Efficient de novo assembly of large genomes using compressed data structures

Jared T. Simpson and Richard Durbin

Genome Res. 2012 22: 549-556 originally published online December 7, 2011
Access the most recent version at doi:10.1101/gr.126953.111

Resource

Efficient de novo assembly of large genomes using compressed data structures

Jared T. Simpson and Richard Durbin[1]

Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SA, United Kingdom

De novo genome sequence assembly is important both to generate new sequence assemblies for previously uncharacterized genomes and to identify the genome sequence of individuals in a reference-unbiased way. We present memory efficient data structures and algorithms for assembly using the FM-index derived from the compressed Burrows-Wheeler transform, and a new assembler based on these called SGA (String Graph Assembler). We describe algorithms to error-correct, assemble, and scaffold large sets of sequence data. SGA uses the overlap-based string graph model of assembly, unlike most de novo assemblers that rely on de Bruijn graphs, and is simply parallelizable. We demonstrate the error correction and assembly performance of SGA on 1.2 billion sequence reads from a human genome, which we are able to assemble using 54 GB of memory. The resulting contigs are highly accurate and contiguous, while covering 95% of the reference genome (excluding contigs <200 bp in length). Because of the low memory requirements and parallelization without requiring inter-process communication, SGA provides the first practical assembler to our knowledge for a mammalian-sized genome on a low-end computing cluster.

[Supplemental material is available for this article.]

The cost of DNA sequencing continues to fall rapidly, faster than Moore's law for computing costs (Stein 2010). To keep pace with the increasing availability of sequence data, ever more efficient analysis algorithms are needed. This is of particular importance when performing de novo assembly of large genomes where data sets can be up to hundreds of gigabases in size. As de novo assembly typically requires performing queries over the entire set of sequence reads, very large data sets present a practical problem for the developers and users of assembly software. Currently available assemblers require either a single computer with very large amounts of memory—typically in the hundreds of gigabytes (Li et al. 2010b; Gnerre et al. 2011)—or a large distributed cluster of tightly coupled computers (Simpson et al. 2009; Boisvert et al. 2010). Recently Conway and Bromage (2011) described a method of encoding a de Bruijn graph using sparse bitmaps, and showed how this could be used in principle for genome sequence assembly in reduced memory.

We have pursued an alternative approach by developing algorithms that operate over a compressed representation of the full set of sequence reads. By using compressed data structures, we exploit the redundancy present in the collection of sequence reads to substantially lower the amount of memory required to perform de novo assembly. Previously, we described a space and time efficient algorithm (Simpson and Durbin 2010) to construct an assembly string graph (Myers 2005) from an FM-index (full-text minute-space index, Ferragina and Manzini 2000). Here we present a practical implementation including new and extended algorithms which perform queries over a compressed FM-index to error-correct, assemble, and scaffold large sets of sequence reads. We have implemented these algorithms in a new assembler called SGA (String Graph Assembler).

[1]Corresponding author.
E-mail rd@sanger.ac.uk.
Article published online before print. Article, supplemental material, and publication date are at http://www.genome.org/cgi/doi/10.1101/gr.126953.111. Freely available online through the *Genome Research* Open Access option.

Most short read assemblers rely on the de Bruijn graph model of sequence assembly, which requires breaking the reads into *k*-mers (Pevzner et al. 2001). By breaking up the reads, genomic repeats of length greater than *k* will be collapsed in a de Bruijn graph. Typically, de Bruijn graph assemblers attempt to recover the information lost from breaking up the reads, and attempt to resolve small repeats, using complicated read threading algorithms. SGA avoids this problem by using the string graph model of assembly. The string graph model keeps all reads intact and creates a graph from overlaps between reads (see Methods). SGA is one of the first assemblers to implement a string graph approach for assembling short reads and the first assembler to exploit a compressed index of a set of sequence reads. It is our view that compression based sequence analysis algorithms will become increasingly important as the number (and size) of full genome sequence data sets continues to grow. Because the most time-consuming parts of our algorithm are parallelizable without the need for inter-process communication, and our memory requirements for a human genome are under 64 GB per node, SGA now provides a practical assembler for a mammalian-sized genome on a low-end computing cluster.

Results

Algorithm overview

The SGA algorithm is based on performing queries over an FM-index constructed from a set of sequence reads. The SGA pipeline begins by preprocessing the sequence reads to filter or trim reads with multiple low-quality or ambiguous base calls. The FM-index is constructed from the filtered set of reads and base-calling errors are detected and corrected using *k*-mer frequencies. The corrected reads are re-indexed then duplicated sequences are removed, remaining low-quality sequences are filtered out, and a string graph is built. Contigs are assembled from the string graph and constructed into scaffolds if paired-end or mate-pair data are available. Figure 1 depicts the flow of data through the SGA pipeline. Further details are given in the Methods section below and in the Supplemental Material.

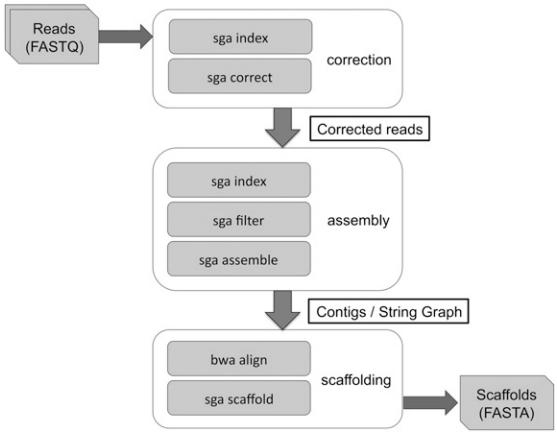

Figure 1. High-level diagram of the SGA assembly pipeline. The assembly has three main stages: error correction, contig assembly, and scaffolding. The error correction stage starts by building an FM-index for the reads (sga index) then performing error correction (sga correct). The assembly stage takes the corrected reads as input, re-indexes them, removes duplicate and low-quality reads, then constructs contigs. The scaffolding stage realigns the original reads to the contigs using BWA, constructs a scaffold graph using the alignments, and outputs a final set of scaffolds in FASTA format. For clarity, minor steps of the pipeline have been omitted from the diagram.

