Information Sciences 420 (2017) 184–199

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A stochastic de novo assembly algorithm for viral-sized genomes obtains correct genomes and builds consensus

Doina Bucur

University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

ARTICLE INFO

Article history:
Received 21 January 2017
Revised 18 June 2017
Accepted 25 July 2017
Available online 19 August 2017

Keywords:
De novo
DNA
Assembly
Genetic algorithm
Consensus genome

ABSTRACT

A genetic algorithm with stochastic macro mutation operators which merge, split, move, reverse and align DNA contigs on a scaffold is shown to accurately and consistently assemble raw DNA reads from an accurately sequenced single-read library into a contiguous genome. A candidate solution is a permutation of DNA reads, segmented into contigs. An interleaved merge operator for contigs allows for the quick minimization of a fitness function measuring the string length of a candidate solution. This study assembles read libraries for three genomic fragments from different organisms, five complete virus genomes, and one complete bacterial genome, with the largest genome length of 159 kbp. To evaluate the accuracy of any assembled genome, test libraries of DNA reads are generated from reference genomes, and the assembly is compared to the reference. The method has very high assembly accuracy: over repeated assemblies for each input genome, the original genome was constructed optimally in over 85% of the runs. Given the consistency of the algorithm, the method is suitable to determine the consensus genome in de-novo assembly problems. There are two limitations to the method: genomes with long repeats may be overcompressed, and the computational complexity is high.

1. Introduction

Problem definition. Genome assembly for a new organism should turn raw sequencing data into a complete genome. As the cost of sequencing decreases, the number of projects attempting de-novo assembly increases, yet problems remain: it is not known what combinations of sequencing data and assembly algorithms lead to accurate genomes, nor how to evaluate the accuracy of the new, resulting genome [2]. The raw sequencing data is a library of overlapping, either single or paired-end DNA or RNA reads (strings over the four-letter alphabet of bases {A,C,G,T}), from either of the two reverse-complementary strands of a DNA molecule, or in either sense of the same genomic strand (an example shown in Fig. 1). With the widely available second-generation sequencers, single reads are short, uniform-length, accurate (on the order of 10^2 bases with 99.9% single-read accuracy for Illumina dye sequencing [13]), and cover the underlying genome multiple times over, with overlap between any adjacent reads. The *coverage* c is $c = n \cdot r/G$, where n is the number of single reads, r the read length, and G the length of the underlying genome.

To assemble a genome, the raw DNA reads must be linearly ordered such that each pair of adjacent reads overlaps. Continuously overlapping sequences of short reads form a *DNA contig*, with the contigs ordered on a *DNA scaffold* which

E-mail address: d.bucur@utwente.nl

http://dx.doi.org/10.1016/j.ins.2017.07.039
0020-0255/© 2017 Elsevier Inc. All rights reserved.

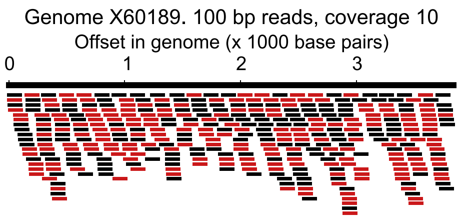

Fig. 1. Raw sequencing data of single, 100-bp reads for a human MHC class III region DNA [27] of length 3835 bp, shown aligned against the original genome. Reads from the forward DNA strand are shown in black, and those from the reverse-complement are shown in red (gray in print). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

is the resulting genome. This is a combinatorial problem, yielding a computationally hard assembler [36]; the larger the genome and the deeper its coverage, the larger the read set in input and the harder the computation.

Existing methods are difficult to evaluate decisively. Importantly, evaluating a de-novo assembly (and the algorithm which produced it) in practice does not often have the benefit of an existing reference genome to compare against. Instead, the resulting genomes are mostly evaluated by the number of contigs in the assembly (this is the case for the prior works using genetic algorithms), the fraction of raw reads that could be assembled, the length of the contigs and scaffold, and the length of the contigs and scaffold relative to the estimated size of the genome [2]. Unfortunately, these so-called *assembly metrics* have been recently shown in comparative studies (some using simulated raw data with a reference genome, for a decisive evaluation of genome correctness) not to accurately reflect the quality of an assembly [2,4,7,37].

Furthermore, comparative studies of assembly algorithms found no best algorithm. In [37], the contiguity of an assembled genome varied wildly among both assemblers and genomes. When comparing the assemblies obtained to a reference genome, the following issues were found: many “chaff” contigs (of very short length), duplicated contigs, compressions of repeat sequences (a widespread problem for short-read assemblers), and contig “misjoins” (i.e., the assembler joined two contigs which are in fact distant in the reference genome). The second Assemblathon [4] also found a high degree of variability among assemblies, to the extent that the result of a single assembler and set of assembler parameters is not to be trusted. This supports the idea of *consensus*: an assembly obtained by different methods is likelier to be accurate.

Two decisive evaluation metrics: reference and consensus. When a reference genome exists, it is the best basis to write *accuracy metrics* (e.g., the degree to which the reference is covered by the assembly, and the percentage of contigs alignable onto the reference). In absence of a reference, *consensus* is used as an effective evaluator: a newly sequenced genome is assumed biologically correct if a number of different assembly algorithms, or repeated runs, have computed it. Obtaining consensus is effectively the current accuracy metric in absence of a reference genome [2]. Our algorithm is evaluated against references, and found to be often correct and fairly consistent on the set of genomes assembled here. Given its consistency, the method is a good means by itself of obtaining the consensus genome.

Summary of this contribution.

- (Algorithmic.) We improve prior genetic algorithms designed for de-novo sequencing with two novel genetic operators: an interleaved merge operator for contigs of any length, and an align operator which merges “chaff” (short) contigs at the appropriate location within long contigs. Both operators serve to speed up the optimization of the fitness function and effectively avoid stagnating in local minima. An early version of the algorithm [5] identified the potential for the align operator to refine a sub-optimal candidate solution into the optimal solution.
- (Experimentation.) Nine genomes without long repeats are assembled: three human and fungal genomic fragments of maximum length 77 kbp, five complete virus genomes (two phage viruses, a strain of Hepatitis C, one of HIV-1, and one of the Zaire Ebola virus) of maximum length 48 kbp, and one complete genome from a small bacterium of length 159 kbp. The original genomes serve as references in the evaluation stage.
- (Optimality of assemblies.) The algorithm is run against raw read libraries sampled from the reference genomes, with single raw reads of uniform length between 100 and 700 bp, and contiguous genome coverage between 5 and 10. From each genome, 10 raw read sets are sampled (to determine whether the algorithm is sensitive to the read set), and for each raw data set, 20 assemblies are executed with different random seeds. The resulting assembly is evaluated for correctness (is it identical to the reference?), and the fraction of correct assemblies is calculated. In all tests, between 85% and 100% of the 200 assemblies per genome gave a *biologically correct assembly* (in other words, computed the optimal solution). The bottleneck to scaling up the algorithm to larger genomes and raw read sets is its runtime and not its ability to compute optimally.
- (Consensus among assemblies.) Given this success rate, the algorithm on its own can serve effectively as a consensus builder for de-novo assemblies, in lieu of a reference genome.

