are organized in mate-pair libraries, each with a mean distance and standard deviation associated with them. This table was compiled from all of Celera's libraries and reflects the numbers of mate-pairs for the 27.27 million fragments available at Celera.

Library types	# fragments	% mate-pairs	mean distance	standard deviation
2k	13.54M	74.5	1951	119
10k	10.89M	80.8	10800	875
50k	2.83M	75.6	50715	7556

GenBank (Benson *et al.*, 2000) on a regular basis. Such a Genbank entry usually evolves over time, as more work is done to determine the BAC's full sequence. Originally, a BAC may start out as a *phase-0* entry, which means that it consists of $60 - 100$ bactigs of length ≈ 800. A *phase-1* or *phase-2* BAC will usually consist of $10 - 30$ bactigs of length ≈ 8000, whereas a *phase-3* BAC consists of precisely one bactig that represents the full source sequence. Note that bactigs associated with a phase-0 or phase-1 BAC are considered unordered and unoriented, whereas a phase-2 BAC comes with additional information on how the bactigs are ordered and oriented with respect to each other. As of September 1, 2000, GenBank contained 33421 relevant human BACs, see Table 1.

Whole Genome Shotgun Assembly

In the WGA strategy, the *whole* genome is randomly broken into fragments that are individually sequenced, with ideally at least 7-fold sequence coverage. Substantial computational resources and appropriate algorithms are then used to assemble the genome (Myers *et al.*, 2000). Due to the abundance of repeats in genomic DNA, a purely overlap-based approach to WGA assembly is not tractable. To address this, Celera produces fragments in *mate-pairs*. This is done by sequencing larger pieces of DNA from both ends, thus producing pairs of sequenced fragments with known relative orientation and approximate distance (Edwards & Caskey, 1991) (employing a mixture of 2k, 10k, 50k, and 100k bp clones).

Celera's fragment data consists of about 27.27 million fragments of human DNA, each between $150 - 800$ bp long. The majority of them come in mate-pairs of known relative orientation and approximate distance. Paired mates are organized by the clone *libraries* they were selected from, each with an associated, approximately Gaussian insert length distribution (see Table 2 for mean and standard deviation of the libraries).

Compartmentalized Shotgun Assembly

The compartmentalized shotgun assembler (CSA) takes as input the BACs and Celera's fragments and tries to achieve the following objectives: 1) to significantly increase the level of assembly of BACs using the information given by the Celera fragments and mate links, 2) to assemble regions of the genome not covered by BAC sequence into "scaffolds" (a *scaffold* is a collection of pieces of sequence of known relative orientation and approximate distance), (3) to produce an accurate tiling of the ordered BACs and Celera scaffolds, and (4) to assemble the connected components, or "compartments" of this tiling using Celera's WGA assembler.

We ran our incremental pipe-line on approximately 27.27 million Celera fragments, 33241 BACs obtained from Genbank on September 1, 2000, and 11640 scaffolds from regions of the genome that were not covered by public BACs. Using this approach, Celera was able to assemble the human genome into 3845 components (see (Venter *et al.*, 2001) for detailed results).

In the remainder of this paper, we describe the general design, the implementation and operation, and the performance of the CSA.

THREE STAGE DESIGN

The compartmentalized shotgun assembler consists of three stages, *fragment recruitment*, *tiling* and *component assembly*; see Figure 1.

First Stage: Fragment Recruitment

Given a snapshot of the BACs in GenBank, our first goal is to determine which BACs align with which of the 27.27 million Celera fragments. More precisely, for each BAC we compare every bactig with every Celera fragment and we say that a fragment is *recruited by* (or *hits*) a bactig if it (or its reverse complement) globally aligns to the bactig with high identity (94% in our implementation).

The human genome has a high abundance of repeats. We screen fragments for known repeats, mark them accordingly, and take this information into account when computing fragment-bactig hits (Myers *et al.*, 2000). However, not all repeats are known ahead of time and we take additional steps to address this problem at run time (Huson *et al.*, 2001).

