Genome analysis

Genome-scale de novo assembly using ALGA

Sylwester Swat¹, Artur Laskowski¹, Jan Badura¹, Wojciech Frohmberg¹, Paweł Wojciechowski¹,², Aleksandra Świercz¹,², Marta Kasprzak¹,*, Jacek Blazewicz¹,²

¹Poznan University of Technology, Institute of Computing Science, Piotrowo 2, 60-965 Poznan, Poland
²Institute of Bioorganic Chemistry, Polish Academy of Sciences, Noskowskiego 12/14, 61-704 Poznan, Poland

* To whom correspondence should be addressed

Associate Editor: XXXXXXX
Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: There are very few methods for de novo genome assembly based on the overlap graph approach. It is considered as giving more exact results than the so-called de Bruijn graph approach but in much greater time and of much higher memory usage. It is not uncommon that assembly methods involving the overlap graph model are not able to successfully compute greater data sets, mainly due to memory limitation of a computer. This was the reason for developing in last decades mainly de Bruijn-based assembly methods, fast and fairly accurate. However, the latter methods can fail for longer or more repetitive genomes, as they decompose reads to shorter fragments and lose a part of information. An efficient assembler for processing big data sets and using the overlap graph model is still looked out.

Results: We propose a new genome-scale de novo assembler based on the overlap graph approach, designed for short-read sequencing data. The method, ALGA, incorporates several new ideas resulting in more exact contigs produced in short time. Among these ideas we have creation of a sparse but quite informative graph, reduction of the graph including a procedure referring to the problem of minimum spanning tree of a local subgraph, and graph traversal connected with simultaneous analysis of contigs stored so far. What is rare in genome assembly, the algorithm is almost parameter-free, with only one optional parameter to be set by a user. ALGA was compared with nine state-of-the-art assemblers in tests on genome-scale sequencing data obtained from real experiments on six organisms, differing in size, coverage, GC content, and repetition rate. ALGA produced best results in the sense of overall quality of genome reconstruction, understood as a good balance between genome coverage, accuracy, and length of resulting sequences. The algorithm is one of tools involved in processing data in currently realized national project Genomic Map of Poland.

Availability: ALGA is available at http://alga.put.poznan.pl.
Contact: marta.kasprzak@cs.put.poznan.pl

Supplementary information: Supplementary material is available at Bioinformatics online.

1 Introduction

Advances in next-generation sequencing (NGS) technologies, which reduce time and cost of producing billions of short sequences in a single run, led to rapid increase of sequencing data. The first NGS technology, 454 sequencing, has been currently superseded by a newer NGS platform, Illumina, and single-molecule sequencing (SMRT) technologies for producing long reads (Ameur et al., 2019). NGS still gives reads of much higher quality than SMS and, together with high accessibility of genomic data, results in many different applications: creation of population studies of genetic variation (Sive, 2015; 1001 Genomes Consortium, 2016), DNA methylation (Kawakatsu et al., 2016), and metagenomic studies (Pereira-Marques et al., 2019). Many applications have also been developed in the context of bioforensics, biosurveillance, and infectious disease diagnostics, see, e.g., the recent survey (Minogue et al., 2019).

However, many organisms still do not have their genomes recognized, and those that already have a reference genome have gaps in it due to high repetition rate or limitation of sequencing technologies. Thus, a lot of effort is put into de novo genome reconstruction (de novo sequence assembly), which bases only on the information about reads reported by a sequencer. Matching the reads together, nowadays hundreds of millions at once, is done by finding pairs of overlapping reads. Such a reconstruction of a genome is characterized by high demand on computational resources and is realized on the basis of assembly graphs: overlap/string graphs

© The Author(s) 2021. Published by Oxford University Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/4.0/), which permits non-commercial re-use, distribution, and reproduction in any medium, provided the original work is properly cited. For commercial re-use, please contact journals.permissions@oup.com
or de Bruijn graphs. Paths in such graphs represent larger fragments of a chromosome.

The computational model of overlap graphs was historically first among those related to DNA sequencing problems. Its origin dates back to 1988, when Lysow and co-authors proposed a graph model for the problem of DNA sequencing by hybridization (Lysow et al., 1988). There, every oligonucleotide was represented by a vertex of the graph, and directed edges corresponded to possible overlaps of the oligonucleotides. For years, the model was adjusted to new requirements; oligonucleotides were replaced by reads produced by sequencers, in overlaps one had to take into account inexact matches due to sequencing errors, and the process of sequence reconstruction moved to the higher level of DNA sequence assembly. However, the basic graph remained more or less the same.