To demonstrate the performance of SGA and its ability to scale to large genomes we have performed error correction and assembly across a range of genome sizes, from bacteria to mammalian. In the following sections, we focus on the assembly of contigs and scaffolds from a single library of high quality short-range paired-end reads. We note however that in doing so we are not addressing the ability to build very large scaffolds using additional long range mate-pair data.

Assembly performance assessment

To assess the performance of SGA we performed assemblies of the nematode *Caenorhabditis elegans* using SGA and three other assemblers. The Velvet assembler (Zerbino and Birney 2008) was one of the first de Bruijn graph-based assemblers for short reads and has become a standard tool for assembling small- to medium-sized genomes. The ABySS assembler (Simpson et al. 2009) was developed to handle large genomes by distributing a de Bruijn graph across a cluster of computers. SOAPdenovo is also based on the de Bruijn graph and designed to assemble large genomes (Li et al. 2010a,b).

C. elegans provides a good real-world test case for assembly algorithms because it has a complete and accurate reference sequence (C. *elegans* Sequencing Consortium 1998), it propagates as a hermaphrodite so the genome of an individual (or strain) is homozygous and essentially free of SNPs and structural variants, and the genome is a reasonable size for evaluation (100 Mbp). We downloaded *C. elegans* sequence reads (strain N2) from the NCBI SRA (accession SRX026594). The data set consists of 33.8M read pairs sequenced using the Illumina Genome Analyzer II. The mean DNA fragment size is 250 bp from which reads of length 100 bp were taken from both ends of the fragment. To reduce the impact of differences between the sequenced individual and the reference sequence, we called a new consensus sequence for the *C. elegans* reference genome (build WS222, www.wormbase.org) based on alignments of the reads to the reference (see Supplemental Methods).

As sequence assemblers are often sensitive to the input parameters, we performed multiple assembly runs with each assembler. The de Bruijn graph assemblers were run for all odd *k*-mer sizes between 51 and 73 (inclusive). The *k*-mer size providing the largest scaffold N50 was selected for further analysis (67 for ABySS, 61 for Velvet, 59 for SOAPdenovo). Similarly, for SGA the *k*-mer size used for error correction and the minimum overlap parameter for assembly were selected to provide the largest scaffold N50 ($k = 41$ for error correction, $\tau = 75$ for the minimum overlap). We also performed a SOAPdenovo assembly using the GapCloser program after scaffolding. GapCloser was able to fill in many gaps within scaffolds, which increased the contig N50 and genome coverage. However, these increases came at the cost of substantially lowered accuracy. In the following analysis, we use the SOAPdenovo assembly without using GapCloser. A comparison of the SOAPdenovo assembly before and after running GapCloser is presented in the Supplemental Material.

We broke the assembled scaffolds into their constituent contigs by splitting each scaffold whenever a run of "N" bases was found. We filtered the contig set by removing short contigs (<200 bp in length). The remaining contigs were aligned to the consensus-corrected reference genome using BWA-SW (Li and Durbin 2010) with default parameters. We considered a number of different assessment criteria, which are described below and summarized in Table 1.

Substring coverage

For the first assessment, we sampled strings from the consensus sequence and tested whether they were exactly present in the contigs. We sampled 10,000 strings of length from 50 bp up to 5000 bp. This assessment combines three measures; the contigs must be accurate (as exact matches are required), complete (as the string must be present in the contig), and contiguous (as strings broken between multiple contigs will not be found). Figure 2 plots the proportion of strings found in the contigs as a function of the string length. All assemblers perform well for short strings (50 to 100 bp). For longer string lengths, SGA outperforms the other three assemblers.

Table 1. Assembly statistics for *C. elegans* data set

	SGA	Velvet	ABySS	SOAPdenovo
Scaffold N50 size	26.3 kbp	31.3 kbp	23.8 kbp	31.1 kbp
Aligned contig N50 size	16.8 kbp	13.6 kbp	18.4 kbp	16.0 kbp
Mean aligned contig size	4.9 kbp	5.3 kbp	6.0 kbp	5.6 kbp
Sum aligned contig size	96.8 Mbp	95.2 Mbp	98.3 Mbp	95.4 Mbp
Reference bases covered	96.2 Mbp	94.8 Mbp	95.9 Mbp	95.1 Mbp
Reference bases covered by contigs ≥1 kb	93.0 Mbp	92.1 Mbp	93.9 Mbp	92.3 Mbp
Mismatch rate at all assembled bases	1 per 21,545 bp	1 per 8786 bp	1 per 5577 bp	1 per 26,585 bp
Mismatch rate at bases covered by all assemblies	1 per 82,573 bp	1 per 18,012 bp	1 per 8209 bp	1 per 81,025 bp
Contigs with split/bad alignment (sum size)	458 (4.4 Mbp)	787 (7.2 Mbp)	638 (9.1 Mbp)	483 (4.4 Mbp)
Total CPU time	41 h	2 h	5 h	13 h
Max memory usage	4.5 GB	23.0 GB	14.1 GB	38.8 GB

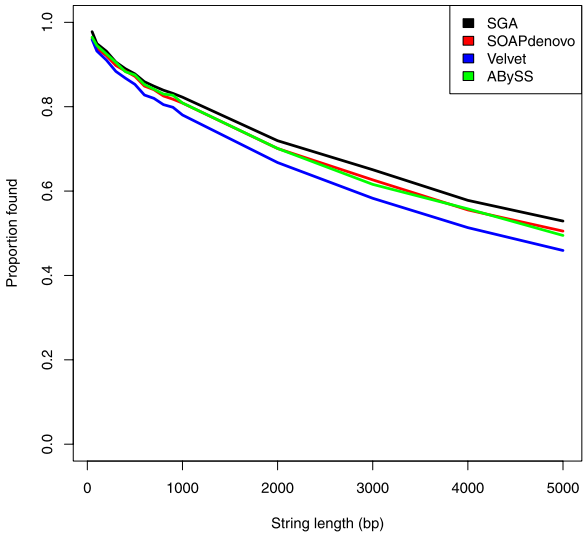

Figure 2. Reference string coverage analysis for the *C. elegans* N2 assembly. For string lengths from 50 bp up to 5000 bp, 10,000 strings were sampled from the consensus-corrected *C. elegans* reference genome. The proportion of the strings found in the SGA, Velvet, ABySS, and SOAPdenovo assemblies is plotted.