Overall, 23.74 million fragments of the 27.27 million

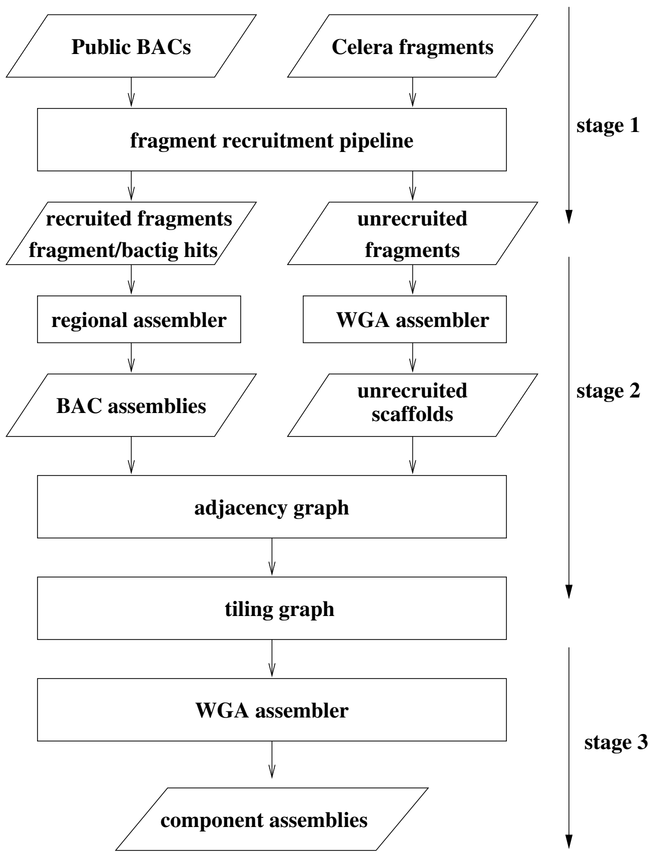

Fig. 1. Three stage design of the compartmentalized shotgun assembler.

remained unscreened (had more than 40 consecutive base pairs not matching a known repeat). Based on all BACs obtained from GenBank, 2.96 million of these fragments (≈ 12.46%) remained unrecruited, and most likely represent sequence not present in the public data set.

Second Stage: Tiling

The goal of this stage is to compartmentalize the data into putatively overlapping or adjacent subsets. First, we assemble the BACs and recruited fragments using the "greedy path-merging" algorithm described in (Huson *et al.*, 2001) and the unrecruited fragments using the WGA algorithm (Myers *et al.*, 2000). Then we determine a tiling of the resulting regional assemblies and "unrecruited scaffolds" which subdivides the data for the final component assembly stage.

Recruited Fragments and BACs Assuming that the size of the euchromatic human genome is about 2.9 billion bp, and given 27.27 million Celera fragments of average length 543 bp, each site in the genome will be covered by about 5.1 fragments, on average.

We can use the fact that the fragments come in pairs of known relative orientation and approximate distance to verify and even correct the given bactig assemblies. Moreover, we can usually determine the relative orientation and ordering of the bactigs of a given BAC, especially if the BAC is in phase-1/2.

Generally speaking, the goal is to assign coordinates to the bactigs that reflect, as closely as possible, the true relative positions and orientations of the bactigs in the source sequence, by making use of the additional information. In (Huson *et al.*, 2001), we formulate this goal as the *bactig ordering problem* in terms of a bactig graph, show its NP-completeness, and describe a heuristic algorithm to solve the problem. In the following paragraph we give a brief summary.

Given a BAC and the set of fragments that hit any of the BAC's bactigs, we first compute the *bactig graph*, which is a weighted, undirected multi-graph, without self-loops. It has two kinds of edges, namely *bactig edges* that represent the bactigs of the given BAC, and *mate edges* that represent mate-pairs between the fragments that hit different bactigs. The *length* of a bactig edge is simply the length of the bactig, whereas a mate edge has both a *length*, reflecting the distance estimation derived from the mate-pairs that are represented by the edge, and a *weight*, representing the actual number of mate-pairs that support the edge.