The overlap graph model is a straightforward conceptualization of the real-world process and works well as long as the sequencing data are not too large. The necessity of representing whole sequences in the graph and calculating inexact sequence alignments for most of pairs of the sequences makes the computational process very time and memory consuming (Blazewicz et al., 2018). The literature reports cases where assemblers from this group did not finish computations for greater data sets because of excessive memory requirements (Gonnella and Kurtz, 2012; Kajtazi et al., 2014). Long processing time of such assemblers is also often reported in the literature (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Subsequent propositions in this research area were aimed at shortening the time and reducing memory usage, usually at the cost of completeness of information kept (Kececioglu and Myers, 1995; Myers et al., 2000; Simpson and Durbin, 2012; Gonnella and Kurtz, 2012; Iie et al., 2014; Swietcz et al., 2018; some of them refer to a similar concept of string graphs). Among the most significant improvements, one may point the resignation from the dynamic programming algorithm in favor of heuristic methods calculating the sequence alignment or alignment-free sequence comparison, a strict selection of pairs of sequences undergoing the comparison procedure, or reduction of the resulting graph.

In the meantime, the main research thread in DNA sequence assembly directed to the so-called de Bruijn graph model. It was proposed in 1989 by Pevzner for solving sequencing by hybridization (Pevzner, 1989) and later applied to de novo sequence assembly (Ihrig and Waterman, 1995; Pevzner et al., 2001; Zerbino and Birney, 2008; Luo et al., 2012; Bankevich et al., 2012; Kajtazi et al., 2014). For the latter purpose, reads are decomposed into a series of overlapping subsequences (k-mers, where k stands for their length), each one represented by a directed edge in the graph, and vertices represent their prefixes and suffixes of length k − 1. In such a graph, only exact matches of sequences kept in vertices are assumed, sequencing errors are handled by allowing a k-mer to occur in different versions. A gain in efficiency of computations is achieved by a much lower volume of stored information and a smaller traversed graph, but mainly by discarding inexact matches. On the other hand, quality of resulting contigs diminishes a bit due to the sequence decomposition to k-mers, as the information about whole reads is partially lost (Blazewicz et al., 2018).

Taking into account all the practical constraints, de novo reconstruction of a DNA sequence in genome scale from NGS reads was done in most cases via an assembler based on the de Bruijn graph approach. Assemblers representing the overlap graph model often worked too long; moreover, they might not finish their computations because of memory limitation. Nowadays, however, the latter approach comes back to practice. There is a strong recommendation to use this graph model when reads are long, as the information lost following the decomposition in the de Bruijn graph approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.

It is worth noting that ALGA can be used without setting any parameter by a user. The parameters of ALGA are adjusted internally by the algorithm itself (see, for example, supplementary material for Assemblathon 2, Bradnam et al., 2013). Our new approach is then more problematic. But even for shorter reads, there is a need to obtain more accurate results. Our new proposition for short-read data, assembler ALGA (ALgornithm for Genome Assembly), is to exploit such a potential of the overlap graph approach with reasonable time and memory usage. We realize this, among others, by switching to bit representation of data and substitution-based error model, using hash values instead of strings in comparing reads, removing transitive connections in advance, and solving iteratively the minimum directed spanning tree problem as a part of graph reduction.
as aligning strings with indels is very costly from the computational point of view. By switching to the bit representation of nucleotide sequences and the error model restricted to substitutions, we gain a lot regarding time at every stage of computations.

### 2.3 Graph construction

As ALGA was intended to solve genome-scale data sets with acceptable memory usage, possibly on personal computers, the overlap graph had to be created as sparse as possible. At the same time, we wanted to keep the whole valuable information, thus to reject only edges duplicating some connections. At the current stage of the algorithm, we restrict our attention to transitive edges.

In the graph, every read outgoing the preprocessing step constitutes a vertex, as well as its reverse complementary counterpart. Reads of different lengths are allowed. Two vertices are connected by a directed edge if they have a prefix-suffix connection acceptable due to two parameters: minimum overlap area, automatically set to 55% of the average read length, and error rate of the overlap, the parameter defined by a user (optional, set to 0 by default). For each pair of reads, one best connection is stored, which is the one representing the greatest possible overlap, and the edge is weighted by the offset of the overlap (i.e., the distance between beginnings of the reads in the overlap). First, errorless overlaps are collected. We start with the shortest possible overlap area and systematically increase it by one after checking all pairs of reads for exact overlaps of the current length. Prefixes and suffixes of reads are processed as hash values instead of strings. This enables us to extend each prefix and suffix in time \(O(1)\) using a rolling hash function. In order to find, for each read \(a\), all reads \(b\) with a given property, we keep all suffixes on a custom-made hash map. This way, each valid pair \((a, b)\) is found in \(O(1)\) time, and the algorithm is able to construct the graph in time \(O(n \times m)\), where \(n\) stands for the number of reads and \(m\) for the average read length, which is the optimal bound.