Assembly contiguity

We assessed the contiguity of the assemblies by calculating the contig alignment length N50. By analyzing the contig alignment lengths, as opposed to the length of contigs themselves, we account for misassembled contigs that can inflate the assembly statistics. For SGA, contig alignments 16.8 kbp and greater covered 50% of the reference genome (50 Mbp). ABySS, SOAPdenovo, and Velvet had contig alignment N50s of 18.4 kbp, 16.0 kbp, and 13.6 kbp, respectively.

Assembly completeness

The contigs assembled by SGA covered 95.9% of the reference genome. The ABySS assembly covered 95.6%, Velvet covered 94.5%, and SOAPdenovo covered 94.8%. Supplemental Figure 1 plots the reference genome coverage as a function of minimum contig alignment length.

Assembly accuracy

We assessed both the structural accuracy and the per-base mismatch rate of the contigs. First, we categorized the contig alignments into three groups. The first group ("full-length") contains contigs that had a single alignment to the reference containing at least 95% of the contig length. The second group ("split") contained contigs that had two alignments to the same chromosome in close proximity (<10,000 bp). These split contigs may either contain local assembly errors, or structural variation (for example a large insertion or deletion) with respect to the reference. All remaining alignments ("bad") were partially aligned (<95% of the contig aligned to the reference), aligned to multiple chromosomes, aligned in greater than two pieces, or did not align to the reference at all. For all assemblies a substantial proportion of the contigs was found to match the *Escherichia coli* genome. As *C. elegans* eat *E. coli*, this is an expected contaminant and one might suspect other bacterial sequences to also be present. For this reason contigs that did not align to the *C. elegans* reference were not included in this analysis.

For the first measure of assembly accuracy, we counted the number and total size of contigs with split or bad alignments. The accuracy of the SGA and SOAPdenovo contigs was similar—458 contigs for SGA (totaling 4.4 Mbp) and 483 contigs for SOAPdenovo (4.4 Mbp) had split or bad alignments. Velvet and ABySS had 787 contigs (7.2 Mbp) and 638 contigs (9.1 Mbp) with split or bad alignments, respectively.

For the second accuracy assessment, we calculated the rate at which aligned contig bases did not match the reference. In this assessment, we used the fully aligned contigs only. We evaluated each assembly at all reference positions covered by its contigs, and also at the subset of positions that were covered by all assemblies. The latter case provides a fairer basis for comparison, removing the effect of differences of coverage of repetitive or complex sequence between the four assemblies. The results are summarized in Table 1. Again, the accuracy of the SGA and SOAPdenovo assemblies was comparable, and both were more accurate than Velvet and ABySS. The mismatch rate of the SGA assembly at reference positions assembled by all four programs was ~1 mismatch per 83 kbp. SOAPdenovo, Velvet, and ABySS had error rates at shared positions of 1 per 81 kbp, 1 per 18 kbp, and 1 per 8 kbp, respectively.

Computational requirements

Of the four assemblers, SGA used the least memory (4.5 GB vs. 14.1 GB, 23.0 GB and 38.8 GB for ABySS, Velvet and SOAPdenovo, respectively). The de Bruijn graph assemblers were considerably more computationally efficient, however, as the SGA assembly required approximately eight times more CPU hours than ABySS, 20 times more than Velvet, and three times more than SOAPdenovo. This speed difference is largely due to the time required to build the FM-index. However, we can reuse one FM-index for multiple runs of SGA, for instance to try different error correction or assembly parameters, whereas the de Bruijn table for ABySS, Velvet, and SOAPdenovo must be recalculated for each choice of *k*.

Whole human genome assembly

As a second demonstration, we assessed the ability of SGA to scale to very large data sets by assembling a human genome. We downloaded 2.5 billion reads (252 Gbp of sequence) for a member of the CEU HapMap population (identifier NA12878) sequenced by the Broad Institute (ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/working/20101201_cg_NA12878/NA12878.hiseq.wgs.bwa.raw.bam). The reads are 101 bp in length from a paired-end insert library of 380 bp mean separation. As the total sequence depth is 84×, we chose to only assemble half the data to reflect typical coverage depths seen for human shotgun sequence data sets.

We constructed an FM-index for subsets of 20 million reads at a time, then iteratively merged the sub-indices in pairs to obtain a single FM-index for the entire data set. We ran the error correction process using a cluster of computers. Each process used the full FM-index to correct 20 million reads. An FM-index was constructed for the corrected reads, duplicate and low-quality reads were removed, and nonbranching chains of reads were merged together. A string graph was constructed from the merged sequences using a minimum overlap parameter of 77. We realigned the reads to the resulting contig set using BWA (Li and Durbin 2009) and constructed scaffolds.

In total, the assembly took 1427 CPU hours across 140 wall clock hours, just under 6 d. The most compute intensive stages were error-correcting the reads and building the FM-index of the corrected reads, which each required 355 CPU hours. However, these stages were distributed across a cluster of computers by

simply splitting the input data, substantially reducing the elapsed (wall clock) time. We ran 123 indexing/merging processes and 63 correction processes; the elapsed time for these stages was 32 and 1 h, respectively. The post-correction read filtering stage—where duplicate and low-quality reads are discarded—was the memory high-water mark, requiring 54 GB of memory. Complete details of the running time and memory usage for each stage of the assembly can be found in Supplemental Table 1.

We also assembled the data with SOAPdenovo (Li et al. 2010b). We first error-corrected the reads using the SOAPdenovo error correction tool then performed three assemblies, with *k*-mer sizes 55, 61, and 67. The 61-mer assembly had the largest scaffold and contig N50 and was used for the subsequent analysis. The 61-mer SOAPdenovo assembly (including error correction) required 479 CPU hours across 121 wall clock hours. The maximum amount of memory used was 118 GB. As with the *C. elegans* assembly, we did not use the SOAPdenovo GapCloser.