Initially, each bactig edge in the bactig graph is a *selected path*. We greedily take the mate edge of highest weight that has not yet been considered. We then merge the two adjacent selected paths into a single selected path of non-overlapping bactigs, if it appears to be reasonable. The output of the algorithm is a *regional assembly* in terms of a set of selected paths, each representing a scaffolding of the involved bactigs.

Finally, using those mates of fragments that hit bactigs in the given BAC, but themselves do not hit any such bactigs, we attempt to join scaffolded bactigs by filling the gaps between them.

Unrecruited Fragments An unrecruited fragment is a Celera fragment that has at least 40 bp of sequence that is not labeled repetitive and that does not possess a high quality alignment with any PFP bactig. We processed unrecruited fragments together with the screened fragments (total of 5.89 million) using Celera's WGA assembler (Myers *et al.*, 2000). The output is a list of *contigs* (stretches of contiguous sequence), distributed in 11640 *scaffolds* of size greater than 5k bp (smaller scaffolds were ignored for the graph construction). In this paper, we refer to these assemblies as *unrecruited scaffolds*.

Graph Construction So far, we have described how one can combine the clone-by-clone data and Celera fragments to obtain good assemblies of both BACs and genomic regions not covered by BACs. The next goal is to determine the relative positions of these assemblies within the genome.

Our data suggests that the presence of both a low-hitting fragment f *and* its (low-hitting) mate g *shared* by two different BACs (e.g. fragments 3 and 4 in Figure 2) is good evidence of the two BACs overlapping. This is also true if a fragment is shared and its mate hits either of the BACs (fragments 1 and 2). Similarly, we pay attention to *bridges*, which are mate-pairs of fragments that each hit one of the two BACs (fragments 11 and 12). On the other hand, if a fragment hits both BACs but the mate is contained in some unrelated BAC, then this is negative evidence, as the shared appearance is probably repeat induced (fragment 8 in the left three BACs, and fragment 3 in the right BAC). These considerations can be captured by a simple "adjacency" measure, which is based on a weighted sum of the number of shared fragments and bridges. We define the *adjacency graph* G as the graph whose nodes correspond to BACs and unrecruited scaffolds, and for which any two such nodes A, B are joined by an edge if there is sufficient evidence that they are adjacent or overlapping.

First we compute the *adjacency graph* whose nodes correspond to the BACs and unrecruited scaffolds and whose edges represent evidence that the connected BACs or unrecruited scaffolds are situated close to each other in the genome. Then human curators produce a *tiling graph* from this using internal and external verification methods to determine which edges reflect true adjacencies and which are repeat induced.

Adjacency Graph If a region of the genome is covered by more than one BAC, then a good proportion of Celera fragments that come from that region will be recruited by both BACs. Similarly, if two BACs come from neighboring regions of the genome, then there is a good chance that mated fragments will be recruited by them.

Unfortunately, this signature is indistinguishable from the signature produced by two non-overlapping clones sharing a repetitive region. To reduce this problem, we only consider *low-hitting* fragments, i.e. fragments that only hit a small number of bactigs. We use 5 as the maximum number of hits, under the reasonable assumption that BACs residing in GenBank cover the genome no more than 5 deep at any place.

Construction of the Tiling Graph Whereas many of the edges in the adjacency graph reflect true proximity, others are false edges introduced by low copy repeats, BACs with chimeric content, contaminated clones or other such problems.

Thus, the graph requires manual curation to identify and delete false edges, using both *internal* and *external* data verification. Internal verification involves examining the alignment between clones, inspecting the exact placement of shared fragments, checking the "happiness" of mate-pairs, detecting mis-assemblies and polymorphism, etc. External verification uses high density STS maps, fingerprint maps of BACs, and other external sources of genomic data to confirm or reject edges.

The output of this curation stage is a *tiling graph* whose edges should all reflect true adjacencies in the genome.

Third Stage: Component Assembly

For each connected component of the tiling graph, we then apply the WGA assembler (Myers *et al.*, 2000) to obtain an assembly of the whole region of the human genome that is covered by the component.

