JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 17, Number 11, 2010
© Mary Ann Liebert, Inc.
Pp. 1519–1533
DOI: 10.1089/cmb.2009.0238

Ray: Simultaneous Assembly of Reads from a Mix of High-Throughput Sequencing Technologies

SÉBASTIEN BOISVERT,[1,2] FRANÇOIS LAVIOLETTE,[3] and JACQUES CORBEIL[1,2]

ABSTRACT

An accurate genome sequence of a desired species is now a pre-requisite for genome research. An important step in obtaining a high-quality genome sequence is to correctly assemble short reads into longer sequences accurately representing contiguous genomic regions. Current sequencing technologies continue to offer increases in throughput, and corresponding reductions in cost and time. Unfortunately, the benefit of obtaining a large number of reads is complicated by sequencing errors, with different biases being observed with each platform. Although software are available to assemble reads for each individual system, no procedure has been proposed for high-quality simultaneous assembly based on reads from a mix of different technologies. In this paper, we describe a parallel short-read assembler, called Ray, which has been developed to assemble reads obtained from a combination of sequencing platforms. We compared its performance to other assemblers on simulated and real datasets. We used a combination of Roche/454 and Illumina reads to assemble three different genomes. We showed that mixing sequencing technologies systematically reduces the number of contigs and the number of errors. Because of its open nature, this new tool will hopefully serve as a basis to develop an assembler that can be of universal utilization (availability: http://deNovoAssembler.sf.Net/). For online Supplementary Material, see www.liebertonline.com.

Key words: de Bruijn graphs, genome assembly, high-throughput sequencing.

1. INTRODUCTION

The availability of high-throughput sequencing has altered the landscape of genome research. The sequencing technologies provide a tremendous amount of sequences at a lower cost with increasing lengths of reads (Medini et al., 2008). With these sequences gathered, computational methods enable genome-wide analysis, comparative genomics studies, the hunt for resistance mutations, and the general study of molecular evolution among others with the constant of requiring accurately-assembled genomes. To decode a genome, its DNA is sheared in fragments, which are decoded with sequencers to produce reads. Reads are assembled by assemblers into larger sequences called contigs.

[1]Département de Médecine Moléculaire and [3]Département d'Informatique et de Génie Logiciel, Université Laval, Québec, Canada.
[2]Infectiologie et Immunologie, Centre de Recherche du CHUQ, Québec, Canada.

Three types of assemblers were previously introduced—greedy assemblers, overlap-layout-consensus assemblers, and de Bruijn assemblers (Pop, 2009). Owing to a minimal overlap length between reads, overlap-layout-consensus assemblers are unable to assemble short reads. The dated assemblers tailored for Sanger sequencing were not designed to tackle the large datasets produced by the new sequencing technologies (Pop, 2009). Although the de Bruijn framework was described as a reliable workhorse for Sanger sequencing projects (Pevzner et al., 2001), few de Bruijn assemblers were available when high-throughput sequencing became widely available. The de Bruijn graph was deemed well-suited for assembly of short reads (Chaisson et al., 2004). A de Bruijn graph allows a compact representation of millions of short (<50 nucleotides) reads (Zerbino and Birney, 2008), where short sequences (k-mers) occuring in reads are only stored once.

Numerous assemblers for high-throughput sequencing are readily available, but none can simultaneously assemble reads from different sequencing technologies. On the other hand, Ray can simultaneously assemble in parallel reads from a mix of sequencing systems.

2. THE ASSEMBLY PROBLEM

In a typical setting, genome fragments are sequenced to produce reads, followed by de novo assembly of these reads. A contig is a contiguous sequence produced by an assembler, and represents a region of the genome. A contig can contain errors—their types are described below. The length of a contig is counted in nucleotides (nt) or basepairs (bp) when we infer the double-stranded DNA molecule. Each algorithm that solves the assembly problem (AP) constructs contigs from a family of reads. We use the terminology of family instead of set because a read can appear more than once in a family.

For a one-chromosome genome, one strives to obtain a contig that is exactly the genome sequence, but this is nearly impossible in practice because of long repeated regions. A given contig can be compared to the genome sequence to find assembly errors. Among contigs that deviate from the genome sequence, one can find the following error types. A chimeric contig (or incorrect contig) contains two (or more) sequences that are not contiguous in the genome. A mismatch occurs in a contig at a particular position if it aligns on the genome sequence, but at this position the letter is not the same. An insertion in a contig is characterized by the addition of a (short) sequence of letters at some position, in respect to the genome sequence. In a similar fashion, a deletion in a contig is characterized by the removal of a (short) sequence of letters at some position, in comparison with the genome sequence. An indel is an insertion or a deletion.

A read can be sampled from the forward strand or the reverse-complement strand in the genome. Reads are generated from the alphabet $\{A, T, C, G, N\}$, where N represents ambiguous bases. A read is, in general, short (36–420 nt) compared to even the smallest bacterial genomes, and individual reads may contain sequencing errors.

In addition, several protocols exist to sequence the extremities of longer molecules in order to generate paired reads. They are critical to genome assemblies, as they provide linkage between sequences separated by a variety of physical distances. For a fragment library, we denote the average length of such molecules by d_μ and the standard deviation with d_σ. For each library, d_μ and d_σ are calculated by Ray automatically. Paired reads provide knowledge of the sequence content of each of the reads in the pair, but also knowledge of the orientation (DNA strand) of each read, and an estimation of the distance (amount of intervening sequence) between them (Scheibye-Alsing et al., 2009).

The quality of a specific assembly must be evaluated through some criteria. The length of the contigs is of course one of them. This leads us to the following definition of the assembly problem.

Definition 1. *Given a family of reads (which can contain errors, be paired, and be from different sequencing technologies), the assembly problem (denoted AP) consists in constructing contigs such that:*

1. *the breadth of coverage (how much of the genome is represented) is maximal;*
2. *the number of assembly errors (chimeric contigs, mismatches, insertions, and deletions) is minimal;*
3. *the number of contigs is minimal.*

The minimality of the number of contigs indirectly minimizes the redundancy, i.e. contigs whose positions in the genome sequence overlap. A contig can be interpreted as a mosaic of reads and the depth of coverage of a position is the number of reads covering it in the mosaic. Uneven genome coverage can complexify the assembly problem, because the classification of a region as being repeated or not is more difficult. In the assembly problem, it is assumed that, on average, each position is covered numerous times (>10×). This allows us to expect that most of the positions will be covered in multiple independant reads, and helps resolve sporadic sequencing errors in occasional reads. The difficulty of the assembly problem lies on the quality of the reads, the uneven coverage, and the complexity of the genome (large in size, non-random sequence composition, and repetitive elements often longer than existing read lengths).

3. SEQUENCING TECHNOLOGIES

Genome variation is responsible for biological diversity and sequencing enables its assessment. Sequencing started with the chain-terminating inhibitor method by Sanger et al. (1977) in which subsequences are randomly stemmed from a sequence template by targeted polymerase chain reaction, which is coupled with chain termination. This method was automated with a computer and fluorescence detection (Smith et al., 1986). The newly-developed concepts of massively-parallel flow-cell sequencing and sequencing-by-synthesis were fundamental shifts in genomics (Holt and Jones, 2008). The new sequencing technologies—Roche/454, Illumina, and SOLiD—produce shorter reads in comparison with the Sanger method. Other forthcoming sequencing technologies promise longer reads. Short reads are sufficient for de novo sequencing if they are paired (Chaisson et al., 2008). Emerging sequencing technologies are evolving fast—the continuing evolution of these technologies, and others in development, is a strong argument for the need of assemblers that are technology (and read length) independant.