We evaluated the assemblies in terms of contiguity, completeness and accuracy. As in the *C. elegans* analysis, we broke the assembled scaffolds into their constituent contigs, filtered out contigs <200 bp in length, then aligned the remaining contigs to the human reference genome (build GRC 37) using BWA-SW (Li and Durbin 2010).

The SGA contig alignments cover 2.69 Gbp of the human reference autosomes and chromosome X (95.0% of the non-"N" portions of these chromosomes). The SOAPdenovo contigs cover 2.65 Gbp of the human reference (93.6%). The SGA contig alignment N50 is 9.4 kbp and the SOAPdenovo contig alignment N50 is 6.6 kbp. The corresponding raw contig N50s are 9.9 kbp and 7.2 kbp. Figure 3 plots the amount of the reference genome covered by each assembly as a function of the minimum contig alignment length. Across all contig alignment lengths, the SGA assembly covered more of the reference genome than SOAPdenovo. In contrast, SOAPdenovo gave larger scaffolds (N50 length of 34.8 kbp compared with 25.1 kbp for SGA), though the single short insert library for this data set limits the ability to build larger scaffolds.

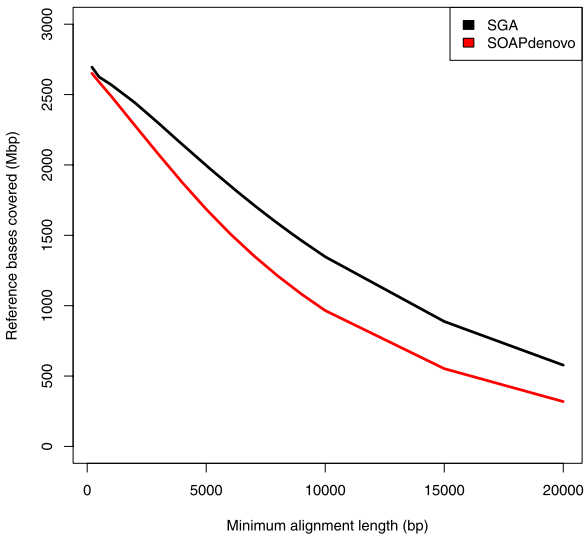

Figure 3. The amount of the human reference genome covered by a contig as a function of the minimum contig alignment length. For each length L on the *x*-axis, contig alignments less than L bp in length were filtered out and the amount of the reference genome covered by the remaining alignments was calculated.

The overall assembly accuracy for both SGA and SOAPdenovo was high; 94.5% of SGA contigs (totaling 2.64 Gbp) had full-length alignments to the reference genome, 1.1% (68 Mbp) had split alignments, and 4.3% (50 Mbp) had low-quality alignments or did not align at all; 96.8% of the SOAPdenovo contigs had a full-length alignment to the reference (totaling 2.60 Gbp), 1.0% had split alignments (53 Mbp), and 2.2% (33 Mbp) had low-quality alignments or did not align to the reference at all. This is consistent with the SGA assembly being a little larger, covering a little more of the reference but also containing a little more additional material.

We also calculated the per-base mismatch rate of the contigs using the same methodology as the *C. elegans* assembly. In this case, we used the human reference genome combined with SNP calls produced by the Broad Institute in the same individual from the same data set by a mapping rather than assembly based approach (Depristo et al. 2011). We only counted mismatches at positions that did not match the reference and did not match a Broad SNP call. We also calculated the mismatch rate at the subset of positions assembled by both SGA and SOAPdenovo. As both SNP calling and assembly can be confused by genomic repeats and segmental duplications, we also calculated the per-base accuracy at positions of the reference that are not masked by RepeatMasker (http://www.repeatmasker.org) and not annotated as segmental duplications (1.3 Gbp of the reference genome remains after this filter). Both assemblies were highly accurate. The mismatch rate for SGA over all covered positions of the reference was 1 per 3574 bp. For SOAPdenovo, the mismatch rate was 1 per 4285 bp. If we only consider reference positions covered by a contig from both assemblies, the mismatch rates are 1 in 4325 bp for SGA and 1 per 5041 bp for SOAPdenovo. When restricting the analysis to positions not masked by RepeatMasker and not annotated as segmental duplications, the mismatch rate is 1 per 52,464 bp for SGA and 1 per 51,125 for SOAPdenovo. At positions assembled by both programs and not masked as repeats or segmental duplications, the mismatch rates are 1 per 59,884 bp and 1 per 60,511 bp, for SGA and SOAPdenovo, respectively.

We note that both the contig mismatches and the mapping-based SNP calls will contain false-positive variants due to mapping errors between the contig or read sequences and the reference. These false positives will have an opposing effect; if the contig sequence is misaligned to the reference, we may count a mismatch in the assembly that is not truly present. This will cause the error rate in the assembly to be overestimated. It is also possible that false positives from misalignments in the mapping-based call set may overlap errors in the assembly. This would lead to an underestimate of the assembly error rate. As we cannot assess the magnitude of these effects, it is difficult to accurately estimate the true base-level error rate in the assemblies. However, if we conservatively consider all remaining mismatches to be assembly errors, it would indicate the per-base accuracy of the SGA and SOAPdenovo assemblies are very similar and better than 1 error in 50 kbp in nonrepetitive regions. The accuracy of SGA is supported by an independent assessment of our assembler's performed during the Assemblathon competition (see Discussion).

Error correction performance

The SGA error corrector is a standalone component of the assembly pipeline. We evaluated the error corrector on publicly available sequence data for *E. coli* K12 MG1655. As the reference genome for this strain is completely sequenced, we compared the error corrected reads with the reference genome to assess the performance

of the error correction algorithm. We constructed two read sets of 20× and 50× by randomly sampling from a data set of 14.2 million read pairs with read length 100 bp downloaded from the European Nucleotide Archive (accession ERA000206). For comparison we also ran the recently published error correction software Quake (Kelley et al. 2010) and HiTEC (Ilie et al. 2011). Like SGA, Quake is based on *k*-mer frequency analysis. HiTEC uses a suffix array to compute substrings of the reads that support each base call at a given position. As both SGA and Quake have multiple input parameters, most importantly the *k*-mer length, we ran multiple trials of these programs. Details of the parameters used are provided in the Supplemental Methods. HiTEC does not require parameter tuning, hence a single run was performed for each data set.