We use mate-pair information to evaluate the quality of a given component assembly by comparing the number of happy and unhappy mate-pairs (a mate-pair of fragments that both occur in the component is called *happy*, if its orientation is correct and if the distance between the two fragments is approximately correct; see (Huson *et al.*, 2001) for an exact definition of happiness). Moreover, the achievable quality of the assemblies provides feedback on the quality of the hand-curated tiling graph.

IMPLEMENTATION

We now discuss our implementation of the compartmentalized shotgun assembler. Due to the importance of the human genome, speed mattered, and the whole pipeline was designed and implemented in five months. Where possible, we made use of existing C code from Celera's WGA assembler. New code was written in C++, using the LEDA library (Mehlhorn & Näher, 1999).

As the full extent of the project was not known ahead of time, we employed a modular design using small programs and simple file-based interfaces. We formulated precise near-term milestones. Upon reaching such a milestone, we evaluated the current data and results and then formulated the next desirable and obtainable goals. It was clear that an incremental design was necessary, as the data to be processed would come in batches and indeed much of it would be redefined over time, as new BACs appeared, existing BACs were upgraded, and Celera's mate-pair sequencing proceeded.

Each of the three stages *fragment recruitment*, *tiling*, and *component assembly* are designed as separate processes that are run independently of one another. In this section we discuss each in turn.

Implementation of the Fragment Recruitment Stage

Fragment recruitment is an incremental process. As new data becomes available, it is submitted either as a batch