3.1. Sequencing errors

Each sequencing technology makes specific errors. The mismatches in Roche/454 and Illumina reads occur mostly at random and the most common Roche/454 sequencing errors are carry-forward errors (because of leftover unincorporated nucleotides), and homopolymer-related errors (because of the absence of reversible terminators) (Rothberg and Leamon, 2008). Some of these errors are correlated (the errors in homopolymer runs are correlated), which makes the AP harder, and leads us to consider algorithms that run on mixed technologies. In the Illumina technology, one nucleotide is incorporated in each cycle (because of reversible terminators), regardless if homopolymers are present. In contrast, the Roche/454 system allows the incorporation of all nucleotides of a homopolymer during the same cycle. The homopolymer problem arises because the intensity of emitted light is linear when the number of incorporated nucleotides in the homopolymeric region is low, but is not linear when too many nucleotides are incorporated at once.

4. ASSEMBLY ALGORITHMS

Since the release of the first assembled genome, computer-aided methods have evolved along with the evolution of sequencing apparatus. Over the course of this evolution, a number of algorithmic pathways were followed and quickly rendered obsolete by the sheer volume outputted by the new parallel sequencing technologies. Available assemblers are dedicated to specific and proprietary technologies. Our assembler Ray is, on the contrary, not dedicated to any specific technology, parallel, and open source. One aim of our algorithm is to allow the simultaneous use of reads produced by different technologies. What follows is a brief description of assembler types, along with a list of the most important related algorithms.

4.1. Overlap-layout-consensus

Intuitively, finding read overlaps should be the primary idea behind assembly. The overlap-layout-consensus framework is merely a three-step process (Myers et al., 2000). First, the overlaps between reads are computed. Second, the order and orientation of reads in the assembly are determined. Third, the consensus allows the determination of the nucleotide at each position in the contigs.

Assemblers implementing this idea are numerous, and were crafted to overcome the assembly hurdles in Sanger-technology projects before high-throughput systems were developed. These software are tailored for the assembly of long reads, such as Sanger reads. They include the Celera assembler (Myers et al.,

2000) and Arachne (Batzoglou et al., 2002). Afterwards, the paradigm was adapted to the Roche/454 system. The Roche/454 sequencer is distributed with Newbler (Margulies et al., 2005).

Also, EDENA is an overlap-layout-consensus assembler that can assemble short reads (35 bases) (Hernandez et al., 2008).

4.2. Greedy assemblers

Using overlaps between reads to build contigs is intuitive. Algorithms can be built upon this idea. A greedy algorithm iteratively grows contigs by choosing best overlaps first. Implementations of the greedy algorithm were the first to be introduced for the short read technologies: SSAKE (Warren et al., 2007), VCAKE (Jeck et al., 2007), and SHARGCS (Dohm et al., 2007).

4.3. Assembly with de Bruijn graphs

The introduction of the de Bruijn graph for the AP is motivated by redundant information in reads: a large depth of coverage implies that a lot of overlaps between reads occurs. In presence of such redundant information, de Bruijn assemblers can solve the assembly problem with a memory usage bounded by the genome length (Zerbino and Birney, 2008).

Before introducing the de Bruijn graph, we need to give some definitions. For the alphabet $\Sigma = \{A, T, C, G, N\}$, Σ^* is the associated set of strings. For a string x, $|x|$ denotes its length, $x[i]$ is the letter (or symbol) at position i (starting at 1), and $x[i..j]$ is the substring of x from positions i to j. For strings x and y, xy denotes their concatenation.

We denote the reverse-complement sequence of $x \in \Sigma^*$ with revcomp(x). The definition is as follows: for a sequence x, its reverse-complement, $y = \text{revcomp}(x)$, is defined as $\text{complement}(y[i]) \overset{\text{def}}{=} x[|x| - (i-1)]$, where the complement is defined as usual: complement(A) = T, complement(T) = A, complement(C) = G, complement(G) = C and complement(N) = N. Here is an example: revcomp($AGGGAT$) = $ATCCCT$.

Finally, a k-mer is a sequence of length k generated with the alphabet of nucleotides. Using these concepts and notations, we define the de Bruijn graph as follows.

Definition 2. *Given an alphabet $\Sigma = \{A, T, C, G\}$ and a integer k, a full de Bruijn graph G has vertices $V(G) = \Sigma^k$, and arcs $\{\langle x, y\rangle, x, y \in V(G), x[2..k] = y[1..k-1]\}$*

where Σ^k is the set of all possible k-mers with Σ. Note that, in a de Bruijn graph, a vertex represents a sequence of k letters (a k-mer), an arc, being composed of two k-mers that overlap on $k-1$ letters, can be viewed as a sequence of $k+1$ letters (a $(k+1)$-mer), and more generally, a walk of n vertices corresponds to a sequence of $n+k-1$ letters. A walk, in an oriented graph, can contain several times the same vertex, akin to walking on the graph respecting the orientation of the arrows. In the de Bruijn framework, a genome sequence of l letters is a walk in the de Bruijn graph, and has $l-k+1$ vertices. We have a similar observation for contigs. In the case of reads, the same holds with the exception that k-mers containing the symbol N are not in the corresponding walk.

Given a full de Bruijn graph G, and a family of reads $D \subseteq \{A, T, C, G, N\}^*$, one can build a subgraph of G by retaining only the vertices and the arcs in the graph that correspond respectively to k-mers and $(k+1)$-mers present in the reads of D, or in reverse-complement reads of D. The value of the integer k is a parameter that has to be fixed for each de Bruijn algorithm, and 19 is the lowest sensible value (we use 21) (Flicek and Birney, 2009). With this value, we have a rich set of vertices in the full de Bruijn graph ($4^{21} = 4{,}398{,}046{,}511{,}104$). Moreover, $k = 21$ is significantly shorter than the length of the reads given by the present sequencing technologies. Recall that for any read, the strand on the genome sequence from which it was generated is unknown. de Bruijn graphs based on reads of a family D will therefore be constructed based on all the reads of D together with all the reverse-complement reads of D.

In the particular case of a pair of reads $(r_1, r_2, d_\mu, d_\sigma)$, four subsequences will be considered for the constructed de Bruijn graph, namely r_1, r_2 and their reverse-complement sequences, where r_1 and r_2 are read sequences of a fragments library, and d_μ is the average fragment length observed for the library (with standard deviation d_σ).

If the reads contain errors, particular structures in the graph—bubbles and tips—will be present (Zerbino and Birney, 2008). A bubble is a structure between two vertices, such that two disjoint (short) walks exist between these vertices, and the sequence of letters of each walk is very similar. A tip is a hanging walk in the graph leading to a dead-end, and whose length is short.

In the case where reads are error-free and cover the whole genome and when there are no repeated sequences with more than k letters, the solution of the AP is a single walk in the de Bruijn graph that goes through each arc exactly once. Such a walk is called Eulerian. In the case where the two strands of the genome have been sequenced without errors, the solution is not a single walk, but the union of two walks which collectively use each edge exactly once.