We evaluated the error correction performance in terms of yield (the number of bases aligned to the reference genome after correction), error rate after correction both in the reads and in assembled contigs made from the reads, and the assembly N50 of the corrected reads. We also evaluated the computational requirements of the three algorithms. The results are presented in Tables 2 and 3. Of the three programs, SGA and HiTEC had the largest assembly N50 after correction (17.1 kbp and 48.8 kbp for SGA, 15.2 kbp and 42.0 kbp for HiTEC for the 20× and 50× data sets, respectively). Quake had the lowest post-correction error rate in the reads and the highest number of perfect reads; however, the assembly N50 was significantly lower than the other two programs. For all three programs, the mismatch rate of the assembled contigs was much lower than that of the corrected reads. For the 50× data set, the mismatch rate of the contigs for the SGA-corrected data set was 1 per 328 kbp (1 per 208 kbp for Quake, 1 per 138 kbp for HiTEC). These differences are likely due to how uncorrectable reads are handled by the three programs. SGA and HiTEC report all reads, even if the reads contained errors that could not be corrected. Quake trims the end of a read if it detects an uncorrectable error. By trimming the reads, Quake is able to discard erroneous portions of reads that the other two programs kept. However, this trimming appears to remove effective coverage of some regions of the genome, which thereby lowered the assembly N50. SGA gives the best overall performance when evaluated in terms of assembly properties.

SGA required substantially less memory than the other two programs (2.5 times less than Quake and nine times less than HiTEC on the 20× data set). Notably, the memory usage for SGA was the same on the 20× and 50× data set as the memory high-water mark in both runs was building the Burrows-Wheeler transform for subsets of 100,000 reads, which were then merged into the final FM-index (see Distributed Construction of the FM-Index in the Methods).

Table 2. Error correction summary for *E. coli* 20× data set

	Uncorrected	SGA	Quake	HiTEC
Total bases	92.8 Mbp	92.8 Mbp	84.9 Mbp	92.8 Mbp
Aligned bases	83.4 Mbp	86.8 Mbp	82.3 Mbp	88.7 Mbp
Error rate	1 per 147 bp	1 per 1777 bp	1 per 7522 bp	1 per 1505 bp
Perfect reads	553,886	848,370	880,478	865,001
Assembly N50	—	17.1 kbp	9.6 kbp	15.2 kbp
Contig mismatch rate	—	1 per 188 kbp	1 per 111 kbp	1 per 21 kbp
Total CPU time	—	511 sec	368 sec	644 sec
Max memory use	—	206 MB	527 MB	1845 MB
Parameters	—	$k = 17$	$k = 16$, $c = 1$	—

Discussion

Genome assembly remains one of the most difficult computational problems in genomics. As DNA sequencing throughput has risen, it has increased again in importance in recent years. Numerous large-scale projects are under way to sequence unexplored genomes across a wide range of species (Genome 10K Community of Scientists 2009). Accurate and complete assembly of human genomes is still a problem that holds great interest as people begin to focus on individual differences. It has been recently demonstrated that with careful selection and assembly of large-insert mate-pair libraries highly contiguous and complete assemblies of mammalian genomes can be generated (Li et al. 2010a; Gnerre et al. 2011). However, a primary algorithmic challenge of assembly, the amount of memory required, has remained. It is this problem in particular that we have addressed in this paper. We have demonstrated that a full assembler can be developed using a compressed representation of the sequence reads and that a human genome can be assembled in under 64 GB of memory. This is an important milestone as it may now make it possible to assemble large genomes using commodity computing services like Amazon EC2.

We have compared SGA with three established de novo assemblers: ABySS, Velvet, and SOAPdenovo. The performance of SGA has also been validated in a complementary comparison by a recent collaborative de novo assembly assessment project, the Assemblathon, organized by UC Santa Cruz and UC Davis. The organizers publically released simulated paired-end and mate-pair reads from a 112 Mbp diploid genome generated by simulated evolution. In this assessment, SGA had the largest scaffold path N50 (a measure of scaffold length, corrected for assembly errors), the lowest number of substitution errors, and the second lowest number of structural errors (Earl et al. 2011). Overall, SGA placed third out of 17 groups, behind ALLPATHS-LG and SOAPdenovo (Li et al. 2010b; Gnerre et al. 2011). While the Assemblathon results are encouraging, they also help to identify areas in which we can improve. For example, the SGA scaffold lengths were comparable to the leading assemblies; however, the contig lengths in this comparison were shorter. This is likely due to our conservative post-scaffolding gap resolution procedure, an area in which we can improve.

Although the primary output of SGA is a haploid sequence, it also keeps track of all variant sequences present in the string graph that were removed during contig generation, together with the corresponding sequence that was retained. It is therefore possible to call heterozygous positions in a diploid genome, including both single nucleotide variants and insertions and deletions.

There are a number of other future avenues to pursue. Use of an FM-index rather than a hash table could allow dynamic selection of the *k*-mer parameter in error correction and the overlap threshold in sequence assembly, without recalculation of new indices. New sequencing technology is becoming available that promises longer read lengths (Korlach et al. 2010). Incorporating long read data with higher-coverage short reads is particularly well suited to the string graph model of assembly as the long reads do not need to be broken into *k*-mers. The compute time of SGA is largely dependent on the time required to construct a BWT

Table 3. Error correction summary for the *E. coli* 50× data set

	Uncorrected	SGA	Quake	HiTEC
Total bases	231.8 Mbp	231.8 Mbp	212.3 Mbp	231.8 Mbp
Aligned bases	208.5 Mbp	216.4 Mbp	205.8 Mbp	221.8 Mbp
Error rate	1 per 148 bp	1 per 1411 bp	1 per 28,174 bp	1 per 1538 bp
Perfect reads	1,386,148	2,099,410	2,221,460	2,164,055
Assembly N50	—	48.8 kbp	26.1 kbp	42.0 kbp
Contig mismatch rate	—	1 per 328 kbp	1 per 208 kbp	1 per 138 kbp
Total CPU time	—	1470 sec	669 sec	1738 sec
Max memory use	—	208 MB	585 MB	4339 MB
Parameters	—	$k = 17$	$k = 15$, $c = 4$	—

from a large collection of strings. By incorporating recent advances in this area (Bauer et al. 2011), we could substantially reduce the running time of our program. Finally, our compression-based analysis methods have applications outside traditional genome assembly. In particular the analysis of metagenomics data is a potential application of our algorithms as the data sets routinely reach hundreds of gigabases in size and reference genomes are typically not available for a substantial fraction of the species sequenced.