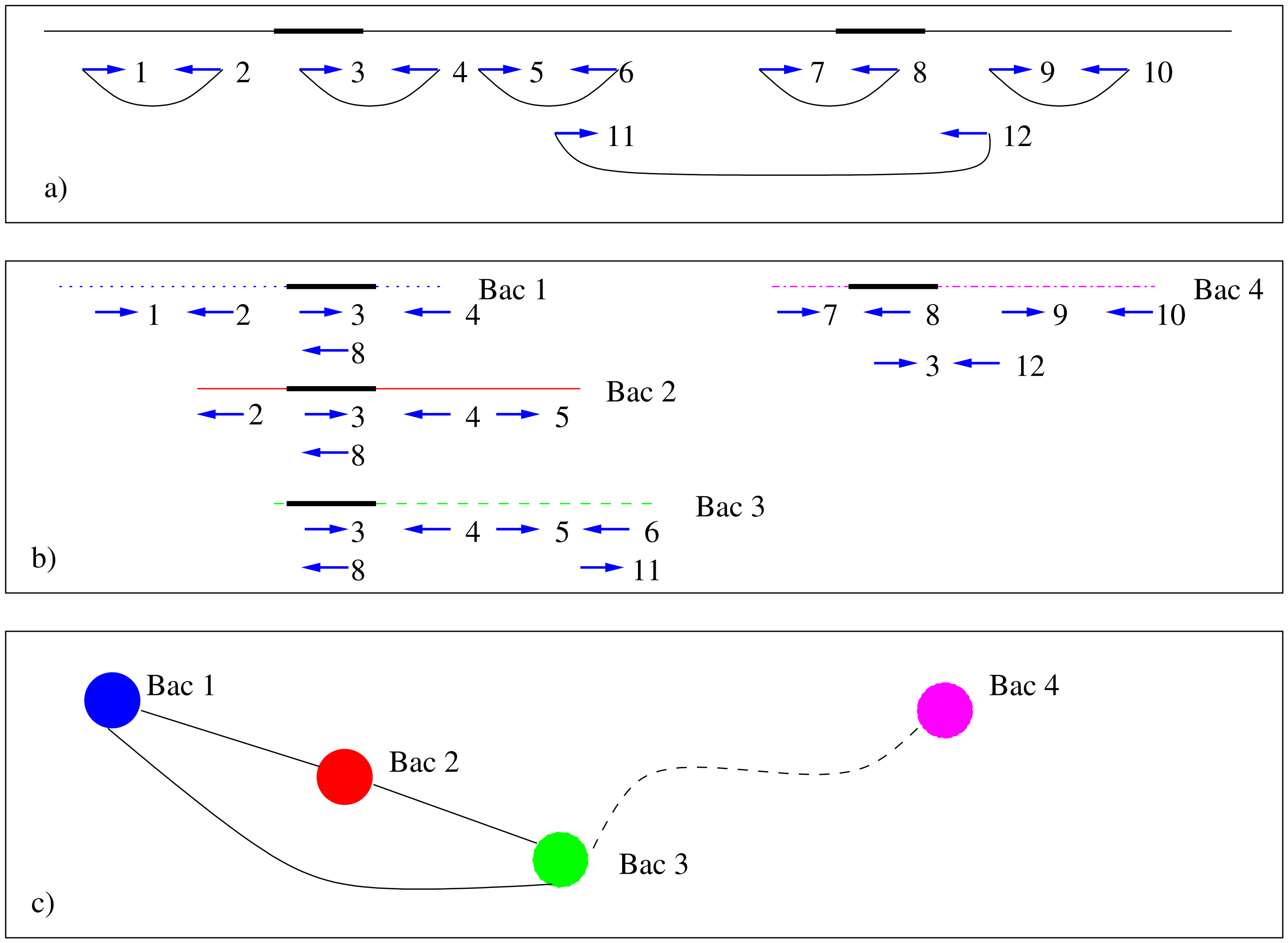

Fig. 2. Example of an adjacency graph. a) is the original sequence with mate pairs sampled from it. The thick lines indicate a repeated region. b) Four BACs that cover a part of the original sequence and c) the corresponding adjacency graph.

of new Celera fragments or as a batch of new or updated public BACs. The process is illustrated in Figure 3.

Any new batch of data first must pass the *gatekeeper*, a program that does bookkeeping and performs a number of checks to insure that incoming data looks reasonable. In the case of a submission of fragments, these are then processed by the *screener*, which attempts to identify and tag possible repeats, using an algorithm similar to BLAST (Altschul *et al.*, 1990). As this computation is quite time intensive, it is distributed on a compute farm using standard load sharing software.

The *populator* then puts the data into two different file-based stores: bactigs into the *bactig store* and fragments into the *fragment store*, overwriting the older versions of bactigs in the case of a BAC redefinition.

The most important and computationally intensive part of this stage is the *overlapper* (Myers *et al.*, 2000). This program compares each Celera fragment with every public bactig to determine which bactigs it hits, i.e. aligns to with high identity. Similar to the screener, the overlapper uses the seed-and-extend idea of BLAST. It looks for seeds consisting of 20-mer exact matches and then attempts to extend them using banded dynamic programming. To avoid repeat-induced hits, repeat regions labeled by the screener do not give rise to seeds.

Our implementation is tuned to finding high-stringency matches and can compare millions of fragment/bactig pairs per second, thus performing orders of magnitude faster than BLAST. Even so, the total required CPU time demands the use of parallel processing. The overlapper program itself is multi-threaded and has an optimal speedup running with 4 threads. In addition, we distribute the overlap computation itself into a number of jobs and run these in parallel on our compute farm. A

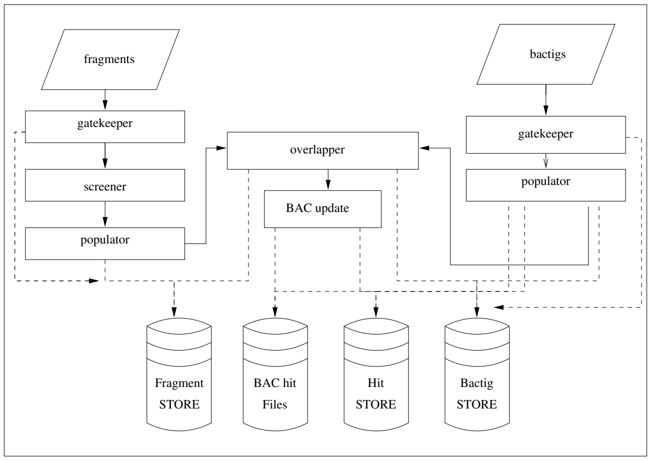

Fig. 3. Fragment recruitment pipeline.

typical job contains approximately 500, 000 fragments and approximately 2000 bactigs, which makes optimal use of machines with 4 GB of main memory by loading up the memory with as many bactigs as possible, and then streaming all fragments against them.