Table 1
The design of the genetic algorithms in related work.

	[31]	[32–34]	[1,8,9,30]	[11,12]
candidate representation	sorted-order representation	permutation	permutation	(a) permutation; (b) ordered transpositions
mutation operators	classic bit point mutation (micro)	read swap (micro), contig inshift (macro), contig reverse (macro)	read swap (micro)	read swap (micro)
crossover operators	classic bit two-point crossover (micro)	edge recombination (micro), order crossover (micro)	order crossover (micro)	partially mapped crossover (micro)
fitness functions	total adjacent overlap (↑) total distant overlap (↓)	total adjacent overlap (↑) total distant overlap (↓)	total adjacent overlap (↑) number of contigs (↓)	total adjacent overlap (↑)

2. Related work

Current genome assemblers implement heuristics which compute an order for the reads, maximizing their overlap and minimizing the number of unused reads. A number of assemblers exist; recent comparative studies [4,7,37] provide an overview of the performance of commercial-grade assemblers on raw sequenced data. GAGE [37] experimented with 8 assemblers for paired-end short reads, and evaluated the accuracy of the results with reference genomes. The Assemblathon [4] compared 43 assemblies and worked with both short- and long-read, single- and paired-end libraries, without evaluating the accuracy of the results with reference genomes. The Nanopore benchmark in [7] used long-read MinION sequence data, and compared 4 assemblers implementing different algorithms.

All studies found “chaff” contigs, duplicated contigs, compressions of repeat sequences in the reference, and contig “misjoins”, to the conclusion that a single assembler and set of assembler parameters cannot be trusted, supporting the need for *consensus* assemblies. In [37], the *assembly metrics* (e.g., the number of contigs obtained, or various basic statistics over the sizes of the contigs) used to evaluate the assembly in absence of a reference were found not to correlate well with *accuracy metrics* (e.g., the degree to which the reference is covered by the assembly, and the percentage of contigs alignable onto the reference). Based on [4], it remains unclear how to assess the quality of the assembly. Assemblers benchmarked in [7] obtained inaccurate assemblies: for one species, all assemblers had very low reference coverage (between 0% and 12%); a greedy assembler had both the genome coverage and the percentage of alignment under 5% for both species.

2.1. Genetic algorithms

Genetic algorithms have found broad application, as varied as optimizing the management of renewable energy sources [3], data mining [39], and materials science [35]. A number of tools have implemented a genetic algorithm for the assembly problem. Table 1 summarizes this prior work. A first genetic algorithm for this problem was based on that for solving TSP [31]; a complex individual representation was used, which was found not to build increasingly improved solutions. However, the two fitness functions designed here were carried forward by later work. They essentially compute the total overlap between adjacent fragments on a scaffold (to be maximized), and the total overlap among all fragment pairs at distant locations on the scaffold (to be minimized).

The follow-ups [32,34] (with parameter tuning in [33]) preserve these fitness functions, but switch to a simpler representation as a *permutation*: an ordered list of read identifiers. This requires genetic operators specialized for this problem; they are either *micro* operators working with raw reads, or *macro* operators working with contigs. These operators are listed below and briefly compared with those used in this work (for which Section 3 gives the full description):

Order crossover (micro). The precursor to our one-point contig-based Order crossover **O**: random read indices l and r are selected, the subsequence between l and r on the first parent is copied into the offspring preserving absolute position, and the remaining slots in the offspring are filled from left to right with the reads not yet in the offspring, in the relative order in which they appear in the second parent. Our work uses a macro version instead, whose advantage is that, by only selecting contig indices, more existing contigs are preserved unbroken.

Edge recombination (micro). Greedily attempts to preserve read adjacencies from the parents into the offspring. Selects the first read r from the first parent, and follows it with that read s which (i) is adjacent to r in both parents, or, failing that, (ii) has the most adjacencies left. This operator has no equivalent in our method.

Swap (micro). Two random reads are swapped; in [33], late swaps become greedy by overlap rather than randomly to avoid local optima. Instead of a swap, our method has the Inshift **I** operator, which is macro.

Inshift (macro): moves a random contig between a random two previously adjacent contigs. Our method preserves this.

Reverse-complement (macro). Reverts a random contig (with the contig bounds being probabilistic in [33]). This macro operator is also preserved in this work, using hard contig bounds.

From the two fitness functions, the first is found to be adequate, but not ideal; neither of the two functions is used in this work.

PALS [1] (with a grid-based parallelization in [30]) adds as fitness function the number of contigs, an advantageous design which our method carries forward. SACMA [8] is a cellular genetic algorithm combined with a local search from PALS; it obtained the most contiguous large assemblies among the GAs (Table 5). In the later [18] and [9], the methods deal with noisy raw reads (e.g., with erroneous calls for some of the base pairs), a feature not yet present in our work. In [9], two swarm-intelligence algorithms are presented: the Artificial Bee Colony (ABC) algorithm and the Queen Bee Evolution Based on Genetic Algorithm (QEGA); the largest contiguous genome it assembled was the 48 kbp **J02459** with (compared to ours) a smaller read library of longer, 700-bp reads. In [18], PALS is combined with Simulated Annealing, and does not obtain contiguous assemblies for genomes larger than 20 kbp. They preserve the genetic design (fitness functions, candidate representation) from PALS. [11] and its extended version [12] also reuse a classic candidate representation, genetic operator, and overlap-based fitness function from prior literature, but experiment with variations of a GA with restarts and recentering, different variants of which are shown to match or outperform prior classic metaheuristics such as PALS and QEGA and the Lin-Kernighan heuristic.

In none of these studies is the biological accuracy of the assembly evaluated; the focus remains on maximizing overlap-based fitness functions, and minimizing the number of contigs in the output.

Finally, an early version of our own algorithm [5] lacked the **Align** operator, and experimented with three genomes (the largest of which was the 48 kbp **J02459**) using 400-bp read libraries; it highlighted the potential for the correct genome to be reconstructed.

Overall, our method deals successfully (i.e., computed optimal, biologically correct genomes) using genomes and noiseless raw-read libraries roughly twice the length of those assembled by prior GAs. It has the distinct advantage of having verified biological correctness, rather than assuming that a contiguous assembly or a high total overlap might also be optimal. An extension of this method is presented in the recent [6]; it preserves the advantages of the GA presented here, and adds two features: (a) the ability to assemble variable-length reads, and (b) the ability to assemble circular genomes. Both features make a step towards this tool being useful for assembling genomes with interspersed repeat sequences.