The use of Eulerian walks was previously described as a solution for the assembly problem (Hutchinson, 1969; Gallant, 1983; Idury and Waterman, 1995; Pevzner et al., 2001). Eulerian walks can be found in polynomial time, but their application to high-throughput sequencing was not particularly successful for making practical assemblies (Flicek and Birney, 2009). Basically, the EULER approach makes a series of equivalent transformations that lead to a final set of paths (Pevzner et al., 2001). With this idea, even if a $(k+1)$-mer is present at two or more locations in the genome sequence, it is still possible to construct a single Eulerian walk, using edge multiplicities. However, to determine for each edge which multiplicity is suitable represents a difficult problem when the reads give an uneven coverage.

Many strategies (other than Eulerian walks) have been proposed to produce a good assembly with a de Bruijn graph. They are based on the idea of simplifying the de Bruijn graph. This simplification addresses the presence of topological errors in the graph and repetitive sequences.

We describe the existing algorithms in Section 4.3.1 and their respective problems in Section 4.3.2. Note that, in contrast, our proposed algorithm neither limits its search to Eulerian walks, nor performs such simplifications on the de Bruijn graph. Indeed, as shown in Section 6, we not only keep the de Bruijn graph as it is, but we even add annotations to preserve, as far as possible, all read information. Then, our approach searches for walks that are compatible with the given family of reads.

4.3.1. de Bruijn algorithms. The pioneering work in this field (Pevzner et al., 2001) proposed an algorithm that searches for Eulerian walks in a de Bruijn graph. Afterwards, this approach was adapted to short reads (Chaisson et al., 2004). With the advance of high-throughput sequencing, Chaisson et al. created the EULER-SR framework which deals with short reads and high-coverage data (Chaisson and Pevzner, 2008; Chaisson et al., 2008). Another de Bruijn assembler, called Velvet (Zerbino and Birney, 2008), was successful at assembling short reads. Velvet builds a de Bruijn graph, and transforms it in a simplified de Bruijn graph. In Velvet, the graph is also corrected by removing tips and by extracting sequence information from bubbles.

Furthermore, ABySS assembled the human genome with a parallel approach (Simpson et al., 2009). ALLPATHS builds a graph and finds all walks between each paired reads (Butler et al., 2008). Another approach for the assembly of very short reads (25 nt) was described by Hossain et al. (2009). In their work, the authors developed SHORTY, which is motivated by the concept of seeds. The latter are long (500 nt) high-quality sequences from which the assembly can stem. In our proposed algorithm, we also make use of the concept of seeds, but these seeds are computed by Ray with solely the short reads provided (in SHORTY, seeds are provided by the user). Finally, note that each of these algorithms only runs on a single technology. And only ABySS and Ray run in parallel and distribute data across computers with message passing interface.

4.3.2. Problems with current de Bruijn assemblers. To address the errors in the graph, a coverage cutoff can be defined to trim erroneous artifacts that are interspersed in the de Bruijn graph (Zerbino and Birney, 2008). Assembly protocols set the coverage cutoff to a particular value and erode anything in the graph whose coverage is lower than the cutoff. It is unlikely that all the vertices that have a depth of coverage lower than the cutoff correspond to concealed errors. Thus, such a cutoff will prohibit an assembler from yielding large contigs.

Existing assemblers are not designed for the assembly of mixes of reads from different systems, being only able to run on a single technology. Also, only ABySS and Ray calculate assemblies in parallel using message passing interface.

5. MIXING SEQUENCING TECHNOLOGIES

Using a mix of sequencing technologies is called hybrid assembly, and decreases the number of correlated errors in the reads (Diguistini et al., 2009). In principle, mixing sequencing technologies improves de novo assemblies. No solution has been designed to specifically and efficiently tackle this problem. Current hybrid-assembly procedures are not assembling simultaneously reads acquired from different systems (Goldberg et al., 2006; Aury et al., 2008; Diguistini et al., 2009). The asynchronous fabrication of contigs does not highlight the errors contributed by each single sequencing system. Moreover, not piecing together reads simultaneously can lead to loss of valuable information about short repeated regions (Chaisson and Pevzner, 2008). To address these shortcomings, Ray makes concurrent use of all available reads. In our experiments, namely in SpErSim (see Section 7), we observed that the assembly problem is simpler when the sequencing errors are based on pure random noise. Hence, we consider that mixing technologies is a good compromise to mimic as closely as possible randomly-occuring noise. The emergence of an array of sequencing technologies makes this compromise realistic.

5.1. Hybrid assembly methods

Goldberg et al. (2006) have assembled Sanger and Roche/454 reads together to produce high-quality draft assemblies. However, the lack of tools to perform simultaneous hybrid assembly was translated in the usage of more than one assembler. They assembled the Roche/454 reads with a first assembler, created Sanger-like reads from the Roche/454 contigs, and assembled the latter with Sanger reads with a second assembler.

While Roche/454 reads are sufficient to produce high-quality drafts, the latter contain small insertions/deletions, and mismatches that are directly caused by the technology used, and therefore are correlated among reads. To improve the rough quality of a draft, sequences from another technology can be aligned onto the draft assembly to find errors (Aury et al., 2008). This method is not a truly hybrid assembly because the Roche/454 and Illumina reads are not simultaneously assembled. Note also that in their paper, the authors suggested that beyond a particular coverage with short Illumina reads, the ability to correct errors present in a Roche/454 assembly can not be significantly improved, which is in contradiction with our experimental results. Indeed, with Ray, we experimentally demonstrated that it is possible to avoid almost all assembly errors by simultaneously assembling the Roche/454 and Illumina reads (see Section 7).

Diguistini et al. (2009) have gathered capillary (Sanger), Roche/454, and Illumina reads to decode the genome of a 32.5 Mb filamentous fungus. Although mixing sequencing technologies is strongly promoted by these authors, the underlying assembler, Forge, is unable to work directly with Illumina reads. The latter were preprocessed, using the Velvet program, to obtain sequences usable by Forge. This dataset is not considered herein because of the lack of a finished sequence.

6. THE ALGORITHM RAY

Ray is derived from the work by Pevzner et al. (2001) in the sense that we make use of a de Bruijn graph. However, our framework does not rely on Eulerian walks. To generate an assembly, we define specific subsequences, called seeds, and for each of them, the algorithm extends it into a contig. We define heuristics that control the extension process in such a way that the process is stopped if, at some point, the family of reads does not clearly indicate the direction of the extension. In example, consider the graph of Figure 1, and suppose that the seed is $z_2 \dots z_3$, then if most of the reads that overlap the seed also contain z_5, then we will proceed by adding z_5 to the contig. On the contrary, we will choose an another direction, and in the case where there is no obvious winner, the process will stop. Note that our heuristics will give a higher importance to reads that heavily overlap a contig than to reads that only intersect a small number of the last vertices of it. Indeed, we consider that reads that overlap strongly in the contig we are constructing are more informative to assign the direction in which the extension should proceed.

We measure the overlapping level with functions called offset_i and $\text{offset}_i^{\text{paired}}$. The proposed heuristics will therefore limit the length of the obtained contigs, but will give better guarantees against assembly errors. Moreover, those heuristics allow us to construct greedy algorithms in the sense that at each point, we extend the contig that we are constructing, or we end the construction. Such a greedy choice that will never be reconsidered is inevitable if one wants to have an algorithm that runs in polynomial time. The choice in favor of greedy optimization is motivated by the NP-hard nature of the problem (Pop, 2009).