If a graph, if constructed without further constraints, could be extremely large. This is due to the presence of a huge number of transitive edges. They can surpass irreducible edges by a factor of \(O(m)\) or greater if the genome contains a large number of identical, repeated regions. A connection of two overlapping reads \(a\) and \(c\) resulting in a longer sequence \(a\) is expressed by a transitive edge in the graph, if there is another read \(b\) that can be inserted, with proper overlaps, in between \(a\) and \(c\) with the same sequence \(a\) as a result. These connections are present in the overlap graph as three edges \((a, c)\), \((a, b)\), and \((b, c)\). The reduction of transitive edges is made as soon as edges being a better choice are found, immediately after finding the second edge of a proper triplet, see Fig. 1; this, however, comes at a cost of some additional computation. Such an action is supported by the implementation we chose, i.e., the systematic increase of the overlap area with the use of the rolling hash function. This way we avoid a peak in memory usage, observed in other algorithms with this, however, comes at a cost of some additional computation. Such an action is supported by the implementation we chose, i.e., the systematic increase of the overlap area with the use of the rolling hash function. This way we avoid a peak in memory usage, observed in other algorithms with such a reduction of transitive edges, applied after collecting in a graph all edges of a triplet.

After that, we admit overlaps with errors (if set up by a user). It is assumed that erroneous positions cannot be located within first or last three nucleotides of reads, as a difference in such regions usually means growing distance and an incorrect path. Connections from vertices with outdegree 0 to ones with indegree 0 are searched for with a method based on a \(k\)-mer comparison of reads, where \(k\) is set to 35. We divide each read into six equal (roughly) segments, select one least lexicographically \(k\)-mer per segment (it can project beyond a segment), and efficiently check via dynamic programming all pairs of candidates containing at least one common \(k\)-mer. In order to make this procedure alphabet-independent, we execute it four times with different “lexicographical” orders assumed: \((A, C, G, T), (C, G, T, A), (G, T, A, C),\) and \((T, A, C, G)\). Edges representing those pairs that overlap in such a way that user’s parameters are satisfied are added to the graph.

### 2.4 Graph simplification

In order to create contigs, the overlap graph built in the previous stage needs to undergo a few simplification steps that transform it to the state where every single edge represents a candidate for a final sequence.

#### 2.4.1 Neighborhood reduction

Some vertices in the graph have more numerous neighborhood (in the sense of outgoing edges) than the others. As long as their neighbors fit them tightly, all the connections are important. However, loose connections accompanying the tight ones can worsen quality of contigs. In ALGA, we treat a connection as tight if it represents the overlap area of length at least 1.4 of the minimum overlap area. All such edges are kept, and if there are at least three such outgoing edges at a vertex, other edges representing loose connections from this vertex are removed. Otherwise, we keep at the vertex some outgoing edges representing loose connections, to the total number of three edges leaving the vertex (if possible), and the ones representing the greatest overlap area are selected.

#### 2.4.2 Triangle inequality rule

Another simplification step is based on the triangle inequality. It is for further reduction of transitive edges, this time identified in a broader range. During the graph construction, these transitive edges which exactly covered other existing connections have been discarded. Now, we accept for reduction all edges violating the triangle inequality, i.e., of a weight not lower than the sum of weights of its constituent edges: \((a, c)\) of weight \(w\) is removed if there are edges \((a, b)\) and \((b, c)\) of weights \(w_1\) and \(w_2\) such that \(w \geq w_1 + w_2\). Currently, matching of reads is not examined.

#### 2.4.3 Short parallel paths

Transitive edges together with their alternative edges are small examples of more general structures, parallel paths. If such paths are short, they usually represent more or less the same sequence of nucleotides as their counterparts or one of them is a shortcut of the other. Removal of short
Swat et al.

2.4.4 Dead-end branches

Execution of the procedure for removing short parallel paths leaves in the graph a lot of short dead-end branches (as, for example, edge (a, i) in Fig. 2C), also called tips. We remove them all by setting the maximal length of the dead ends to be removed to value D, the same as in the previous procedure. At every fork, if all outgoing (incoming) paths are not longer than D, the longest one is treated as the main path and retained.