2.2. Other heuristics and algorithmic combinations

Comparative studies [4,7,37] give benchmarks of industrial-grade assemblers, which come in three large algorithmic categories: greedy, Overlap-Layer Consensus, and based on de-Bruijn graphs; all have the great advantage that their time complexity is low, and thus can attempt to assemble much larger read libraries than a genetic algorithm. Other assemblers implement algorithms different than the categories above. [14] studies variations of four prior greedy, genetic, cluster-based, and pattern-matching algorithms. FGS [19] does a rather inaccurate fuzzy genome sequencing using dynamic programming, for shorter genomes and smaller read libraries than solved in this study.

Besides comparing with related genetic algorithms, our algorithm is also compared against [16], a PSO/DE method with a Lin-Kernighan heuristic, shown to indeed be optimal in fitness (by comparing the results to those of an exhaustive-search method). The method scales up to a read library for the 2 Mbp-genome Staphylococcus aureus, i.e., 18 thousand reads, with a mean read length of 540 bp; it computed a very fragmented scaffold with no fewer than 1315 contigs for this genome, the largest of which contained 286 reads. A graph-based minimum spanning tree algorithm is implemented in [17] and is tested with genomes up to 2 Mpb, also obtaining many chaff contigs.

Three other assemblers are also compared against in this work. SSAKE [40], SOAPdenovo [15], and Velvet [41] were chosen based on their algorithmic category, ease of use, and for known performance: SOAPdenovo and Velvet (based on de-Bruijn graphs with added heuristics) were occasionally found to be competitive in terms of assembly contiguity [37]. SSAKE uses a greedy iteration: it iteratively searches for the longest possible overlap between any two reads using a preconstructed prefix tree in the search, and is highly sensitive to the exact raw read library sequenced from the underlying genome.

3. Methodology

The method does stochastic optimization using an iterative, population-based, evolutionary-inspired algorithm in which a candidate solution is an ordering of the raw reads in input, further grouped into contigs, i.e., subsequences of overlapping reads. At each iteration (or generation) in the genetic algorithm, a population of candidate solutions are comparatively evaluated with a composite fitness function which measures the total length of the contig strings in the candidate. The best among the candidates in a population are used to generate the next generation, via mutations and crossover genetic operations which work natively with contigs rather than raw reads. The generation of the candidate solutions in the first generation, the selection of candidates for comparative evaluation, and the genetic operators are all stochastic in nature, so that repeated runs of the algorithm with different random seeds are necessary to characterize its behavior.

This assembly problem has been recast in the literature into the shortest common superstring (SCS) problem, i.e., finding, for n finite strings $s_1, s_2, \dots, s_n$ over an alphabet of size greater than 2, a shortest superstring S such that every string s_i can be obtained by deleting zero or more elements from S. The SCS problem is known to be NP-complete [36], as is the simpler version of SCS in which the n strings are totally ordered in a superstring [31], which is essentially what we attempt to solve. Our method is built on prior genetic algorithms for the SCS or assembly problem, e.g. [1,8,9,34], which will be described in Section 2.

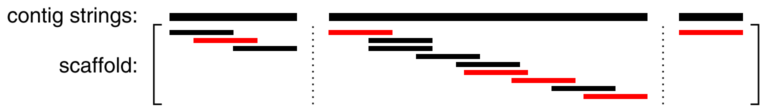

Fig. 2. A candidate solution for a problem with 13 raw reads in input, and 3 contigs on the scaffold. Reads with reverse-complement bits True are shown in red (gray in print); reads with reverse-complement bits False are shown in black. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Summary of mutation operators, crossover operators, and fitness functions. Single-letter abbreviations are introduced for operators. The fitness functions are marked with ↓, since they will be minimized.

Mutation operators	(**R**) contig **Reverse-complement**
	(**S**) contig **Split**
	(**I**) contig **Inshift**
	(**M**) contig **Merge**
	(**A**) contig **Align**
Crossover operators	(**O**) scaffold one-point **Order crossover**
Fitness functions	length of DNA scaffold (↓) number of DNA contigs (↓)

3.1. Representation of candidate solution

A *segmented permutation* first arranges the raw, uniform-length DNA reads in the input in a total order, to model a DNA scaffold. Then, it logically segments this scaffold into DNA contigs, where each contig is a subarray of raw DNA reads, under the condition that every adjacent pair of reads in a contig overlaps by at least a minimum amount of overlap (typically on the order of 10 bp). Fig. 2 shows a sketch of a candidate solution over 13 raw reads and 3 contigs.

A candidate solution is thus an object with the following attributes:

DNA scaffold: a segmented permutation of the raw reads in input, of which each segment models a contig, and adjacent reads in a contig overlap.

Reverse-complement bits: for each raw read, a Boolean variable which is True if the corresponding read is used in its reverse-complemented form in this candidate scaffold with respect to the input read set.

DNA contig strings: for each DNA contig on the scaffold, the string obtained by overlapping the raw reads in the contig.

For all candidate solutions in the initial population, all contigs contain a single read, and the contigs are ordered randomly on the scaffold. The sum of the lengths of these contig strings will thus start at value nr, where n is the number of reads in the input set, and r is the raw read length, and will decrease in time, as contigs merge.

3.2. Operators over candidate solutions

The genetic operators applied to a candidate solution at each iteration of the genetic algorithm are summarized in Table 2 and sketched visually in Fig. 3.

Reverse-complement: A randomly chosen contig string in the representation becomes the reverse-complement of the original, by reverse-complementing the corresponding contig on the scaffold. With this operator, the algorithm will be able to try to place a sequence of raw reads into a candidate genome regardless of which of the two complementary DNA strands (or senses, on some single-strand genomes) this candidate genome forms. Each read in the contig is complemented[1] (e.g., ACTG becomes TGAC) and both the reads and their order on the contig are reversed, with the amounts of overlap between reads unchanged (e.g., contig ACTG TGCC, in which the reads overlap by two base pairs, becomes GGCA CAGT). In the example in Fig. 3, the first contig is reverse-complemented.

Split: A randomly selected contig (with at least two reads) is split at a random point, forming two new adjacent contigs on the scaffold. Two new DNA contig strings are computed for the scaffold, and the total string length of the scaffold will increase. This operator is one means for the algorithm to escape local optima. In the example in Fig. 3, the second contig is split after internal index 6.

Inshift: A randomly chosen contig is moved to another position on the scaffold (in Fig. 3, the third contig shifts to follow the first contig). All contig lengths remain unchanged. As this operator changes relative positions of contigs on the scaffold, it makes new overlaps between adjacent contigs possible.