Methods

Overview

We previously described an algorithm to construct an assembly string graph (Myers 2005) for a set of error-free sequence reads using the FM-index (Simpson and Durbin 2010). Here, we expand upon this work and describe algorithms to correct base calling errors, remove duplicate sequences and construct contigs and scaffolds for real sequencing data. SGA is implemented as a modular pipeline, which allows it to be easily extended as improved algorithms are developed or sequencing technology changes. In this section we present the algorithmic principles of SGA, which are further described in the Supplemental Material.

Definitions and notation

Let X be a string of symbols $a_1,\ldots,a_l$ from an alphabet Σ. The length of string X is denoted by $|X|$. A substring of X is denoted by $X[i,j]$ where $1 \le i \le j \le |X|$. A substring of the form $X[1, i]$ is a prefix of X and a substring $X[k, |X|]$ is a *suffix* of X. We use $\overline{X}$ to denote the reverse complement of a string.

Let R be an indexed set of strings (or sequence reads) where R_i denotes the i-th string in the collection. For convenience, we consider each string in R to be terminated with a unique symbol $\$_i$, where $\$_i < \$_j$ iff $i < j$. The suffix array of R is defined as $\mathbf{SA}_R[i] = (j, k)$ iff $R_j[k, |R_j|]$ is the i-th lexicographically lowest suffix in R. The Burrows-Wheeler transform (Burrows and Wheeler 1994) of R can be defined in terms of the suffix array as follows. Let $\mathbf{SA}_R[i] = (j, k)$ be an element of the suffix array. The corresponding element of the BWT of R is:

$$B_R[i] = \begin{cases} R_j[k-1] & \text{if } k > 1 \\ \$ & \text{if } k = 1 \end{cases}.$$

FM-index fundamentals

The FM-index is a data structure developed by Ferragina and Manzini (2000) to allow efficient searching of a compressed representation of a text, T. The FM-index is built from the Burrows-Wheeler transform of T by defining two auxiliary data structures. Let $C(a)$ be the number of occurrences in T of symbols lexicographically lower than symbol a. Let $Occ(a,i)$ be the number of occurrences of a in the substring $B_T[1,i]$. Ferragina and Manzini showed that, with these two structures, the number of occurrences of a pattern W in T can be found in $O(|W|)$ time using an iterative pattern-growth procedure. The locations of W in T can be found in $O(|W| + n)$ time where n is the number of occurrences. These procedures, and our method for performing inexact overlap matches between reads using a seed-and-extend strategy, are described in the supplement.

The Burrows-Wheeler transform is a permutation of the original text. As the BWT sorts repeated substrings into contiguous intervals, the string B_T contains runs of repeated symbols. This allows efficient compression of B_T using run-length encoding. This compression strategy is particularly effective for high-coverage sequencing data as the lengths of runs in B_T are dependent on the depth of coverage. The SGA implementation of the FM-index encodes a <symbol, run> pair using a single byte to encode runs of up to 32 symbols. On average, five to eight symbols are stored per byte for high coverage (>20×) 100 bp read data, depending on the error rate of the reads.

Graph-based assembly

Graph-based assembly algorithms model the assembly problem as a set of strings (as vertices) and their relationship to each other (as edges). The problem of reconstructing the source genome can be cast in terms of finding a walk through the graph. An overlap graph is formed from a set of reads R by finding all pairwise overlaps of length at least τ between members of R. We say that two reads X and Y overlap when a suffix of X matches a prefix of Y or vice versa. In this case, we place a bidirected *SP*-edge (suffix/prefix) into the overlap graph linking X and Y. If a prefix of X matches a reverse-complemented prefix of Y, we place a *PP*-edge in the graph. Edges of type *SS* are defined similarly.

The string graph can be derived from the overlap graph by first removing duplicate reads (distinct elements of R with the same or reverse-complemented sequence) and contained reads (elements in R that are a substring of some element in R or their reverse complements), then removing transitive edges from the graph. We say that an edge $X \rightarrow Z$ is transitive if the graph contains the edges $X \rightarrow Y$ and $Y \rightarrow Z$ and the directions of the edges are compatible ($X \rightarrow Y$ and $X \rightarrow Z$ must both represent suffix [or prefix] overlaps for X and so on). Informally, the edge $X \rightarrow Z$ is transitive if the path $X \rightarrow Y \rightarrow Z$ "spells" the same string as $X \rightarrow Z$. We call the non-transitive edges of the overlap graph irreducible and the subgraph containing only the irreducible edges the string graph. Our previous work described an algorithm for directly constructing the string graph without the need to explicitly construct the full overlap graph. This was achieved by developing a function to compute the set of irreducible overlaps for a given read using the FM-index. We refer to Simpson and Durbin (2010) for further details.

Distributed construction of the FM-index for large read sets

We begin with the construction of the FM-index for the complete set of reads. A naive algorithm would first build a suffix array of R from which B_R is easily computed. The full suffix array requires $n\log(n)$ bits of memory where n is the total number of bases in the read set. For a human genome sequenced to 30× coverage, this

would require over 400 GB of memory, a prohibitively large amount. To address this, we have implemented a distributed construction algorithm that builds an FM-index for each subset of R, R_1,R_2,..R_m. We then iteratively merge pairs of the intermediate indices together using a BWT merging algorithm (Ferragina et al. 2010) until a single index of the entire data set is obtained. As the space occupancy of the FM-index is typically less than an order of magnitude smaller than that of a suffix array, this indexing strategy allows us to efficiently build the FM-index for very large sequence collections. This construction strategy can be easily parallelized as the construction of the FM-index for each read subset, and most merging operations, can be computed independently.