2.4.5 Path compression

Directed paths between forks are compressed to single edges according to the following conditions. Each path \((a_0, a_1, \ldots, a_p)\), where \(d^{-1}(a_0) \neq 1 \text{ or } d^{-1}(a_0) > 1 \text{ and } d^{-1}(a_p) > 1 \text{ or } d^{-1}(a_p) \neq 1\) and \(d^{-1}(a_i) = d^{-1}(a_i) = 1, 0 < i < p\), is replaced by edge \((a_0, a_p)\). It can be assumed without loss on contiguity by a vertex, respectively. Weights of the compressed edges are set to weights of the corresponding paths (being sums of weights of their constituent edges). Information about components of the original paths is kept.

2.5 Graph traversal

Catalysts for final sequences (contigs) are nucleotide sequences corresponding to edges in the transformed overlap graph. Single edges can be merged into longer structures. This situation occurs when there are edges \((a, b)\) and \((b, c)\) long enough (in ALGA defined as at least twice the average read length), where \(d^{-1}(b) = 1\) and there exist paired reads \(r_1\) and \(r_2\) such that \(r_1\) is a fragment of a sequence represented by \((a, b)\) and \(r_2\) is a fragment of a sequence represented by \((b, c)\). From among alternative traversals through a vertex \(b\) this one is chosen which is confirmed by a greater number of pairs \(r_1, r_2\). In order to avoid duplicated sequences, corresponding to complementary strands of a DNA fragment, candidates for contigs are compared with already accepted contigs. The candidates are rejected if they are too similar to any sequence stored so far. In ALGA, the level of similarity is set to 95% measured in vertices of compared un compressed paths. The similarity is determined with taking into consideration also vertices corresponding to reverse complementary versions of reads. Finally, in order to minimize mismatch rate, contigs are trimmed from both sides so that each nucleotide is covered by at least three reads.

3 Results

In the computational experiment, ALGA was compared with assemblers highly rated in scientific community: Velvet (Zerbino and Birney, 2008), SGA (Simpson and Durbin, 2012), SOAPdenovo2 (Luo et al., 2012), SPAdes (Bankevich et al., 2012), Readjoiner (Gonnella and Kurtz, 2012), Platanus (Kajitani et al., 2014), MEGAHIT (Li et al., 2015), SAGE2 (Molnar et al., 2018), and with our previous assembler GRASShopper (Swierz et al., 2018). Altogether, we have five assemblers based on the de Bruijn graph approach (Velvet, SGA, SOAPdenovo2, SPAdes, MEGAHIT) and five based on the overlap/string graphs (SGA, Readjoiner, SAGE2, GRASShopper, ALGA). Only the assembly stages of the packages were compared (a few of them also have a scaffolding module). Versions of the programs, web pages with source code and values of parameters used in the experiment are specified in Supplementary material (including other programs supporting the computations). In Section 3.1 a wide comparison of the assemblers is made, Section 3.2 describes additional tests with a whole human genome data set.

3.1 Comparison with other assemblers

Characteristics of data sets used in tests are summarized in Table 1. They were produced by an Illumina sequencer according to the paired-end
Table 1. Characteristics of data sets used in the comparison. ‘G’ stands for genome length, ‘N’ for number of read pairs, ‘R’ for average read length, ‘I’ for average insert size, ‘D’ for average depth of coverage.

| Genome       | G [Mbp] | N [M] | R [bp] | I [bp] | D       |
|--------------|---------|-------|--------|--------|---------|
| H. sapiens ch.14 | 91.0    | 18.3  | 101    | 158    | 41      |
| C. elegans   | 100.3   | 34.3  | 110    | 225    | 75      |
| A. thaliana  | 120.3   | 48.0  | 101    | 272    | 81      |
| C. sorokiniana | 58.7    | 48.5  | 250    | 499    | 413     |
| E. coli      | 4.6     | 22.7  | 101    | 504    | 989     |
| M. parvicella| 4.2     | 4.9   | 98     | 315    | 229     |

protocol, for whole genomes of nematode C. elegans, plant A. thaliana, alga C. sorokiniana, bacteria E. coli and Microthrix parvicella, and for chromosome 14 of H. sapiens. The length of chromosome 14 is approx. 107 Mbp, but due to a large gap of unknown nucleotides (N) at the beginning, its effective length is shorter as reported in the table. A few assemblers benefited from read correction made by Musket (Liu et al., 2013), which is based on analysis of frequencies of k-mers within a data set. We performed tests for all assemblers in two configurations: with and without the correction made by Musket, and took to the comparison this configuration which gave better results. Assemblers which benefited from the Musket correction are: SOAPdenovo2, Readjoiner, SAGE2, and ALGA. It should be noted, that the authors of SAGE2 recommend to make the read correction by their own program Racer; however, Musket gave better results with SAGE2 (on average). All references for the data sets are given in Supplementary material.