Merge: A pair of adjacent contigs is merged into a new contig, if and only if the two contigs overlap by at least a minimum amount; the contigs may overlap by longer than the read length, in which case the reads in the two contigs

[1] A's complement is T, C's complement is G, and vice versa.

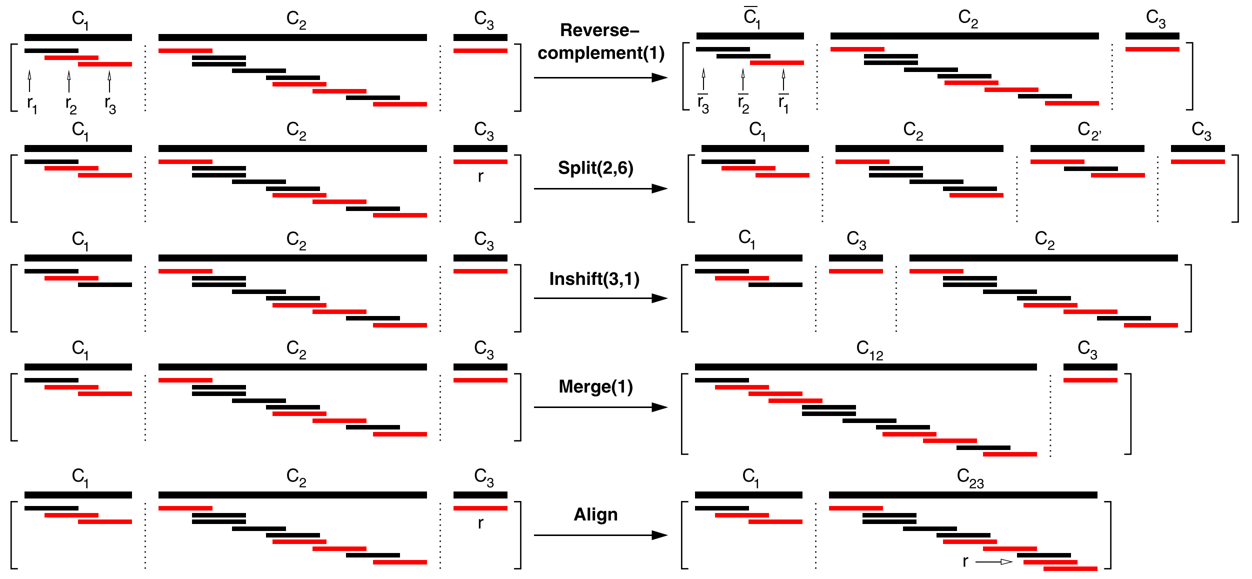

Fig. 3. Mutation operators. Reads with reverse-complement bits True are shown in red (gray in print); reads with reverse-complement bits False are shown in black. A string with an overline, e.g., $\overline{r_1}$, denotes the reverse-complement of the original string. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

will be interleaved in the final contig. A new contig string is computed, and the total string length of the scaffold will decrease; this is the main mechanism the algorithm has to "compress" the candidate solution. In Fig. 3, the first and second contigs are merged. There are two variants of this operator:

Random merge: (Default) Among all possible merges on a candidate scaffold, a random choice is made.
Greedy merge: A random choice is made among the 50% best merge options in terms of overlap.

Align: This operator aims to reduce the amount of "chaff" on the scaffold of a candidate solution, under the assumption that much of the work required to construct a correct genome was already completed, and a long, near-optimal contig exists on the scaffold (alongside leftover, short contigs yet to be incorporated in the genome). The operator tries to place the first read from the shortest existing contig c at a suitable position onto the longest existing contig C, *without increasing* the string length of C—if successful, c loses that read. This strong condition of non-increase is increasingly likely to hold in the later stages of the algorithm, if and only if C is indeed a large, contiguous part of the correct genome. In the case when C is already equal to the correct genome and c is a chaff contig, it is certain that a raw read from c will fully overlap onto C. The *threshold of length* for the longest contig C above which Align is triggered is a configurable parameter.

Order crossover: A random contig index is selected for the first parent's scaffold. All the contigs to the left of this point are preserved (in both position and internal structure) in the first offspring. The remaining reads for this offspring will come from the second parent, in the order in which they appear on the scaffold of that parent; two of these reads will be adjacent on the same contig only if they were as such on the second parent—otherwise, they will form separate contigs.

The initials of these operators, **R, S, I, M, A, O** are used in the following as abbreviations. They are applied in each iteration of the evolutionary process to a candidate solution in a configurable order (e.g., **RASIMO**), each operator at a configurable rate of application.

Operators **R, S, I**, and **M** all perform a modification of a candidate solution which cannot be achieved by any combination of other operators—in other words, this operator subset is *minimum*. **O** may be modelled by a long sequence of mutation operators; the innate advantage of crossover is speed, i.e., it drastically raises the rate of change in the population, which is particularly beneficial in the early stages of the algorithm. **A** may be modelled by two Split, one Inshift, and two Merge mutations, and has a crucial role in "polishing" a near-optimal candidate solution into an optimal one, in the late stages of the algorithm.

3.3. Candidate fitness and limitations

The main component of the fitness function is the *length of the candidate scaffold*, i.e., the sum of lengths of contig strings, which is to be minimized.

Table 3
Genomes sequenced.

	Organism	Code	Ref.	Genome size (base pairs)
Genomic fragments	Human MHC class III region DNA	**X60189**	[27]	3835
	Human apolipoprotein B-100 mRNA	**M15421**	[26]	10,089
	Neurospora crassa DNA linkage group II BAC	**BX842596**	[28]	77,292
Virus genomes (complete)	Enterobacteria phage ΦX174	**NC001422**	[22]	5386
	Hepatitis C virus	**NC009824**	[23]	9456
	Human immunodeficiency virus 1 (HIV-1)	**JQ316129**	[24]	9104
	Zaire Ebola virus	**KT589390**	[29]	18,957
	Enterobacteria phage λ	**J02459**	[21]	48,502
Bacterial genomes (complete)	Endosymbiont Carsonella	**AP009180**	[20]	159,662

Limitations. This choice of fitness function limits the problem definition, as follows: to achieve optimal solutions, the underlying genomes must be without long DNA repeats (certainly not longer than the length of the raw read); otherwise, libraries of raw reads obtained from this genome may have a shortest assembly which is even *shorter* than the original genome (a common issue for all assembly techniques, only solvable decisively by using long-read libraries). As an example, take the 13 base-pair genome `TACCCATTACCCA`, with two repeated base-pair sequences of length 6; raw reads of length 4, however extensively they overlap and cover this genome, can assemble into scaffolds of a shorter length 10, such as `TACCCATTAC` or `TTACCCATTA`. In certain cases, even shorter, but badly placed repeats will cause overcompression. Take the 13-bp genome `ACTCCATTATACT`, where the 3-bp ends are identical sequences, and take 4-bp reads; because of the overlap of the ends, the shortest assembly is the 12-bp `ATACTCCATTAT`. A further situation where this fitness is "overly optimal" consists of genomes with stretches of identical base pairs, such as the 5 base-pair repeat in `CTAAAAAGT`; if the raw reads available (say, `CTAAA` and `AAAGT`) have a length of 5 base pairs (sufficient to recover the 5-base-pair repeat), but only cover that part of the genome thinly, assembling this "thin" read library using this fitness function yields the overly compressed genome `CTAAAGT`.