The *BAC update* program uses the resulting fragment/bactig hits (or possible delete intructions) to update *BAC hit files* and the *hit store*. The BAC hit files contain a history of all fragment/bactig hits per BAC, whereas the hit store maintains a global view of fragment/bactig hits, i.e. for each fragment a list of bactigs it hits and vice versa.

Additional *perl*-scripts and programs are used to control the logic of this pipeline. For example, when a batch of new fragments is submitted, after screening them, we must compare this increment of fragments with all bactigs already submitted to the pipeline. On the other hand, when a batch of new BACs is submitted, this increment of bactigs must be compared with all present fragments, whereas for redefinitions of BACs, we must additionally delete outdated bactigs.

The result of this stage of the assembler is one BAC hit file for every BAC, listing all hits of fragments to its bactigs. All fragments that do not hit any bactig are listed in a file of unrecruited fragments. These files are the interface to the later stages of the assembler.

Implementation of the Tiling Stage

Local Assemblies Each BAC is processed by three programs. The first program, called the *data collector*, parses the BAC file, fetches the corresponding bactig and fragment data from the corresponding stores and produces a complete input file for the *regional assembler*. It also detects unscreened repeats and removes probably repetitive fragments, the placement of which is not confirmed by their mate. The regional assembler applies the greedy path-merging assembly algorithm to produce an improved assembly of the given BAC in terms of scaffolds and contigs, with each contig assembled from a number of fragments and bactigs. These fragments and bactigs are merged into a multi-alignment and then a consensus sequence is produced using the *consensus* program. Since we are aligning extremely similar sequences, a simple shift-and-evaluate “abacus” technique suffices (Myers *et al.*, 2000) (a linear number of pairwise alignments is merged into a multialignment and then postprocessed to compress columns). The result is one *BAC assembly file* per BAC.

All unrecruited fragments are fed to Celera’s WGA assembler. From the resulting assembly we extract all scaffolds bigger than a certain threshold and generate one *unrecruited scaffold file* per scaffold.

Construction of the Tiling Graph We have developed a program, *AnnoGraph*, that takes as input a complete set of files for BAC assemblies and unrecruited scaffolds, builds the adjacency graph, and provides interactive viewing and editing capabilities on a component-by-component basis. The program has a general mechanism for launching further programs on selected sets of nodes or edges such as dot-plots or mate-pair tests for chimerism, and thus supports interactive exploration and evaluation of the graph. Additionally, AnnoGraph displays external BAC annotations such as STS markers, probable chromosome assignment, sequencing center, etc.

Celera’s Map Team undertook the task of curating this graph and producing a tiling graph. In a first step, where possible, BACs and unrecruited scaffolds were assigned to specific chromosomes, based on both internal and external evidence, and then the graph was recomputed on a chromosome-by-chromosome basis. Problematic edges in the graph were investigated to decide whether they indicated an overlap or rather reflect a similarity due to an unscreened repeat, chimerism, or other problems. Most frequently, this was decided by visual inspection of the ordering and sequence alignment of the bactigs of the two involved BACs. Obviously, this task is very time consuming and requires a high level of expertise in the field of human genomics. The end result is a set of components which compartmentalize all BACs and unrecruited scaffolds into genomic regions.

Implementation of the Assembly Stage

The only remaining task was to assemble the sequence of the curated components using Celera’s *WGA assembler*.

The assembler takes as input a component of the curated tiling graph (i.e., several putatively overlapping or adjacent BACs or unrecruited scaffolds) and all associated BAC assemblies and unrecruited scaffolds, and computes

Table 3. Phase-0 BAC statistics for the number of input bactigs and output contigs and scaffolds, and their mean sizes.

	Number of pieces					mean size (bp)
	mean	std. dev.	median	maximum	sum	
Input bactigs	91.5	44.1	81	411	271633	784
Output contigs	58.6 (85.7)	26.8	51	221	173833	870
Output scaffolds	55.3 (77.87)	24.7	49	188	164128	922

Table 4. Phase-1/2 BAC statistics for the number of input bactigs and output contigs and scaffolds, and their mean sizes.