Computations were done on a PC cluster consisting of general purpose CPU nodes, each equipped with two Intel Xeon E5-2697 v3 processors and 256 GB RAM, and GPGPU nodes, each with two Nvidia V100 graphics cards with Intel Xeon Gold 5115 and 18 or 754 GB RAM. Processes were run on 16 cores in parallel. Outcomes of the assemblers were evaluated with the standard tool QUAST (Gurevich et al., 2013), which reports commonly used metrics, such as genome fraction, duplication ratio, largest alignment, NG50, NG75, number of contigs, and many others. The program was run with the minimum length of contigs undergoing the evaluation set to 250, with the exception for C. sorokiniana, where it was 500 because of longer reads. Tables generated by QUAST for the outcomes are enclosed in Supplementary material (Tables S1–S6), here summarized. Average values given below have been calculated for six data sets, except for GRASShopper, which failed with the data set of A. thaliana due to too high memory usage and five values have been counted there.

Our conclusion was to exclude from further analysis these assemblers which produced results (on average) very far from users’ expectations: Readjoiner because of too small genome coverage. The most important role of assemblers is to generate reliable contigs. Typically, the longer contigs are, the greater fraction of disagreements to a reference genome they contain. Some assemblers pay too much attention to extend contigs, so they win on the metrics promoting length, but lose on the number of alignment errors. Table 2 and Fig. 3 present this aspect of results.

Table 2. Results summarized for all genomes from Table 1 (average values) from the point of view of assemblers’ functionality. Genome fraction is the part of a reference genome covered by contigs aligned to it; duplication ratio is the relation of aligned result to the aligned part of a genome (with 1 being optimum); inaccuracy is the sum of lengths of inaccurate contigs (being contigs misassembled, unaligned or partially unaligned to a reference genome) divided by the genome length.

| Assembler | Genome fraction | Duplication ratio | Inaccuracy |
|-----------|-----------------|-------------------|------------|
| ALGA      | 95.98 %         | 1.004             | 1.26 %     |
| GRASShopper | 86.44 %       | 1.053             | 1.76 %     |
| MEGAHIT   | 96.00 %         | 1.017             | 30.33 %    |
| Platanus  | 92.30 %         | 1.008             | 0.47 %     |
| Readjoiner| 44.54 %         | 1.133             | 2.90 %     |
| SAGE2     | 76.65 %         | 1.008             | 4.83 %     |
| SGA       | 95.77 %         | 1.008             | 1.28 %     |
| SOAPdenovo2| 91.89 %        | 1.008             | 1.12 %     |
| SPAdes    | 96.62 %         | 1.004             | 27.76 %    |
| Velvet    | 81.93 %         | 1.015             | 6.01 %     |

Fig. 3. Results shown as the dependency of NG50 values and the length of misassemblies. NG50 means the length of a contig that together with at least such long contigs cover half of a reference genome. Total misassembly length is the sum of lengths of inaccurate contigs (being contigs misassembled, unaligned or partially unaligned to a reference genome). The better the results are, the closer to the right-bottom part of the graph they are visualized. Axes are in logarithmic scale.
As an additional measure indicating accuracy of assemblers’ outcomes we took BUSCO, which returns a number of gene sequences, at the level of order or phylum of an organism, correctly recognized (or not) within produced contigs (Seppey et al., 2019). Table 3 contains values of this measure summed up for all data sets, detailed results for particular organisms are given in Supplementary material (Tables S7–S12).

3.2 Whole human genome

Separate tests were done with a whole human genome data set (specified in Supplementary material). It contains 359 M pairs of reads of average length 151, average depth of coverage for the genome is 34. Previous described details of the computational experiment remain valid, with RAM of CPU nodes expanded to 320 GB. Only four assemblers finished computations with a result: ALGA, MEGAHIT, SGA, SOAPdenovo2; others either did not fit within available memory or had other runtime error. MEGAHIT once again produced contigs of inaccurate inaccuracy, this time 23.4%.

SOAPdenovo2 obtained genome fraction 85.7% only. ALGA and SGA achieved results of similar high quality, ALGA much faster, SGA with much lower memory usage. In genome fraction, SGA was slightly better, 90.5% vs. 90.3% for ALGA, but with a high value of duplication ratio: 1.108 vs. 1.099 for ALGA. ALGA assembled much longer contigs than SGA, ALGA’s NG50 value is 2.5 times greater (11 495 vs. 4481), largest subgraph, achieved overall the second best result for ALGA, SGA, and Platanus, ALGA is the fastest one and, in the average usage of memory, the second one (after SGA). The respective values for ALGA, SGA, and Platanus, average for data sets from Table 1, are: 4.7, 3.2, 108.5 (memory peak in GB) and 50, 190, 77 (time in minutes, incl. Musket for ALGA).