The *number of DNA contigs* is the second component of the fitness, also to be minimized. The two functions are summed up.

Both fitness-function components are assembly metrics, which is an inherent limitation of doing de-novo assembly, where a reference genome for that exact organism may not be available. As *assembly metrics* do not correlate well with *accuracy metrics* (which do compare the assembly against a reference genome) [37], our algorithm is evaluated by running it with raw-read libraries obtained from a known genome, so that a post-factum evaluation of the results can employ an accuracy metric, namely the identity between our solution and the reference.

3.4. Algorithmic complexity

The time complexity of applying each genetic operator to a candidate solution with n raw reads and a read length of r is $O(nr)$ in all cases except the Greedy Merge, where an extra $O(n \log n)$ worst-case factor is needed (likely dominated by $O(nr)$). The time complexity of computing the scaffold-length fitness over a candidate with $c \leq n$ contigs on the scaffold, in our candidate representation, is $\Theta(c)$: the length of the scaffold is the sum of the lengths of the DNA contig strings, and these are stored in the representation. Computing the contig-count fitness is constant-time if the length of the data structure modelling the segmented permutation is stored explicitly in the implementation.

4. Experimental results

This section first describes the choice of genomes to assemble, and the method by which the raw read libraries are obtained. The experimental results follow, together with further experimental results using variants of the genetic operators and parameters. Finally, some limitations of the algorithm are demonstrated experimentally.

4.1. Genomes

The nine known genomes listed in Table 3 were used to evaluate the algorithm. Three of these genomes (namely, the human and fungus genomic fragments **X60189, M15421**, and **BX842596**) are benchmarks carried forward from related work on DNA assembly [1,8,9,31,34]; a fourth (the λ phage virus **J02459**) was also seen in prior work, but is experimented with here in its entirety (48502 bp), rather than by selecting the first 20 kbp of the genome (as done previously in [1,16,32,34]).

The remaining five genomes are viral and bacterial genomes of general interest: the ΦX174 bacteriophage **NC001422** is a virus whose DNA-based genome was the first to be sequenced in 1977 using the first-generation Sanger method [38]. The Hepatitis C virus strain **NC009824**, the HIV-1 strain **JQ316129**, and the Ebola variant **KT589390** are moderately sized virus genomes of interest. The proteobacterium Candidatus Carsonella ruddii **AP009180** has one of the smallest genomes among sequenced bacteria; it is included in this study due to its larger-than-virus size.

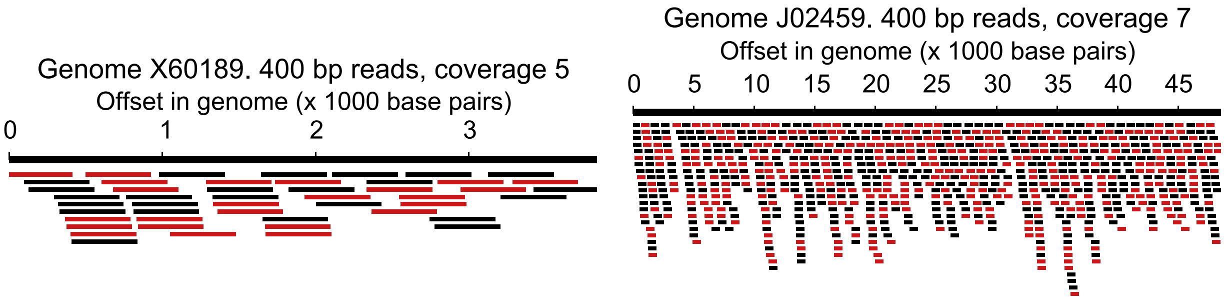

Fig. 4. Raw sequencing data of single, 400-bp reads for the **X60189** genome of length 3835 bp and the **J02459** genome of length 48,502, shown aligned against the original genome. Forward reads are shown in black, and those reverse-complemented are shown in red (gray in print). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 4
Default parameters for the GA.

	Value
Population size	100
Termination condition	100 stagnating generations
Selection	Tournament selection, size 5
Elites	1
Operator order of application	**RASIMOM**
R, S, I, M mutation rates	0.5
A mutation rate	1
A threshold (fraction of estimated genome length)	0.75
O mutation rate	0.1
Read libraries per test case	10
Runs per read library	20
Minimum overlap (bp)	20, 30

These nine genomes lack long repeated sequences and long subsequences of identical base pairs which would trigger the limitations of the method described in Section 3.3.

4.2. Raw read libraries

Raw reads of a uniform length are extracted from the genomes described in Section 4.1, by randomly sampling substrings of each genome so that (a) the substrings give an average coverage c (here, low to medium, between 5 and 10) for the genome, which essentially translates into the library having a certain number n of raw reads of length r (100, 400, or 700 bp), and (b) the substrings form a contiguous cover for the genome, with any two adjacent reads overlapping by at least a minimum number of base pairs m (either 20 or 30 bp). A random half of the raw reads obtained are reverse-complemented, to simulate the situation in which they are sequenced from either sense or strand of a genome.

Three raw read libraries are shown in Figs. 1 (in Section 1), and 4 (in this section), for the two genomes **X60189** and **J02459**, three choices of coverage $c \in \{5, 7, 10\}$, and raw read lengths 100 or 400 bp. The largest of these three read libraries contains 849 400-bp reads. The largest read library assembled in this study is sampled from the **KT589390** genome and contains 1896 100-bp reads.

Since it is entirely possible that the quality of assemblies varies for two distinct raw read libraries obtained from the same genome (particularly for low degrees of genome coverage), for each genome, coverage c, and read length r in test, 10 read libraries are obtained. Examples are shown in Figs. 1 and 4. Each library is sequenced via 20 repetitions of the assembly algorithm, using different random seeds.

4.3. Operator order, parameters, and software implementation

The assembly runs maintain a *default configuration* for the genetic algorithm, across all test instances, unless another configuration is mentioned explicitly. Table 4 lists the operator order and parameter values which are our defaults.