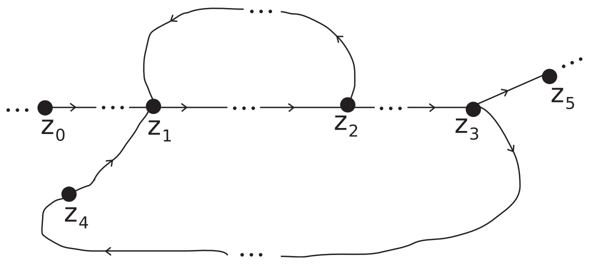

FIG. 1. A subgraph of a de Bruijn graph. This figure shows a part of a de Bruijn graph. In this example, short reads are not enough for the assembly problem. Suppose that the true genome sequence is of the form $\langle \cdots z_0 \cdots z_1 \cdots z_2 \cdots z_1 \cdots z_2 \cdots z_3 \cdots z_4 \cdots z_1 \cdots z_2 \cdots z_3 \cdots z_5 \cdots \rangle$. If the length of the reads (or paired reads) is smaller than the $z_1 \cdots z_2$ subsequence, no hints will help an assembly algorithm to differentiate the true sequence from the following one $\langle \cdots z_0 \cdots z_1 \cdots z_2 \cdots z_3 \cdots z_4 \cdots z_1 \cdots z_2 \cdots z_1 \cdots z_2 \cdots z_3 \cdots z_5 \cdots \rangle$. On the other hand, if there is a read that starts before z_1 and ends after z_2, there will be a possibility to solve this branching problem.

To minimize running time and memory utilization, we use the sequence data provided by the reads through a de Bruijn graph annotation, each read being annotated only once and attached to a single vertex of the graph.

6.1. *Coverage distribution*

Given a family of reads $D = \langle r_1, \ldots, r_t \rangle$, we denote by D^+ the family consisting of the reads of D and the reverse-complement sequences of D, or more precisely, $D^+ = \langle r_1, \ldots, r_t, \text{revcomp}(r_1), \ldots, \text{revcomp}(r_t) \rangle$. Moreover, for any integer c, a k-mer is c-confident if it occurs at least c times in the reads of D^+. Note that a k-mer that appears twice in the same read is counted twice. We define f_{D^+} as the function that associates to each integer value c the number of k-mers that are covered c times according to D^+. As proposed by Chaisson and Pevzner (2008), this function is the sum of an exponentially-decreasing curve and a Poisson distribution. The first represents the errors contained in the reads, and the second is a Poisson distribution because each read is a subsequence picked among all subsequences around a particular length with some almost uniform probability. The graph of the function f_{D^+} has the shape of the function drawn in Figure 2—the coverage distributions for the *A. baylyi* ADP1 dataset are shown in this figure, with Roche/454 reads, Illumina reads, and a mix of those. f_{D^+} has a local minimum, that we will call c_{min}, followed by a local maximum, that we will denote c_{peak}. Basically, c_{peak} is an estimate of the average coverage depth of the genome, and c_{min} is an estimation of the value where, with high probability, the amount of incorrect data is lower than the amount of correct data.

6.2. *Annotated de Bruijn graph*

Given a family of reads D, a de Bruijn parameter k, and a coverage cutoff c, the associated annotated de Bruijn graph, noted $Bruijn(D^+, k, c)$ is defined as follows. The vertices V of the graph are the c-confident k-mers and the set of arcs A is composed of all arcs between two vertices of V that correspond to a

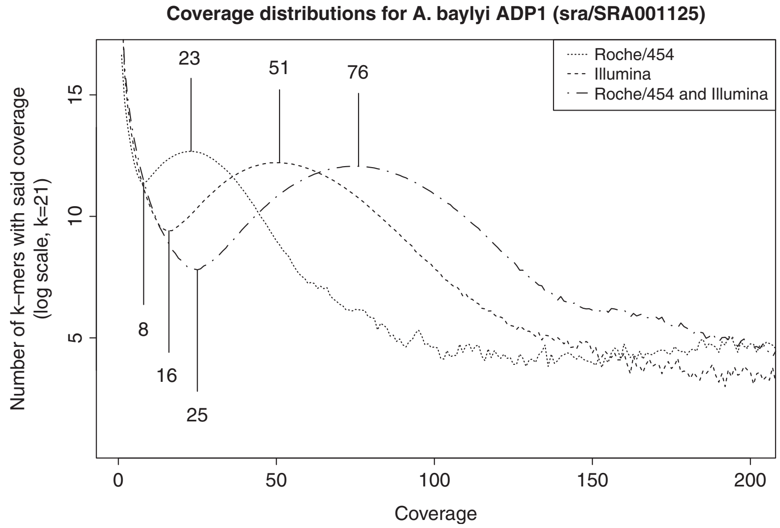

FIG. 2. Coverage distributions. This figure shows the coverage distributions of k-mers for the *A. baylyi* ADP1 dataset with Roche/454, Illumina, and Roche/454 and Illumina, $k = 21$. The minimum coverage and the peak coverage are identified for the Roche/454, Illumina, and Roche/454 and Illumina coverage distributions. The peak coverage of Roche/454+Illumina is greater than the sum of the peak coverage of Roche/454 and the peak coverage of Illumina, which suggests that the mixed approach allows one to recover low-coverage regions.

1-confident $(k+1)$-mer. For example, for $k=5$ and $c=2$, the two k-mers *ATGCT* and *TGCTG* will be vertices of the graph if and only if they both appear at least twice in the reads of D^+. Moreover, $\langle ATGCT, TGCTG \rangle$ will be an arc of the graph if and only if *ATGCTG* appears at least once in the reads of D^+.

For the annotation, we consider the first c-confident k-mer of each read r; we will denote this k-mer x_r. We then remove from D^+ every read r for which the associated x_r has a confidence value of 255 or more. A vertex with a confidence value of 255 or more is repeated because any region of a genome is usually covered 30–50 times. We consider that any x_r for which we have such coverage will belong to a repeated region. These regions are more susceptible to generate misassemblies. We avoid these regions, and do not consider the corresponding reads for the annotation. This choice implies that the algorithm may output contigs of shorter length, but we will gain on the quality of these contigs.

For the remaining x_r's, first note that x_r is a subsequence of the read r that is of length k and such that any subsequence of length k that starts before x_r in r is not c-confident. To each such remaining x_r, we complete the annotation (r, s), where s is the starting position of subsequence x_r in the read r. Hence, on a given vertex x, one can recover any read for which x is the first c-confident k-mer. Note that this annotation system is not memory intensive since each read is annotated only once. Moreover, the other positions of the read are recovered by the algorithm using the following functions.

6.2.1. Offset functions. Given a walk $w = \langle x_1 \cdots x_l \rangle$, and an arc $\langle x_l, y \rangle$, we define the i$^{\text{th}}$ offset value of y according to w, noted offset$_i(w, y)$, as the number of annotations (r, s) on the vertex x_i, such that y is a subsequence of r, and such that the distance between the starting point of x_i and the starting point of y are equal in both the read r and the walk w concatenated with the vertex y. The starting position of y in the read r is $l-i+s+1$.