As to the quality of results of assemblers from particular groups, it can be observed that three assemblers of highest inaccuracy (counted as in Table 2) use the de Bruijn graph model. However, another assembler from that group (Platanus) has the smallest inaccuracy value.

Tests with data from sequencing the whole human genome showed that only two assemblers were able to reconstruct the genome with satisfactory values of quality measures: ALGA and SGA. Both follow the overlap/string graph approach. They returned results of similar quality, except for duplication ratio, NG50, and largest alignment, where ALGA achieved much better values.

We used to implement the overlap-layout-consensus strategy in our methods for DNA sequencing and assembly (e.g. Blazewicz et al., 2002, 2009; Swiercz et al., 2018) as giving more reliable contigs, where the most significant problem was time and memory requirements. Now, ALGA manages large data sets, even of big depth of coverage. The genome of A. thaliana, of the length over 120 Mbp and covered with the average depth 81, appeared to be the most difficult data set in the first part of the computational experiment, according to the greatest factor of incorrectly assembled contigs. Actually, the big coverage depth brought many new possibilities of connecting reads and traversing through forks in a graph, and assemblers hardly coped with such a situation. ALGA, by preferring errorless connections and doing a series of graph reduction steps, especially the one referring to the problem of minimum spanning tree of a local subgraph, achieved overall the second best result for A. thaliana, and the best results for other highly covered genomes of C. seokokiana (the average depth 413) and E. coli (the average depth 989). It fits within memory constraints due to bit representation of data and building a sparse graph, and shortens computation time by a new procedure of comparing reads.

Switching to the error-free mode of aligning reads (by setting the parameter of error rate to zero) is worth attention as a decision toward optimizing computation time, memory usage and quality of solutions. A difficulty in the overlap graph approach which cannot be ignored is to establish a final consensus sequence from partial information about overlapping pairs of reads (neighbors in an already found path). It is a hard computational problem consisting in solving a series of problems of multiple sequence alignment in subsequent segments of a solution.

In practice, due to time requirements, it is solved by a simple heuristic value. The second and third place in this ranking is taken by SGA and Platanus, respectively.

From the point of view of contig lengths, the particular results in Fig. 4 show that ALGA produced best results for each data set except A. thaliana, where it was ranked after SGA. At the same time, ALGA generates the smaller number of contigs, averaged for all data sets. The longest aligned contig belongs to ALGA for H. sapiens and A. thaliana, for other data sets ALGA takes the second position after SGA or Platanus (not counting MEGAHIT and SPAdes).

Wide representation of tested assemblers and their equal allocation to the models of de Bruijn and overlap/string graphs allow us to infer basic predispositions of these models. The supremacy of de Bruijn-based assemblers according to time and memory usage was already known, but it happens that the other assemblers perform better; complete tables with such information can be found in Supplementary material (Tables S13–S15). Two best results in memory usage have been achieved by the overlap/string graph assemblers Readjoner and SGA. Readjoner also worked in shortest time, with ALGA being next. In memory usage ALGA is ranked in the middle. When considering the group of three best-quality assemblers from the computational experiment, ALGA, SGA, and Platanus, ALGA is the fastest one and, in the average usage of memory, the second one (after SGA). The respective values for ALGA, SGA, and Platanus, average for data sets from Table 1, are: 14.7, 3.2, 108.5 (memory peak in GB) and 50, 190, 77 (time in minutes, incl. Musket for ALGA).

As quality of results of assemblers from particular groups, it can be observed that three assemblers of highest inaccuracy (counted as in Table 2) use the de Bruijn graph model. However, another assembler from that group (Platanus) has the smallest inaccuracy value.

Tests with data from sequencing the whole human genome showed that only two assemblers were able to reconstruct the genome with satisfactory values of quality measures: ALGA and SGA. Both follow the overlap/string graph approach. They returned results of similar quality, except for duplication ratio, NG50, and largest alignment, where ALGA achieved much better values.

We used to implement the overlap-layout-consensus strategy in our methods for DNA sequencing and assembly (e.g. Blazewicz et al., 2002, 2009; Swiercz et al., 2018) as giving more reliable contigs, where the most significant problem was time and memory requirements. Now, ALGA manages large data sets, even of big depth of coverage. The genome of A. thaliana, of the length over 120 Mbp and covered with the average depth 81, appeared to be the most difficult data set in the first part of the computational experiment, according to the greatest factor of incorrectly assembled contigs. Actually, the big coverage depth brought many new possibilities of connecting reads and traversing through forks in a graph, and assemblers hardly coped with such a situation. ALGA, by preferring errorless connections and doing a series of graph reduction steps, especially the one referring to the problem of minimum spanning tree of a local subgraph, achieved overall the second best result for A. thaliana, and the best results for other highly covered genomes of C. seokokiana (the average depth 413) and E. coli (the average depth 989). It fits within memory constraints due to bit representation of data and building a sparse graph, and shortens computation time by a new procedure of comparing reads.