A modest population size of 100 (with a matching terminating condition of 100 stagnating generations) proved sufficient for this algorithm to be reasonably successful over all our test cases.

At each generation, a selected candidate solution goes through zero or more genetic operators in sequence. The default sequence of operators **RASIMOM** was constructed as follows: Operators **R, S**, and **I**, which modify a candidate scaffold moderately and will not improve fitness, are grouped together and executed apart from operator **O**, which also does not improve fitness, but can modify a scaffold to a large degree. The remaining operators **M** and **A** perform merging and can improve

Table 5
Summary of results, runtimes (on 3.1 GHz computing cores), and comparison with prior methods. This algorithm was run with the default parameter values listed previously in Table 4, with the exception of the last test instance for genome KT589390, where the operator sequence was MRMSIMOMA (with slightly better outcome) rather than the default RASIMOM. When at least one run obtained a single contig, the number 1 under **Lowest contig count** is marked by ✓ (if the contig was identical to the reference), ✗ (if the contig did not equal the reference), or neither (for some prior literature where the authors did not perform this verification step).

				This algorithm			Other GAs [1,8,9,30,34]	PSO [16]	SSAKE [40]	SOAPdenovo [15]		Velvet [41]
Code	Read length (bp)	Cov.	Reads / library	Lowest contig count	% optimal assembly	Core time	Lowest contig count	Lowest contig count	Lowest contig count	Lowest contig count	% optim. assmb.	Lowest contig count
X60189	400	5	48	1 ✓	**100.0%**	40 s	1[1,9,34]	1 ✓	3	5	–	4
NC001422	400	5	68	1 ✓	**100.0%**	2 min	–	–	3	1 ✓	30%	1 ✗
JQ316129	400	5	114	1 ✓	**100.0%**	7 min	–	–	11	1 ✗	0%	1 ✗
NC009824	400	5	119	1 ✓	**100.0%**	6 min	–	–	9	1 ✓	60%	1 ✗
M15421	400	5	127	1 ✓	**100.0%**	6 min	1[1,9], 6[34]	1 ✓	13	1 ✓	70%	1 ✗
KT589390	400	5	237	1 ✓	**100.0%**	40 min	–	–	18	1 ✓	40%	1 ✗
X60189	100	10	384	1 ✓	**88.5%**	28 min	–	–	4	1 ✗	0%	4
NC001422	100	10	539	1 ✓	**100.0%**	1 h	–	–	3	1 ✗	0%	1 ✗
BX842596	700	5	553	1 ✓	**99.5%**	10 h	1[8], 3[9]	1 ✓	50	2	–	4
BX842596	700	7	773	1 ✓	**99.5%**	18 h	1[8], 2[1,30]	1 ✓	14	2	–	4
J02459	400	7	849	1 ✓	**98.5%**	10 h	–	–	11	1 ✓	40%	1 ✗
JQ316129	100	10	911	1 ✓	**99.0%**	4 h	–	–	11	2	–	1 ✗
NC009824	100	10	946	1 ✓	**99.5%**	5 h	–	–	9	1 ✗	0%	1 ✗
M15421	100	10	1009	1 ✓	**99.0%**	6 h	–	–	9	1 ✗	0%	1 ✗
AP009180	700	7	1596	1 ✓	**89.5%**	125 h	–	–	31	3	–	1 ✗
KT589390	100	10	1896	1 ✓	**87.5%**	40 h	–	–	9	1 ✗	0%	1 ✗

fitness; they are interleaved with the other operators in the sequence, to take advantage of the different orderings on the scaffold. Merge is called twice, once after each group of operators which do not improve fitness.

Four of the six genetic operators were assigned a 50% chance of being triggered. The Align operator is only triggered and may modify the candidate solution in the late stages of the algorithm, when the threshold of 75% holds, i.e., there exists a contig on the candidate scaffold whose length is at least 75% of some *estimated length* of the underlying genome. The Order crossover is triggered a low 10 % of the time.

A Merge of two contigs is performed if the contigs overlap by at least 20 bp (for the shortest raw reads, of length 100 bp) or 30 bp (in all other cases).

The current software implementation of the algorithm is based on the Python library `Inspyred` [10].

4.4. Results

A summary of numerical results is given in Table 5, together with a numerical comparison with related work. To start with, the first four columns define our test instances, i.e., 16 combinations of a genome, a depth of coverage c, and a read length r—for each such combination 10 read libraries are sampled, as described in Section 4.2, and each library is assembled 20 times with the basic algorithm parameters in Table 4.

Given these 200 runs per test instance, Table 5 (columns 5–7) presents the numerical results as follows:

- The column **Lowest contig count** gives the number of contigs obtained on the scaffold of the final assembled genome; an optimal assembly requires this to be 1. For our algorithm, this count is the best obtained among the 200 runs of a test instance.
- Furthermore, only in the cases when a single contig was indeed obtained, the number 1 under **Lowest contig count** is succeeded by either ✓ (if the contig was verified to be identical to the reference genome, i.e., the solution was optimal), ✗ (if the contig did not equal the reference genome), or neither (for some cases from prior literature where the authors did not perform this verification step, and we cannot recover that result).
- The column **% optimal assembly** gives that fraction of the 200 runs where our algorithm obtained an optimal solution (i.e., a single correct contig identical to the reference).
- The **Core time** column lists the rounded average runtime of an assembly on a single, 3.1 GHz computing core. The 200 runs needed to cover each test instance were executed in parallel.

The algorithm consistently reaches the optimal solution, and does so in a vast majority of the runs. The larger the read library, the more difficult the problem is, which is seen in the fact that 100% of the runs for genome **X60189** with 400-bp raw reads and a low coverage of 5 were optimal, but only 88.5% were so for the same genome with shorter reads, medium coverage, and thus read libraries eight times as large.