	Number of pieces					mean size (bp)
	mean	std. dev.	median	maximum	sum	
Input bactigs	19.8	14.7	17	203	415687	8102
Output contigs	8.9 (11.1)	8.5	7	144	186138	17380
Output scaffolds	2.1 (3.9)	3.6	1	108	44134	73303

an *ab initio* assembly of the source sequence corresponding to the whole component.

In preparation for employing the WGA assembler, all bactigs from BACs present in the given component were "shredded" into fragments of length 550 with a coverage of 2. These fragments, together with all Celera fragments that were recruited by any of the BACs contained in the component, formed the input set for our WGA component assembler.

Since individual BAC assemblies were only used to facilitate the construction and curation of the adjacency graph, the resulting WGA assembly is free to correct mistakes made in the upstream processes. If an error occurred and two regions were joined incorrectly, the WGA can separate these regions into different scaffolds. Finally all resulting scaffolds are passed on to be mapped to the correct genomic region.

PERFORMANCE AND RESULTS

In this section we describe the performance of the compartmentalized shotgun assembler running on the data summarized in Table 1 and Table 2.

Fragment Recruitment Stage

Given the 33421 BACs and 27.27 million fragments as input, a start-to-finish run of the fragment recruitment stage took approximately 14 days, running on a cluster of 20 Compaq ES 40 (4 GB main memory, four 677 MHZ CPUs) servers and one Compaq GS 160 (64 GB main memory, sixteen 667 MHZ CPUs).

The size of the two main stores containing all sequence data grew to 6 GB for the bactig store and 33 GB for the fragment store, whereas the BAC hit files occupied 1.44 GB.

Tiling Stage

Running on the specified farm of machines, computation of all BAC assemblies took about 3 days. In addition, computation of the unrecruited scaffolds took about 1 day.

Generally we consider a regional assembly successful if we can order almost all bactigs into a small number of large scaffolds. "Almost all" means that we disregard small bactigs that are not incorporated into the large scaffolds due to, e.g contamination, bad data quality, or repeats.

More specifically, our measure of quality is the number of scaffolds and contigs that span more than 90% of the sum of all contigs. This is a more accurate measure of the performance of our method than the overall averages, since even in very successful regional assemblies we often found a few very small bactigs that could not be incorporated into the single main scaffold spanning almost the whole BAC. Counting these small bactigs as scaffolds would unduly penalize an otherwise successful assembly. Tables 3 and 4 illustrate the performance of our method on phase-0 and phase-1/2 BACs, where the numbers refer to the above mentioned 90% threshold. The first column contains in addition the total mean. As expected, the algorithm does not perform well on phase-0 data. Too few fragments get recruited and thus we can hardly extend the small bactigs nor order them.

In contrast phase-1/2 BACs are much easier to assemble. Table 4 shows that we more than double the size of the ungapped pieces from 8k bp to 17k bp. Or, put it in another way, we could close on average more than half of the gaps. Further, the amount of ordering is substantial. On average we can order the pieces of the BAC into 2 scaffolds of average length 73k bp.

Table 5. Computer resource allocation for each of the three stages.

Stage	max. memory gigabytes	CPU time hours	wall clock time hours
Fragment Recruitment	4	9240	336
Tiling	2	3970	1280
Assemblies	8	2470	62

Given the performance of the regional assembler on BACs in different phases we observe that a BAC must be sequenced and assembled with at least 3x coverage in order to obtain a successful assembly using mate-pairs.

In order to compute the tiling graph we first have to preprocess the results of the fragment recruitment stage which takes about 4 hours on a single CPU. After this step, generating an initial adjacency graph for any given collection of BACs and unrecruited scaffold takes only minutes. However, the task of producing a tiling graph from the adjacency graph is a formidable one and required the full attention of Celera's Map team. The resulting tiling graph consists of 3845 components ranging from the size of a single BAC to many hundreds of BACs and unrecruited scaffolds.