Switching to the error-free mode of aligning reads (by setting the parameter of error rate to zero) is worth attention as a decision toward optimizing computation time, memory usage and quality of solutions. A difficulty in the overlap graph approach which cannot be ignored is to establish a final consensus sequence from partial information about overlapping pairs of reads (neighbors in an already found path). It is a hard computational problem consisting in solving a series of problems of multiple sequence alignment in subsequent segments of a solution.

In practice, due to time requirements, it is solved by a simple heuristic
Genome-scale de novo assembly using ALGA

Homo sapiens chr. 14

Caenorhabditis elegans

Arabidopsis thaliana

Chlorella sorokiniana

Escherichia coli

Microthrix parvicella

Fig. 4. Results shown as the part (per cent) of a reference genome covered by contigs aligned to it, depending on the minimal length of contigs taken into account. X axis is in logarithmic scale.

method giving non-optimal results. The de Bruijn graph approach does not have such a problem, because all connections appearing there are exact. The assumption of error-free connections in the overlap graph seemingly worsens a solution, but in practice, for currently obtained depth of coverage and small percentage of sequencing errors, it works very well (as shown in Results), even for the whole human genome assembid at once. But even in the case of errors allowed, the problem of establishing a consensus sequence in ALGA is turned to be computationally easy thanks to the assumed model of errors restricted to substitutions only, and we solve it optimally in polynomial time. The advantage of the error-free mode of ALGA over the mode with errors allowed, for Illumina data sets of a typical quality, can be observed through results shown in Supplementary material (Table S17).

5 Conclusion

The proposed method for sequence assembly, ALGA, proved to be effective in tests on a genome scale, including data sets coming from Illumina sequencing of several model organisms. It achieved better results than nine other tested assemblers in the sense of overall quality of genome reconstruction. In particular measures, like genome coverage, duplication ratio, accuracy, NG50, number of contigs, and others, ALGA is ranked as the best or nearly the best. It is also the second fastest assembler with memory consumption kept at a medium level, which comes to the fore when restricting attention to assemblers giving best quality contigs.

Long reads SMS technologies, such as the ones developed by Pacific Biociences or Oxford Nanopore, becomes the key mean to sequence de novo long or repetitive genomes. Assembly tools based on the de Bruijn graph model are preferred for shorter reads because of the information lost following the read decomposition. But also a high error rate connected with the new technologies poorly corresponds with the decomposition graphs. As we mentioned in Introduction, the sequencing errors are represented in these graphs by multiplying k-mers in versions differing in erroneous places. Further graph exploration bases on the possibility of identification which k-mers seem to be false positives, which can be efficiently done only if they are not too numerous comparing to the rest of the graph. This is the room to apply assembly methods following the overlap graph approach, and the efficient implementation of ALGA can be a part of a method satisfying practical expectations in this area.

Funding

The research has been done based on infrastructure developed within European Center for Bioinformatics an Genomics project (Genomic Map of Poland, www.ecbig.pl/page/genomic-map-of-poland/), grant no. POIR.04.02.00-30-A004/16 supported by the European Regional
Author contributions
JBI, MK, AS and PW conceived the study and supervised tasks of the project. SS developed and implemented the algorithms. SS, AL, JBl and WF carried out computational experiments. MK and SS drafted the manuscript. AS drew up results of computations. All the authors were involved in discussions, data analysis and proofreading, and approved the final version of the manuscript.