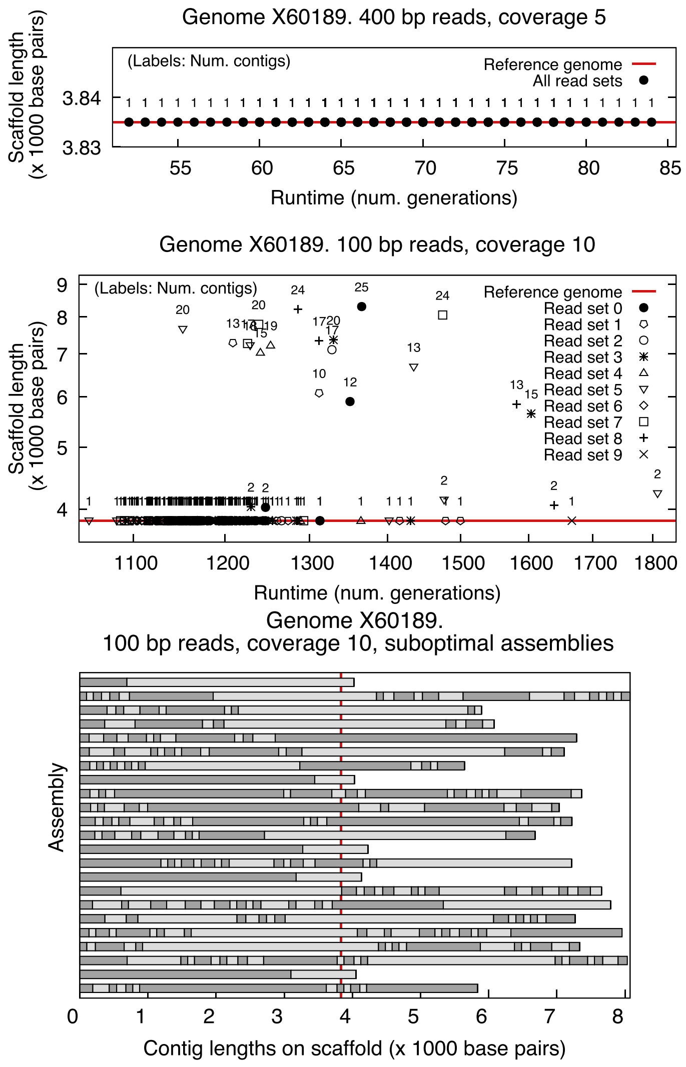

Fig. 5. (top) 200 assemblies for genome **X60189** with 400-bp raw reads and coverage 5, and (center) 200 assemblies for the same genome with 100-bp raw reads and coverage 10. (bottom) contig lengths for all the non-optimal assemblies for the same genome with 100-bp raw reads and coverage 10. In all cases, the length of the reference genome is shown as a red line (gray in print). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fig. 5 (top and center) shows the detailed outcome of all runs for the two test instances for the human genomic fragment **X60189**. For each run, the plots show the scaffold length in base pairs, and the number of contigs of the best solution obtained, together with the number of iterations needed by the algorithm to obtain that solution. The length of the reference genome is shown as a red line (gray in print); the single-contig solutions with that length are all optimal assemblies. For the second test instance, the 11.5% of the runs which had sub-optimal result are distributed among the read libraries, and the lengths of the contigs obtained there are also given in Fig. 5 (bottom); many of these local minima of fitness contain both long, likely mismerged contigs, and chaff contigs which could not be aligned onto long contigs.

The progression of iterations in a single run of the algorithm for the shortest test instance (genome **X60189** with 400-bp raw reads and coverage 5) is given as an example in Fig. 6. Each generation shows the scaffold (i.e., the contig lengths) of the best candidate solution in that generation; the first generation consists of scaffolds in which each contig is a single raw read. It is to note that only in the later generations the algorithm forms long contigs, and the convergence on a minimum is fast after that point.

For all test instances in Table 5, a large majority of the runs have located the global minimum. Fig. 7 gives more summary plots for the 200 runs of three test instances: the phage λ genome **NC001422** with short raw reads and medium coverage, and the human genomic fragment **M15421** and the HIV-1 strain **JQ316129** with medium-length raw reads and low coverage, with all of the runs optimal, despite the difference in problem difficulty due to the different sizes of read libraries.

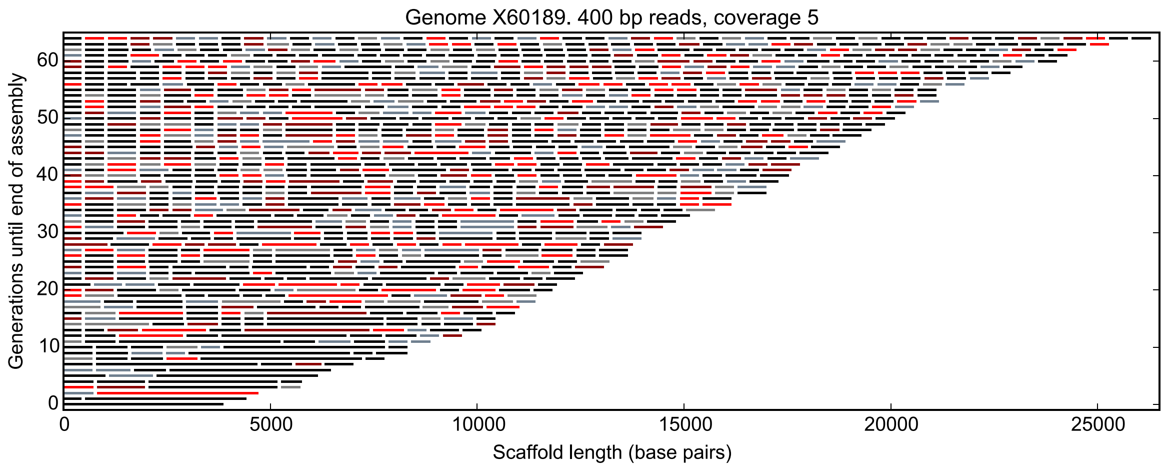

Fig. 6. The progression of generations in an assembly run for genome **X60189** with 400-bp raw reads and coverage 5. For each generation, the plot shows the contig lengths of the best solution of the generation; the colors assigned to contigs are random here, for ease of differentiation.

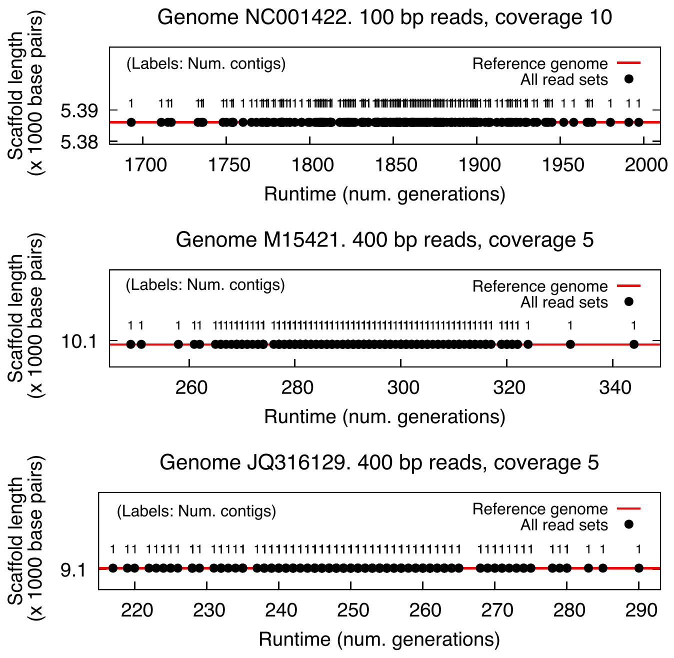

Fig. 7. 200 assemblies for genomes **NC001422** (with 100-bp reads and coverage 10), **M15421** and **JQ316129** (with 400-bp reads and coverage 5).