As an example, put $k=5$ and $c=1$, and consider that we have a single read $r = ATGCATCG$ in D. Since $c=1$, all the 5-mers of r belong to the de Bruijn graph. Also, on the vertex $x_r = ATGCA$, we will have the annotation $(r, 1)$. Now consider the walk $w = \langle ATGCA, TGCAT, GCATC \rangle$, and the vertex $y = CATCG$. Then, we have that offset$_1(w, y) = 1$ because the read r starts at position 1 in w, and the distance between y and x_r is the same in the read and in the concatenation of the walk w with the new arc $\langle GCATC, CATCG \rangle$. Note that y is a subsequence of r, whose starting position is 4, which is exactly $l-i+s+1$, since the length of the walk w is $l=3$, the position x_r in w is $i=1$, and the position of x_r in the read r is $s=1$.

We will have the same type of functions for paired reads, except that since the fragment length is approximated by d_μ for each library, we will tolerate a certain difference between the length d of a particular pair of reads and the average length d_μ for the corresponding fragment library—namely d_σ. More formally, for a read pair $(r_1, r_2, d_\mu, d_\sigma)$, let us define the meta read r_{12} which has length d, begins with r_1, ends with revcomp(r_2), and is composed of dummy letters in the middle. Then, given a walk $w = \langle x_1 \cdots x_l \rangle$, and an arc $\langle x_l, y \rangle$, we define the i$^{\text{th}}$ paired offset value of y according to w, noted offset$_i^{\text{paired}}$ (w, y), as the number of annotations (r_1, s_1) on the vertex x_i, for which there exists a paired read $(r_1, r_2, d_\mu, d_\sigma)$, such that y is a subsequence of revcomp(r_2), and the distances between the starting point of x_i and the starting point of y in the meta read r_{12} and in the walk w concatenated with the vertex y differ by a value that is at most d_σ. Mathematically, the starting position of y in the read r_2 is $d_\mu - |r_2| - (l - i + s_1 + 1) \pm d_\sigma$.

As stated above, we want to give more importance to reads that have a large overlap with the contig we are constructing. Let us point out that when the value of i is getting smaller in the offset function, the magnitude of the overlap is increasing. So given a partially constructed contig $w = \langle x_1 \cdots, x_l \rangle$, and two possible arcs $\langle x_l, y \rangle$ and $\langle x_l, y' \rangle$ that we can add to it, we will compare the offset values of y and y' according to w, if there is a clear winner, the algorithm will choose it, otherwise it will stop the ongoing extension.

6.3. Seeds

The proposed algorithm starts on some walks, for which we have a strong confidence that they are subsequences of the genome sequence. Such particular walks are referred to as seeds. To obtain them, we first consider a de Bruijn graph with a very high cutoff value. We fix this cutoff to $\frac{c_{min} + c_{peak}}{2}$. c_{peak} corresponds to the average coverage and c_{min} to where the number of errors is lower than the number of true sequences. We consider that by taking the average of those two, a very small amount of errors will be in the de Bruijn graph $Bruijn(D^+, k, \frac{c_{min} + c_{peak}}{2})$. The resulting stringent de Bruijn graph suffers from a huge loss of the information contained in the reads, but this procedure is only used to determine the seeds. More specifically, the seeds are the set of maximum walks in *Bruijn*$(D^+, k, \frac{c_{min}+c_{peak}}{2})$ that are composed of vertices of indegree and outdegree at most one.

6.4. The heuristics

In this section, we present the heuristics on which our algorithm is based. In these rules, m is a multiplicative value indicating the confidence required to perform a choice—as the coverage increases, its value decreases. For a vertex, $m = 3.0$ when its coverage is at least 2, but no more than 19, $m = 2.0$ when its coverage is at least 20, but no more than 24, and $m = 1.3$ when its coverage is at least 25. For any vertex, the relation between its coverage and the value of the multiplicator m is not a parameter, and works well on various tested datasets (see Section 7). The values taken by m are motivated by the distribution of reads on a seeds—two reads starting consecutively on a seed will be nearer if more local coverage is available.

Given a walk $w = \langle x_1 \cdots, x_l \rangle$, and two arcs in the de Bruijn graph *Bruijn*(D^+, k, c) that start at x_l, namely $\langle x_l, y \rangle$ and $\langle x_l, y' \rangle$, we say that

Rule 1: y wins over y' if the three following inequalities hold.

$$\begin{aligned} m \cdot \sum_{i=1}^{l} (l-i) \cdot \text{offset}_i^{\text{paired}}(w, y') &< \sum_{i=1}^{l} (l-i) \cdot \text{offset}_i^{\text{paired}}(w, y) \\ m \cdot \sum_{i=1}^{l} \text{offset}_i^{\text{paired}}(w, y') &< \sum_{i=1}^{l} \text{offset}_i^{\text{paired}}(w, y) \\ m \cdot \min_{i \in \{1..l\}} (\text{offset}_i^{\text{paired}}(w, y') > 0) &< \min_{i \in \{1..l\}} (\text{offset}_i^{\text{paired}}(w, y) > 0) \end{aligned}$$

```
GrowSeed($w = \langle x_1, \ldots, x_l \rangle$)
  $s \leftarrow 0$
  WHILE $s <$ number of vertices in walk $w$ DO
    $s \leftarrow$ number of vertices in walk $w$
    Consider the arcs $\langle x_l, y_1 \rangle, \ldots, \langle x_l, y_m \rangle$
    IF one of the $y$'s wins against all the others according to Rule 1 THEN
      Extend the value of the walk $w$ by adding this particular arc
    ELSE IF one of the $y$'s wins against all the others according to Rule 2 THEN
      Extend the value of the walk $w$ by adding this particular arc
  RETURN extended $w$

Ray(D,k,c)
  Build Bruijn($D+, k, c$)
  FOR each seed $w = \langle x_1, \ldots, x_l \rangle$ DO
    IF the seed was not previously reached THEN
      $w' \leftarrow$GrowSeed($w$)
      $x \leftarrow$ the reverse-complement sequence of $w'$
      $x' \leftarrow$ GrowSeed($x$)
      store the walk $x'$
  Merge overlapping walks in a reverse-complement fashion
  RETURN the resulting contigs
```

FIG. 3. The Ray algorithm. Ray is a greedy algorithm on a de Bruijn graph. The extension of seeds is carried out by the subroutine GrowSeed. Each seed is extended using the set of Rules 1 and 2. Afterwards, each extended seed is extended in the opposite direction using the reverse-complement path of the extended seed. Given two seeds s_1 and s_2, the reachability of s_1 from s_2 is not a necessary and sufficient condition of the reachability of s_2 from s_1. Owing to this property of reachability between seeds, a final merging step is necessary to remove things appearing twice in the assembly.