Error correction algorithm

Real sequencing data contains base calling errors. SGA error correction is currently designed to handle substitution errors, which are the dominant error mode in the Illumina sequencing platform (Bentley et al. 2008). We have implemented two error correction methods. The first is a *k*-mer frequency-based corrector, which has been successfully used in other sequence assemblers (Pevzner et al. 2001; Li et al. 2010b). In our implementation, the *k*-mer frequencies are not stored in a lookup or hash table but rather directly calculated from the FM-index. This has the advantage of using less memory and allowing greater flexibility in the parameter choices as the FM-index can support any value of k unlike a hash table, which must be reconstructed for each choice of k.

The *k*-mer correction algorithm begins by classifying each base call in a given sequence read as trusted or untrusted based on *k*-mer frequencies. If position i is covered by a *k*-mer that is seen in R at least c times, it is marked as trusted. For the positions that are not trusted, we test if an alternative base call yields a *k*-mer covering the position that is seen more than c times. If no valid correction exists, or multiple valid corrections are possible, the base is left unchanged and the error correction for the read terminates. If a single correction is possible, the correction is accepted and the procedure continues until all positions are trusted. The minimum coverage parameter c is conservatively chosen to avoid collapsing SNPs (if the genome is diploid) or distinct copies of a repeat. This parameter can either be manually provided or automatically selected by SGA by finding a trough in the *k*-mer frequency histogram.

The second correction method implemented in SGA is based on inexact overlaps between reads. A description of this method can be found in the Supplemental Material. The *k*-mer based corrector is considerably faster than the overlap-based corrector (approximately two times faster on the 20× *E. coli* data set presented in the results) and is the default method of correction in SGA. Both correction methods have an option to use sequence base quality of the read being corrected to vary the coverage threshold c required to support a base call.

Read filtering

To construct the string graph we require a subset of R consisting of unique reads. We achieve this by removing exact contained and duplicate reads. To compute this subset, we use the FM-index to calculate full-length matches for each read in R. If a read R_i has a full-length match (including reverse complements) to some other read R_j we keep R_i iff $i < j$, otherwise R_i is discarded. Once the unique subset U of R has been calculated, we do not need to recompute the FM-index of U from scratch. The BWT of U can be derived from the FM-index of R by marking the positions in B_R that correspond to reads that were discarded and exporting the unmarked positions as B_U (Sirén 2009).

Some reads remain uncorrected after error correction. To prevent these sequences from impacting the assembly, we remove sequences with unique *k*-mers. By default, this filter requires all 27-mers in a read to be seen at least twice.

Read merging and assembly algorithm

After correction and filtering, the vast majority of the remaining reads do not contain errors. We could directly apply our string graph construction algorithm to these. However, the resulting graph would have a vertex for every read and therefore require a substantial amount of memory when assembling very large genomes. The majority of reads in the initial graph are simply connected (that is, without branching) to two other reads—one matching a prefix of the read and one matching a suffix. This chain of reads can be unambiguously merged to reduce the size of the graph. We have developed an algorithm to locally construct the assembly graph around each read. For each read, we find the predecessor and successor vertices in the graph by querying the FM-index for the irreducible edge set. If the read connects to its neighbors without branching, we continue the search from the neighboring reads. This search stops when a branch in the graph is found. This procedure will discover all nonbranching chains in the graph and allow the chain to be replaced by a single merged sequence. As the predecessor/successor queries only require the FM-index, this merging step requires comparatively little memory when compared to loading the full graph. Once we have performed this merging step, we build an FM-index for the merged sequences and use this FM-index to construct the full string graph. We then perform the standard assembly graph post-processing step of removing tips from the graph where a vertex only has a connection in one direction (Chaisson and Pevzner 2008; Zerbino and Birney 2008; Simpson et al. 2009; Li et al. 2010b).

To account for heterozygosity in a diploid genome, we have developed an algorithm to find and catalog variation described by the structure of the graph, similar to the "bubble-popping" approaches taken by de Bruijn graph assemblers. Let v be a vertex in the graph which branches (the prefix or suffix of v has multiple overlaps). Following each branch, we search outwards from v for a set of walks, W, which meets the following conditions: (1) All walks terminate at a common vertex u and (2) no vertex visited in any walk between v and u has an edge to a vertex that is not present in a walk in W. The first condition ensures that the walks describe equivalent sequence in G—any assembly of G that visits v and u must use one of the found walks. The second condition ensures that the induced subgraph of G described by the walks is self-contained—we can remove any walk in W without breaking any walk in $G \setminus W$. Once a set of walks meeting these conditions has been found, we select one of the walks to remain in the graph. We align the sequence described by the other walks to the sequence of the selected walk and, if the sequence similarity is within tolerance (by default 95%) in all cases, the nonselected walks are removed from the graph. We retain the sequences of the removed walks in a FASTA file to allow the variation present in the genome to be analyzed after assembly.

Paired-end reads/scaffolding

The final stage of the assembly is to build scaffolds from the contigs using paired-end or mate-pair data. Similar to other approaches to scaffolding (Pop et al. 2004), our method is based on constructing a graph of the relationships between contigs. We begin by realigning the paired reads to the contigs using BWA. The copy number of each contig in the source genome is estimated from the read alignments using Myers' A-statistic which approximates the log-odds ratio be-

tween the contig being unique and a collapsed repeat (Myers 2005). By default, we classify contigs with an A-statistic ≥20 as unique and the remainder as repetitive. We construct a scaffold graph where each unique contig is a vertex. Contigs linked with read pairs are connected by a bidirected edge labeled with the estimated gap size separating the contigs. Paths through this scaffold graph describe layouts of the contigs into scaffolds.

Our scaffolder first removes ambiguous or likely erroneous edges from the graph. For each contig in the graph with more than one edge in a particular direction, we test whether the linked contigs have an ordering that is consistent with each pairwise distance estimate. An ordering of contigs $C_1,C_2,\ldots,C_n$ is called consistent if no pair of contigs has an overlap (implied by their positions in the layout) greater than α bases ($\alpha = 400$ by default). If the contigs cannot be consistently ordered, we break the graph by removing all edges of the affected contigs.