References
1001 Genomes Consortium (2016) 1.135 Genomes reveal the global pattern of polymorphism in Arabidopsis thaliana. Cell, 166, 481–491.
Ameur, A., Kloosterman, W.P., and Hestand, M.S. (2019) Single-molecule sequencing: towards clinical applications. Trends Biotechnol., 37, 72–85.
Bankevich, A., Nurk, S., Antipov, D., Gurevich, A., Dvorkin, M., Kulikov, A., et al. (2012) SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. J. Comput. Biol., 19, 455–477.
Blazewicz, J., Bryja, M., Figlerowicz, M., Gawron, P., Kasprzak, M., Kirtun, E., Platt, D., Przybytek, J., Swiercz, A., and Szajkowski, L. (2009) Whole genome assembly from 454 sequencing output via modified DNA graph concept. Comput. Biol. Chem., 33, 224–230.
Blazewicz, J., Formanowicz, P., Guinand, F., and Kasprzak, M. (2002) A heuristic managing errors for DNA sequencing. Bioinformatics, 18, 652–660.
Blazewicz, J., Kasprzak, M., Kierzyńska, M., Frohnbreg, W., Swiercz, A., Wojciechowski, P., and Zurkowski, P. (2018) Graph algorithms for DNA sequencing - origins, current models and the future. Euc. J. Oper. Res., 264, 799–812.
Bradnam, K., Fass, J., Alexandrov, A., Baranay, P., Bechner, M., Birol, I., et al. (2013) Allpaths-Limbo: evaluating de novo methods of genome assembly in three vertebrate species. Gigasience, 2, 1–10.
Gonnella, G. and Kertz, S. (2012) Readjoiner: a fast and memory efficient string graph-based sequence assembler. BMC Bioinformatics, 13, 82.
Gurevich, A., Saveliev, V., Vyahhi, N., and Tesler, G. (2013) QUAST: Quality assessment tool for genome assemblies. Bioinformatics, 29, 1072–1077.
Idury, R. and Waterman, M. (1995) A new algorithm for DNA sequence assembly. J. Comput. Biol., 2, 291–306.
Ilie, L., Haider, B., Molnár, M., and Solís-Oba, R. (2014) SAGE: string-overlap assembly of genomes. BMC Bioinformatics, 15, 302.
Kajitani, R., Toshimoto, K., Noguchi, H., Toyoda, A., Ogura, Y., Okuno, M., et al. (2014) Efficient de novo assembly of highly heterozygous genomes from whole-genome shotgun short reads. Genome Res., 24, 1384–1395.
Kawakatu, T., Huang, S.C., Jupe, F., Sasaki, E., Schmitz, R.J., Urich, M.A., et al. (2016) Epigenomic diversity in a global collection of Arabidopsis thaliana accessions. Cell, 166, 492–505.
Keececioglu, J. and Myers, E. (1995) Combinatorial algorithms for DNA sequence assembly. Algorithmica, 13, 7–51.
Li, D., Liu, C., Luo, R., Sadakane, K., and Lam, T. (2015) MEGAHIT: an ultra-fast single-node solution for large and complex metagenomes assembly via succinct de Bruijn graph. Bioinformatics, 31, 1674–1676.
Liu, Y., Schroder, J., and Schmidt, B. (2013) Musketa: a multistage k-mer spectrum-based error corrector for Illumina sequence data. Bioinformatics, 29, 308–315.
Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., et al. (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. GigaScience, 1, 18.
Lyuov, Yu., Florentiev, V., Khorlin, A., Khrapko, K., Shik, V., and Mirzabekov, A. (1988) Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. Dokl. Akad. Nauk SSSR, 303, 1508–1511.
Minogue, T.D., Koehler, J.W., Stefan, C.P., and Conrad, T.A. (2019) Next-generation sequencing for biodefense: biothreat detection, forensics, and the clinic. Clin. Chem., 65, 383–392.
Molnar, M., Haghshenas, E., and Iie, L. (2018) SAGE2: parallel human genome assembly. Bioinformatics, 34, 678–680.
Myers, E., Sutton, G., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., et al. (2000) Whole-genome assembly of Drosophila. Science, 287, 2196–2204.
Pereira-Marques, J., Hout, A., Ferreira, R.M., Weber, M., Pinto-Ribeiro, I., van Doorn, L.-J., et al. (2019) Impact of host DNA and sequencing depth on the taxonomic resolution of whole metagenome sequencing for microbiome analysis. Front. Microbiol., 10: 1277.
Pevzner, P. (1989) 4-tuple DNA sequencing: computer analysis. J. Biomol. Struct. Dyn., 7, 63–73.
Pevzner, P., Tang, H., and Waterman, M. (2001) An Eulerian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. U.S.A., 98, 9748–9753.
Seppey, M., Manni, M., and Zdobnov, E. (2019) BUSCO: assessing genome assembly and annotation completeness. In: Kollmar, M. (ed.) Gene Prediction, 227–245, series Methods in Molecular Biology vol. 1962, Humana, New York.
Simpson, J. and Durbin, R. (2012) Efficient de novo assembly of large genomes using compressed data structures. Genome Res., 22, 549–556.
Siva, N. (2015) UK gears up to decode 100000 genomes from NHS patients. Lancet, 385, 103–104.
Swiercz, A., Frohnbreg, W., Kierzyńska, M., Wojciechowski, P., Zurkowski, P., Badura, J., Laskowski, A., Kasprzak, M., and Blazewicz, J. (2018) GRASShopPER - An algorithm for de novo assembly based on de Bruijn graphs. Genome Res., 18, 821–829.