Table 1. Dataset Descriptions

Dataset	*Description*	*Reads accession key*
Simulations: *S. pneumoniae* R6		
SpSim	Simulated 50-bp reads, depth of coverage: 50	—
SpErSim	SpSim with 1% mismatch	—
SpPairedSim	Simulated 200-bp fragments, 50-bp reads representing their extremities, depth of coverage: 50	—
Mixed dataset 1: *E. coli* K12 MG1655		
Roche/454	Roche/454 reads	sra/SRA001028
Illumina	Illumina paired reads	sra/SRA001125
Mixed	—	sra/SRA001125, sra/SRA001028
Mixed dataset 2: *A. baylyi* ADP1		
Roche/454	Roche/454 reads	sra/SRA003611
Illumina	Illumina unpaired reads	sra/SRA003611
Mixed	—	sra/SRA003611
Mixed dataset 3: *C. curtum* DSM 15641		
Roche/454	Roche/454 reads	sra/SRA008863
Illumina	Illumina unpaired reads	sra/SRA008863
Mixed	—	sra/SRA008863

So, according to Rule 1, y wins over y' if 1) the number of paired reads that strongly overlap y and w is more than m times the corresponding value for y', 2) the total amount of paired reads that overlap y and w is more than m times than the same amount for y', and 3) the paired read that has the maximal overlap over w and y is more than m times the same value for y'. Recall that the smallest is the value of i, the biggest is the overlap for the corresponding reads. **Rule 2:** By replacing the functions $\text{offset}_i^{\text{paired}}$ by their corresponding functions offset_i in the three equations above, we get Rule 2. Hence, Rule 2 is doing exactly the same type of measurement as Rule 1, except that it works on single reads. These rules are utilized to extend seeds in the Ray algorithm. The Ray algorithm is presented in Figure 3.

6.5. Technical points about the implementation

The data structure selected to extract k-mer information in reads is the splay tree (Sleator and Tarjan, 1985)—a balanced binary search tree based on the splay operation. k-mers are stored on 64 bits—thus, the maximum value for k is 32.

7. RESULTS AND DISCUSSION

Ray compares favorably with current algorithms. We emphasize that only Ray performs very well on mixed datasets, and that the parallel nature of Ray makes it scalable. The goal of this evaluation is to demonstrate that Ray helps to assemble genomes using high-throughput sequencing. Comparisons with available assemblers are in Tables S1–S5.

7.1. Datasets

Datasets are presented in Table 1. For our datasets, we selected living organisms for which high-quality data were available—these included *Streptococcus pneumoniae*, *Escherichia coli*, *Acinetobacter baylyi*, and *Cryptobacterium curtum*. *Streptococcus pneumoniae* is a significant cause of human diseases, and the avirulent strain R6 (NCBI accession number: nuccore/NC_003098) is a platform for investigation (Hoskins et al., 2001). *Escherichia coli* K-12 (NCBI accession number: nuccore/NC_000913) is a model organism (Blattner et al., 1997). We gathered Roche/454 reads (Miller et al., 2008) and Illumina reads (produced by

Table 2. Assemblers

Assembler	*Version*
ABySS (Simpson et al., 2009)	1.1.2
EULER-SR (Chaisson et al., 2008)	1.1.2
Newbler (Margulies et al., 2005)	2.0.00.20
Ray	0.0.7
Velvet (Zerbino and Birney, 2008)	0.7.61

Illumina inc.) for *E. coli* K-12 strain MG1655. *Acinetobacter baylyi* ADP1 (NCBI accession number: nuccore/NC_005966) is highly competent for natural transformation, making it very convenient for genetic engineering (Barbe et al., 2004). We downloaded Roche/454 and Illumina reads for *A. baylyi*ADP1 to test Ray (Aury et al., 2008). *Cryptobacterium curtum* (NCBI accession: nuccore/NC_013170) is an opportunistic pathogen, and its genome was assembled with Roche/454 reads and Sanger reads (Mavromatis et al, 2009).

We simulated two error-free datasets from *Streptococcus pneumoniae* R6 (SpSim and SpPairedSim). We also simulated one error-prone dataset from *Streptococcus pneumoniae* R6 (SpErSim). As pointed out by Hossain et al. (2009), it is striking how the presence of errors in real-world sequencing data is underestimated by the common use of simulated data. To test this, we tested Ray on real datasets. We downloaded datasets from the Short Read Archive (SRA; available at www.ncbi.nlm.nih.gov/sra). These sequence reads are from *Escherichia coli* K-12 MG1655 (Miller et al., 2008), *Acinetobacter baylyi* ADP1 (Aury et al., 2008), and Cryptobacterium curtum DSM 15641 (Mavromatis et al., 2009). For each of these genomes, the corresponding dataset contained Illumina reads, and Roche/454 reads. Table S1 describes datasets on which comparisons presented in Tables S2–S5 were performed.

7.2. Quality assessment metrics

To assess the quality of assemblies and compare our method with other approaches, we selected a set of metrics usually utilized when comparing sequence quality. These metrics are described in the definition of the AP (see Definition 1) and characterize a fine assembly. They are the number of contigs having at least 500 letters (or bp), the number of bases (bp), the mean size of contigs (bp), the N50 (the length of the largest contig such that all contigs with at least its length contains at least 50% of the assembly, in bp), the largest contig size (bp), the genome coverage (breadth of coverage—percentage of bases in the genome covered), incorrect contigs, mismatches, and indels. The number of contigs is a measure of the fragmentation of the assembly. A higher number of assembly bases indicates that the assembly contains two strands

Table 3. Assemblies of Simulated Error-Free and Error-Prone Datasets

Assembler	*Contig ≥500 bp*	*Bases (bp)*	*Mean size (bp)*	*N50 (bp)*	*Largest contig (bp)*	*Genome coverage (%)*	*Incorrect contigs*	*Mismatches*	*Indels*	*Running time*
SpSim										
ABySS	417	1898819	4553	7349	27222	0.9343	0	4	0	1m56.066s
EULER-SR	261	1967594	7538	11621	61396	0.9419	6	68	123	7m22.779s
Velvet	280	1917129	6846	11279	44362	0.9437	1	23	8	2m15.931s
Ray	259	1954999	7548	11561	77867	0.9608	0	0	0	3m25.240s
SpErSim										
ABySS	418	1898547	4541	7349	27222	0.9342	0	4	0	4m52.727s
EULER-SR	267	1965104	7359	11477	61349	0.9413	6	79	237	11m15.383s
Velvet	290	1913682	6598	10302	42572	0.9423	2	27	11	2m40.792s
Ray	259	1939235	7487	11554	77853	0.9531	0	0	0	4m29.223s
SpPairedSim										
ABySS	151	2019778	13376	22045	104182	0.9815	0	213	9	3m38.944s
EULER-SR	235	1976831	8412	12383	61593	0.9458	13	69	187	9m59.464s
Velvet	113	1950222	17258	32111	123292	0.9565	30	382	140	2m15.371s
Ray	96	1964569	20464	36692	127906	0.9632	0	1	0	5m52.834s

Table 4. Assemblies of Mixed Readouts

Data	*Contig ≥500 bp*	*Bases (bp)*	*Mean size (bp)*	*N50 (bp)*	*Largest contig (bp)*	*Genome coverage (%)*	*Incorrect contigs*	*Mismatches*	*Indels*	*Running time*
Mixed dataset 1: *E. coli* K-12 MG1655										
Illumina	126	4591168	36437	72499	174569	0.9818	0	2	4	47m54.377s
Roche/454	874	4513335	5163	8771	42344	0.9731	9	64	247	29m53.841s
Mixed	109	4579657	42015	87318	268385	0.9831	1	234	6	62m30.978s
Mixed dataset 2: *A. baylyi* ADP1										
Illumina	259	3677696	14199	25852	72730	0.9749	0	82	6	29m48.993s
Roche/454	109	3547847	32549	61793	214173	0.9846	0	69	380	43m3.785s
Mixed	91	3540404	38905	82891	215819	0.9804	1	7	1	36m27.635s
Mixed dataset 3: *C. curtum* DSM 15641										
Illumina	72	1606647	22314	36518	91303	0.9862	0	1	1	19m51.388s
Roche/454	30	1609423	53647	261125	477358	0.9904	0	0	8	21m24.064s
Mixed	27	1602133	59338	116274	236544	0.9897	0	0	1	35m8.569s