Once the graph has been cleaned of inconsistent edges, we find and isolate any directed cycles then compute the connected components of the graph. For each connected component, we find the terminal vertices of the component (vertices that have an edge in only one direction) and find all paths between each pair of terminal vertices. The path containing the largest amount of sequence is retained as the primary layout of the scaffold. We note that, although each of the assemblies presented in this paper is based on a single library, the SGA scaffolder supports multiple libraries of different sizes, as for example presented in the Assemblathon paper (Earl et al. 2011).

The scaffolds are represented as an alternating list of contigs and gaps—$C_1,g_1,C_2,g_2,\ldots,C_n$. We attempt to fill in the gaps through a two-stage process. Let C_i and C_j be two adjacent contigs separated by a distance of g_i. As C_i and C_j are vertices in the string graph we previously constructed, we search the string graph for a walk connecting these vertices with the constraint that the total walk length can be no larger than $|C_i| + |C_j| + g_i + \theta_i$ where θ_i allows for the inexact distance estimate (by default three times the standard error of the distance estimate). If a single walk is found to meet this constraint, we replace C_i,g_i,C_j in the scaffold by the walk string. If no walk can be found connecting the vertices and g_i is negative (the contigs are predicted to overlap), we align the ends of C_i and C_j. If the predicted overlap is confirmed to exist, the sequences of C_i and C_j are merged. If the gap cannot be resolved, we simply fill the sequence between C_i and C_j with g_i "N" symbols.

Software availability

SGA is open source and freely available at https://github.com/jts/sga.

Acknowledgments

J.T.S. is supported by a Wellcome Trust Sanger Institute Research Studentship. R.D. is funded by Wellcome Trust grant WT077192. We thank Mark DePristo for access to the NA12878 data set.

References

Bauer MJ, Cox AJ, Rosone G. 2011. Lightweight BWT construction for very large string collections. In *Proceedings of the twenty-second annual symposium, Combinatorial Pattern Matching*, pp. 219–231. Springer-Verlag, Berlin, Heidelberg.

Bentley DR, Balasubramanian S, Swerdlow HP, Smith GP, Milton J, Brown CG, Hall KP, Evers DJ, Barnes CL, Bignell HR, et al. 2008. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* **456:** 53–59.

Boisvert S, Laviolette F, Corbeil J. 2010. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *J Comput Biol* **17:** 1519–1533.

Burrows M, Wheeler DJ. 1994. *A block-sorting lossless data compression algorithm*. Digital SRC Research Report. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.6774.

C. elegans Sequencing Consortium. 1998. Genome sequence of the nematode *C. elegans*: a platform for investigating biology. *Science* **282:** 2012–2018.

Chaisson MJ, Pevzner PA. 2008. Short read fragment assembly of bacterial genomes. *Genome Res* **18:** 324–330.

Conway TC, Bromage AJ. 2011. Succinct data structures for assembling large genomes. *Bioinformatics* **27:** 479–486.

Depristo MA, Banks E, Poplin R, Garimella KV, Maguire JR, Hartl C, Philippakis AA, Del Angel G, Rivas MA, Hanna M, et al. 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat Genet* **43:** 491–498.

Earl D, Bradnam K, St John J, Darling A, Lin D, Fass J, Yu HO, Buffalo V, Zerbino DR, Diekhans M, et al. 2011. *Genome Res* **21:** 2224–2241.

Ferragina P, Manzini G. 2000. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pp. 390–398. IEEE Computer Society, Washington, DC. http://dx.doi.org/10.1109/SFCS.2000.892127.

Ferragina P, Gagie T, Manzini G. 2010. Lightweight data indexing and compression in external memory. http://arxiv.org/abs/0909.4341.

Genome 10K Community of Scientists. 2009. Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. *J Hered* **100:** 659–674.

Gnerre S, Maccallum I, Przybylski D, Ribeiro FJ, Burton JN, Walker BJ, Sharpe T, Hall G, Shea TP, Sykes S, et al. 2011. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci* **108:** 1513–1518.

Ilie L, Fazayeli F, Ilie S. 2011. HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics* **27:** 295–302.

Kelley DR, Schatz MC, Salzberg SL. 2010. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol* **11:** R116. doi: 10.1186/gb-2010-11-11-r116.

Korlach J, Bjornson KP, Chaudhuri BP, Cicero RL, Flusberg BA, Gray JJ, Holden D, Saxena R, Wegener J, Turner SW. 2010. Real-time DNA sequencing from single polymerase molecules. *Methods Enzymol* **472:** 431–455.

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

Li H, Durbin R. 2010. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics* **26:** 589–595.

Li R, Fan W, Tian G, Zhu H, He L, Cai J, Huang Q, Cai Q, Li B, Bai Y, et al. 2010a. The sequence and de novo assembly of the giant panda genome. *Nature* **463:** 311–317.

Li R, Zhu H, Ruan J, Qian W, Fang X, Shi Z, Li Y, Li S, Shan G, Kristiansen K, et al. 2010b. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res* **20:** 265–272.

Myers EW. 2005. The fragment assembly string graph. *Bioinformatics* (Suppl 2) **21:** ii79–ii85.

Pevzner PA, Tang H, Waterman MS. 2001. An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci* **98:** 9748–9753.

Pop M, Kosack DS, Salzberg SL. 2004. Hierarchical scaffolding with Bambus. *Genome Res* **14:** 149–159.

Simpson JT, Durbin R. 2010. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics* **26:** i367–i373.

Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. 2009. ABySS: a parallel assembler for short read sequence data. *Genome Res* **19:** 1117–1123.

Sirén J. 2009. Compressed suffix arrays for massive data. In *SPIRE '09, Proceedings of the 16th International Symposium on String Processing and Information Retrieval*, pp. 63–74. Lecture Notes in Computer Science, Vol. 5721. Springer-Verlag, Berlin.

Stein LD. 2010. The case for cloud computing in genome informatics. *Genome Biol* **11:** 207. doi: 10.1186/gb-2010-11-5-207.

Zerbino DR, Birney E. 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res* **18:** 821–829.

Received May 31, 2011; accepted in revised form December 5, 2011.