Component Assembly Stage

Finally we ran Celera's WGA assembler on each of the components. Most of the run-time was spent in constructing the input sets, whereas the actual computation took less than 2 days wall clock time and 2470 CPU hours. The assemblies resulted in 53591 scaffolds, 2845 of which span more than 95% of the published genome (Venter *et al.*, 2001).

In Table 5 we summarize the resource requirements for the three stages. The column CPU time refers to the CPU time consumed whereas the column "wall clock time" reports the time needed for backups, manual curation of the adjacency graph etc. (The maximal memory requirement is per machine used.) Although the maximal memory requirement was low, the use of our 64 GB machine sped things up considerably, presumably due to its better caching behavior for disk based data.

ACKNOWLEDGEMENTS

We are greatly indebted to the members of Celera's Map team, who hand curated the components of the adjacency graph to obtain a biologically meaningful tiling graph, in particular Lin Chen, Pat Dunn, Carlos Evangelista, Zhiping Gu, Ping Guan, Tom Heiman, Ruiru Ji, Zhaoxi Ke, Peter Li, Yiding Lei, Alex Levitsky, Yong Liang, Xiaoying Jonathan Lin, Fu Lu, Natalia Milshina, Helen Moore, Deborah Nusskern, Jack Wang, Jian Wang, Alison Yao, Ming Zhan, Qing Zhang, Weiqing Zhang, Xiangqun Zheng, Fei Zhong, and Wenyan Zhong. We would like to thank our colleagues Mike Flanigan, Randall Bolaños, and Laurent Mouchard.

REFERENCES

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, **215**, 403–410.

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., Rapp, B. A. & Wheeler, D. L. (2000). Genbank. *Nucleic Acids Research*, **28**, 15–8.

Edwards, A. & Caskey, C. (1991). Closure strategies for random DNA sequencing. *Methods: a companion to Methods in Enzymology*, **3**, 41–47.

Huson, D. H., Reinert, K. & Myers, E. W. (2001). The greedy path-merging algorithm for sequence assembly. In *Proceedings of the Fifth Annual International Conference on Computational Biology (RECOMB01)*. pp. 157–163.

International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature*, **409**, 860–921.

Lander, E. S. & Waterman, M. S. (1988). Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, **2**, 231–239.

Marshall, E. (2000). Human genome. Rival genome sequences celebrate a milestone together. *Science*, **288**, 2294–5.

Mehlhorn, K. & Näher, S. (1999). The LEDA Platform of Combinatorial and Geometric Computing. Cambridge University Press.

Myers, E. W., Sutton, G. G., Delcher, A. L., Dew, I. M., Fasulo, D. P., Flanigan, M. J., Kravitz, S. A., Mobarry, C. M., Reinert, K. H. J., Remington, K. A., Anson, E. L., Bolanos, R. A., Chou, H.-H., Jordan, C. M., Halpern, A. L., Lonardi, S., Beasley, E. M., Brandon, R. C., Chen, L., Dunn, P. J., Lai, Z., Liang, Y., Nusskern, D. R., Zhan, M., Zhang, Q., Zheng, X., Rubin, G. M., Adams, M. D. & Venter, J. C. (2000). A whole-genome assembly of Drosophila. *Science*, **287**, 2196–2204.

Sanger, F., Coulson, A. R., Hong, G. F., Hill, D. F. & Petersen, G. B. (1992). Nucleotide sequence of bacteriophage λ DNA. *J. Mol. Bio.*, **162**, 729–73.

Sanger, F., Nicklen, S. & Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, **74**, 5463–5467.

U.S. Dep. of Energy, Office of Energy Research & Office of Biological and Environmental Research (1997). Human genome program report. `http://www.ornl.gov/hgmis/publicat/97pr/`.

Venter, J. C., Adams, M. D., Myers, E. W. *et al.* (2001). The Sequence of the Human Genome. *Science*, **291**, 1145–1434.

Webber, J. L. & Myers, E. W. (1997). Human whole-genome shotgun sequencing. *Genome Research*, **7**, 401–409.