Roche/454 reads were assembled with Newbler, whereas Illumina and mixed data were assembled with Ray.

for particular genomic regions, or that the DNA molecules were contaminated. The distribution of contig lengths is evaluated with the mean contig length, N50, and largest contig length. The genome coverage is crucial in evaluating the fraction of the genome being covered by the assembly. The number of contigs, mean size, N50, largest contig length, and the breadth of coverage are standard indicators, but they miss the assembly errors. The sensitivity of an assembly software towards repetitive elements—the main source of misassemblies—is evaluated with the number of incorrect contigs. Mismatches, small insertions, and small deletions are local errors. Lowering the number of these error types is paramount to the analysis of genome variation. The number of incorrect contigs, the number of mismatches, and the number of indels allow critical assessment of genome assemblies. Assembly contigs were aligned to a reference, and these alignments were evaluated using the metrics described above. Alignments of sequences were done with Mummer (Kurtz et al., 2004) and Blast (Altschul et al., 1997).

7.3. Genome assemblers

Numerous assemblers are available. We selected the most recent assemblers in our benchmarks. We compared Ray with Velvet (Zerbino and Birney, 2008), EULER-SR (Chaisson et al., 2008), ABySS (Simpson et al., 2009), and Newbler (Margulies et al., 2005). We refer the reader to Table 2 for assembler descriptions. ABySS was run according to the documentation provided. EULER-SR was run with $k=21$. A rule file was provided when paired information was available. Newbler was run with default parameters using the command-line interface. Ray was run with the sequence files as input, and we fixed the parameters to the same value on all datasets, namely we fixed $c=2$ and $k=21$. The values of c_{min} and c_{peak} were calculated automatically (see Section 6.1), and so were the average fragment length and standard deviation for paired libraries. Ray is an assembler implementation using auto-calculated parameters (c_{min}, c_{peak}, and d_μ and d_σ for each paired library) which renders the process readily applicable. Velvet was run with hash size 21, and the expected and cutoff coverages were determined according to the manual.

7.4. Simulations

Results on simulated data are shown in Tables 3 and S2. We chose to include simulated data because in the case of real data, the reference is not always error-free. Three datasets are presented. SpSim is an error-free dataset with 50-bp reads. SpErSim contains the same reads as SpSim, in addition of random mismatches. SpPairedSim is similar to SpSim, but includes paired information.

SpSim (simulated). The SpSim dataset contained simulated short 50-base reads at 50× (depth of coverage) from the 2038615-base *Streptococcus pneumoniae* R6 genome. The peak coverage c_{peak} in the coverage distribution had a value of 29, and the c_{min} was not relevant in this dataset (thanks to the absence of errors). All five assemblers, except Newbler, were able to analyze short reads (Tables 3 and S2). In Newbler, overlaps were detected using a minimum overlap length, and this minimum overlap length was longer than the read length. The assemblies were similar, but Ray was slightly better for this dataset. Indeed, Ray produced 259 contigs, and its assembly covered 96% of the genome. Both ABySS and Ray produced no incorrect contigs. EULER-SR produced more mismatches than the other assemblers and yielded indels on this error-free data set. Ray had the best overall performance on this dataset.

SpErSim (simulated, 1% random mismatch). The results for this dataset were similar to those obtained with the dataset above. From these results, we concluded that assemblers are not very sensitive to random noise. The addition of errors, however, increases the amount of consumed memory, owing to the additional *k*-mers occuring in the reads, but not in the genome. These errors are instanciated as bubbles and tips in the graph.

SpPairedSim (simulated). This dataset included short 50-base reads at 50× from *S. pneumoniae* R6, and these reads were generated in a paired-end fashion. The paired reads are the extremities of 200-nt fragments. Only Newbler was unable to assemble this dataset, for the same reason as in the dataset SpSim. Ray produced no incorrect contigs, only 1 mismatch, and 0 indel (Tables 3 and S2). Even if the number of contigs is very low, the number of mismatches and the number of indels must also be minimized to facilitate downstream analyses. It is very important that the assembly contains the lowest number of errors because otherwise mismatches and indels will be interpreted as mutations by biologists. Given the null number of chimeric contigs for Ray, and the large number of mismatches for other assemblers, Ray outperformed the other assemblers for this dataset.

7.5. *Real mixed datasets*

Real datasets are described in Tables 1 and S1. Table 4 presents the added value provided by mixing reads, for three different genomes: *E. coli* K-12 MG1655, *A. baylyi* ADP1, and *C. curtum* DSM 15641. Comparisons with other assemblers are presented in Table S3 (*E. coli* K-12 MG1655), Table S4 (*A. baylyi* ADP1), and Table S5 (*C. curtum* DSM 15641).

E. coli. For *E. coli* K-12 MG1655, Newbler gave 874 contigs whereas Ray produced 126 with only Illumina reads, and 109 with both technologies (Roche/454 and Illumina) with 1 misassembled contig that aligned in two blocks: 1–83124 with 4210996–4294119, and 82863–112667 with 4294197–4323999. This contig was presumably correct according to the colinearity between aligned segments. For this genome, the Illumina reads were grouped in two paired libraries, d_μ and d_σ had values of 215 and 11 for the first library and 486 and 26 for the second.

A. baylyi. For *A. baylyi* ADP1, Newbler produced 109 contigs with 380 indels. Ray assembled the Illumina reads in 259 contigs and the 454/Roche and Illumina reads in 91 contigs, which is less than 109, the number for Newbler with Roche/454 reads only. Mixed reads allowed Ray to detect homopolymer errors (indels): Newbler produced 380 and Ray only 1. For the *A. baylyi* ADP1 mixed data, the misassembled contig aligned in two blocks: 1358–560 with 2804111–2804909, and 234–1 with 2804911–2805144—a single indel in the contig.

C. curtum. For *C. curtum* DSM 15641, Newbler produced 30 contigs with 8 indels, and Ray assembled data in 76 contigs (Illumina reads) and in 27 contigs (Roche/454 and Illumina reads). Ray (27 contigs) outperformed Newbler (30 contigs), and the other metrics were constant, except for the largest contig length, for which Newbler was better.

Accordingly, for these reported results, Ray is an excellent assembler—and the only one to assemble successfully mixed datasets (Tables S3–S5). Also, Ray is better than Velvet and ABySS on Illumina data.

Furthermore, the assembly of *E. coli* K-12 MG1655 with paired short sequences confirmed that read length does not matter for de novo assembly—a pair of reads can be transformed explicitely in a longer sequence of a length *d* (*d* being near the d_μ of its corresponding paired library) like in EULER-SR (Chaisson et al., 2008), or implicitely, like in Ray.

7.6. *Scalability*

Ray assembles reads in an efficient way. In Tables 3, 4, and S2–S5, the running time indicates that Ray ranked well in speed evaluation. Ray is scalable because it uses messages passing, like ABySS. Scalability will be instrumental, if not paramount, to tackle not only larger genomes, but also to sort out the tremendous amount of data and weed out the errors coming from the cutting-edge sequencers, such as the Illumina HiSeq 2000—which has a reported output of 150–200 Gb. To the best of our knowledge, only

ABySS and Ray make extensive use of message passing interface, allowing them to run on many computers at once.

8. CONCLUSION

The throughput of new sequencing instruments is overwhelming assemblers. Thus, better bioinformatics systems are necessary and critical in sequence assembly. The cost of DNA sequencing has dramatically lowered over the last years while the sheer volume of data generated by sequencing instruments has grown exponentially. The lower cost makes genome research more affordable whereas the generated volume of data increases the complexity of the analyses. To assist in these analyses, numerous assembly algorithms have been published over the years. However, none of these can tackle simultaneously a mix of reads from different sequencing systems. The ability to assemble simultaneously such mixes of reads is useful to alleviate the error rates of each single sequencing platform. Ray allows this type of analysis. In principle, genomes obtained with Ray will require less sequence finishing.

Our contributions are an open source parallel assembler, the best results on simulated datasets (Tables 3 and S2), improved assemblies on Illumina reads in comparison with current state-of-the-art short-read assemblers (Tables 3 and S3–S5), and the first assembler that can simultaneously assemble reads from a mix of sequencing technologies (Tables 4 and S3–S5).

Future directions are to assemble a human genome with Ray, and to adapt the code for the new upcoming sequencing platforms. With new sequencing technologies emerging, an assembler that can handle all of them in an efficient fashion is highly desirable.

ACKNOWLEDGMENTS

We are grateful to Mohak Shah, Frédéric Raymond, Ken Dewar, Élénie Godzaridis, and Éric Paquet for critical reading of the manuscript. We are greatly indebted to Dr. Torsten Seemann for providing precious feedback. We thank the Canadian Institutes of Health Research (J.C.) and the Natural Sciences and Engineering Research Council of Canada (NSERC discovery grant 262067 to F.L.) for research funding. J.C. holds the Canada Research Chair in Medical Genomics. S.B. is recipient of a Master's award (200902CGM-204212-172830) and a Doctoral award (200910GSD-226209-172830) from the Canadian Institutes of Health Research. We also wish to acknowledge the CLUMEQ consortium and Compute Canada for access to computing infrastructure (colosse.clumeq.ca), and the Canadian Foundation for Innovation for infrastructure funding.

DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Altschul, S., et al. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25, 3389–3402.

Aury, J.-M., et al. 2008. High quality draft sequences for prokaryotic genomes using a mix of new sequencing technologies. *BMC Genomics* 9, 603.

Barbe, V., et al. 2004. Unique features revealed by the genome sequence of Acinetobacter sp. ADP1, a versatile and naturally transformation competent bacterium. *Nucleic Acids Res.* 32, 5766–5779.

Batzoglou, S., et al. 2002. Arachne: a whole-genome shotgun assembler. *Genome Res.* 12, 177–189.

Bentley, D.R., et al. 2008. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* 456, 53–59.

Blattner, F.R., et al. 1997. The complete genome sequence of *Escherichia coli* K-12. *Science* 277, 1453–1462.

Butler, J., et al. 2008. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Res.* 18, 810–820.

Chaisson, M., et al. 2004. Fragment assembly with short reads. *Bioinformatics* 20, 2067–2074.

Chaisson, M.J., et al. 2008. De novo fragment assembly with short mate-paired reads: does the read length matter? *Genome Res.* 19, 336–346.

Chaisson, M.J., et al. 2008. Short read fragment assembly of bacterial genomes. *Genome Res.* 18, 324–330.

Diguistini, S., et al. 2009. De novo genome sequence assembly of a filamentous fungus using Sanger, 454 and Illumina sequence data. *Genome Biol.* 10, R94.

Dohm, J.C., et al. 2007. SHARGCS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Res.* 17, 1697–1706.

Flicek, P., and Birney, E. 2009. Sense from sequence reads: methods for alignment and assembly. *Nat. Methods* 6, S6–S12.

Gallant, J.K. 1983. The complexity of the overlap method for sequencing biopolymers. *J. Theor. Biol.* 101, 1–17.

Goldberg, S.M.D., et al. 2006. A Sanger/pyrosequencing hybrid approach for the generation of high-quality draft assemblies of marine microbial genomes. *Proc. Natl. Acad. Sci. USA* 103, 11240–11245.

Hernandez, D., et al. 2008. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome Res.* 18, 802–809.

Holt, R.A., and Jones, S.J.M. 2008. The new paradigm of flow cell sequencing. *Genome Res.* 18, 839–846.

Hoskins, J., et al. 2001. Genome of the bacterium *Streptococcus pneumoniae* strain R6. *J. Bacteriol.* 183, 5709–5717.

Hossain, M.S., et al. 2009. Crystallizing short-read assemblies around seeds. *BMC Bioinform.* 10, S16.

Hutchinson, G. 1969. Evaluation of polymer sequence fragment data using graph theory. *Bull. Math. Biophys.* 31, 541–562.

Idury, R.M., and Waterman, M.S. 1995. A new algorithm for DNA sequence assembly. *J. Comput. Biol.* 2, 291–306.

Jeck, W.R., et al. 2007. Extending assembly of short DNA sequences to handle error. *Bioinformatics* 23, 2942–2944.

Kurtz, S., et al. 2004. Versatile and open software for comparing large genomes. *Genome Biol.* 5, R12.

Margulies, M., et al. 2005. Genome sequencing in microfabricated high-density picolitre reactors. *Nature* 437, 376–380.

Mavromatis, K., et al. 2009. Complete genome sequence of *Cryptobacterium curtum* type strain (12-3T). *Stand. Genomic Sci.* 1, 96–100.

Medini, D., et al. 2008. Microbiology in the post-genomic era. *Nat. Rev. Microbiol.* 6, 419–430.

Miller, J.R., et al. 2008. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics* 24, 2818–2824.

Myers, E.W., et al. 2000. A whole-genome assembly of Drosophila. *Science* 287, 2196–2204.

Pevzner, P.A., et al. 2001. An Eulerian walk approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. USA* 98, 9748–9753.

Pop, M. 2009. Genome assembly reborn: recent computational challenges. *Brief Bioinform.* 10, 354–366.

Rothberg, J.M., and Leamon, J.H. 2008. The development and impact of 454 sequencing. *Nat. Biotechnol.* 26, 1117–1124.

Sanger, F., et al. 1977. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. USA* 74, 5463–5467.

Scheibye-Alsing, K., et al. 2009. Sequence assembly. *Comput. Biol. Chem.* 33, 121–136.

Simpson, J.T., et al. 2009. ABySS: a parallel assembler for short read sequence data. *Genome Res.* 19, 1117–1123.

Sleator, D.D., and Tarjan, R.E. 1985. Self-adjusting binary search trees. *J. ACM* 32, 652–686.

Smith, L.M., et al. 1986. Fluorescence detection in automated DNA sequence analysis. *Nature* 321, 674–679.

Warren, R.L., et al. 2007. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics* 23, 500–501.

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

Address correspondence to:
Mr. Sébastien Boisvert
Département de Médecine Moléculaire
Université Laval
1065, avenue de la Médecine
Québec (Québec), Canada, G1V 0A6

E-mail: sebastien.boisvert.3@ulaval.ca