4.5. Comparison with existing methods

The rightmost columns in Table 5 list the **Lowest contig count** evaluation metric for:

- Various prior tools built around a genetic algorithm, and one prior tool [16] based on particle swarm optimization, with these results extracted from their respective publications, when they cover a test instance similar to ours.
- The short-read assembler SSAKE [40], which implements a greedy heuristic based on the amount of overlap in order to assemble the raw reads.
- The short-read assembler SOAPdenovo [15], which builds a graph over k-long genomic substrings (*kmers*) where edges model overlap (k is usually a relatively small integer), and searches for an Eulerian path to approximate a Hamiltonian path. SOAPdenovo performed well on bacterial genomes in a comparative study [37].
- The Velvet assembler [41], also based on graphs of short kmers, showed good genome contiguity in [6] when used with the long-read option, which attempts to heuristically solve any short repeats introduced by the genome fragmentation into kmers.

The fungal genomic fragment **BX842596** (of length 77 kbp, with 700-bp raw reads and coverage 7, i.e., 700–800 reads in a read library) is the largest assembly job for which a prior GA obtained a single contig [8]. NB: the related work [1,8,16] assembled **BX842596** with coverage 4 instead of our 5, but Table 5 lists that result nevertheless. Also, the full phage λ genome **J02459** was also assembled in [9], but using a smaller read library of longer, 700-bp reads, so it is not listed in the table; this work obtained a single contig, but did not verify its biological correctness.

The related work using genetic algorithms made little attempt to also evaluate the correctness of the single-contig genome obtained, against a reference or by using consensus with different methods; since different studies obtained, for single-contig assemblies of the same genome, different values for the same fitness function (measuring the total amount of overlap on the scaffold), it is unlikely that many of the single-contig solutions are biologically accurate. In contrast, the accuracy evaluation is done here for all our results.

SSAKE is run, on each read library, with the arguments `-w 10 -m 20`, which state a minimum amount of overlap of 20 bp, and aims for a depth of coverage of 10 for any contigs on the scaffold. The contigs found by SSAKE (Table 5) include chaff (i.e., single reads which the algorithm could not align onto long contigs). In its defense, SSAKE is known to have better performance with the deeper coverage 20; it obtained one contig which was verified to align 99.92% to the reference genome for the ΦX174 (**NC001422**) viral genome. The SSAKE runs showed, besides a clear lack of contiguity in the assemblies, a very large variability in the contig count obtained by SSAKE for the same genome, when assembling different raw-read libraries.

SOAPdenovo was run, on each read library, with `-R` to attempt to solve repeats heuristically (which yields fewer contigs than without this option), the kmer length `-K 19` for short raw reads (of 100 bp) or `-K 29` (in the other cases), `asm_flags=3` and `map_len=20`. Occasionally, the executable aborted with an error message, and another kmer size had to be chosen between 19 and 33 for which the run would complete. The tool was run once for each of our 10 read libraries. The results for SOAPdenovo are given in two columns, similarly to the columns for our algorithm. In no case the tool obtained the correct assembly across all the read libraries of a genome; Table 5 notes a success when at least one read library yielded the correct assembly. SOAPdenovo often obtained single contigs on the scaffold, yet more often than not these assemblies are not identical to the reference, and are slightly shorter in length, i.e., overcompressed: for example, for the test instance **NC001422** (of reference length 5386 bp) with 100-bp reads, SOAPdenovo obtained 10 contiguous genomes of 10 different lengths: 5383, 5374, 5351, 5360, 5357, 5377, 5382, 5370, 5371, and 5381 bp. Like for SSAKE, it is possible that SOAPdenovo is more accurate over read libraries with deeper genome coverage.

Velvet was run with the kmer length 31 and options `-long -exp_cov` (also to attempt to solve repeats, which also in this case improved the contiguity of the assembly). Velvet showed considerable consistency when run over different read libraries obtained from the same genome, in terms of contiguity: in most test cases, the number of contigs obtained was constant across libraries. However, for these test genomes, where no long (but only very short) repeats exist, it appears overly compressive: when assembling 10 different read libraries of the same test instance **NC001422** (reference length 5386 bp) with 100-bp reads, Velvet obtained 8 contiguous genomes of 6 different lengths: 5344, 5342, 5356, 5353, 5338, 5341 bp. In comparison with SOAPdenovo, these genomes show more overcompression. As a note, in a study focused solely on assembly algorithms for genomes with long repeats, Velvet far outperformed SOAPdenovo in terms of genome contiguity, which shows that the two assemblers have different strengths.

4.6. Variation in genetic operators and numerical parameters

The results obtained by our tool in Table 5 used the default parametrization of the GA, listed previously in Table 2, except for the test instance in the last row (the Ebola genome **KT589390**, raw read length 100 bp, which is the largest test instance in this study). For this instance, Table 5 lists the results obtained with a different order of application of genetic operators: MRMSIMOMA rather than the default RASIMOM (all other parameters remained the same). MRMSIMOMA essentially attempts twice as much merging as RASIMOM, and yielded optimal assemblies in 87.5% of the 200 runs, while the default RASIMOM setting obtained a slightly lower 78.5% rate of optimality. We hypothesize that more merging of contigs in the early stages of the algorithm reduces the size of the problem left to solve by later stages, and brings a slight advantage. On the other hand, the sequence MRMSIMOMA is more expensive computationally: in our current implementation of the tool, the sequence comes with a roughly 30% increase in the runtime of an assembly.

The results all used a Random Merge operator. We can't conclude on the effectiveness of the Greedy Merge operator: on some test instances, it had a slightly better or slightly worse optimality rate; in some cases, it showed a slight decrease in runtime.

The Align operator plays a crucial role. Fig. 8 (top) shows the scaffold length and number of contigs obtained in the best solution, each for a single raw read library pertaining to a test instance, and each with and without the Align operator. While the RASIMOM operator sequence computed the optimal assembly in all the runs, the runs with the RSIMOM sequence stagnated on a low-quality scaffold with many contigs. All the scaffolds computed in RSIMOM runs are shown in Fig. 8 (bottom), where it becomes apparent that the absence of Align yields many chaff contigs—a fact noted before in [5]. However, almost all the runs constructed a single long contig which is identical to the reference genome; it is this fact which allows Align to trigger in the later iterations and refine the scaffold into the optimal solution.

For genomes without long repeats, using an Align threshold lower than the default 0.75 could be more beneficial in terms of the percentage of optimal runs, if slightly more expensive in runtime. With a threshold of 0.5, on the **NC001422**

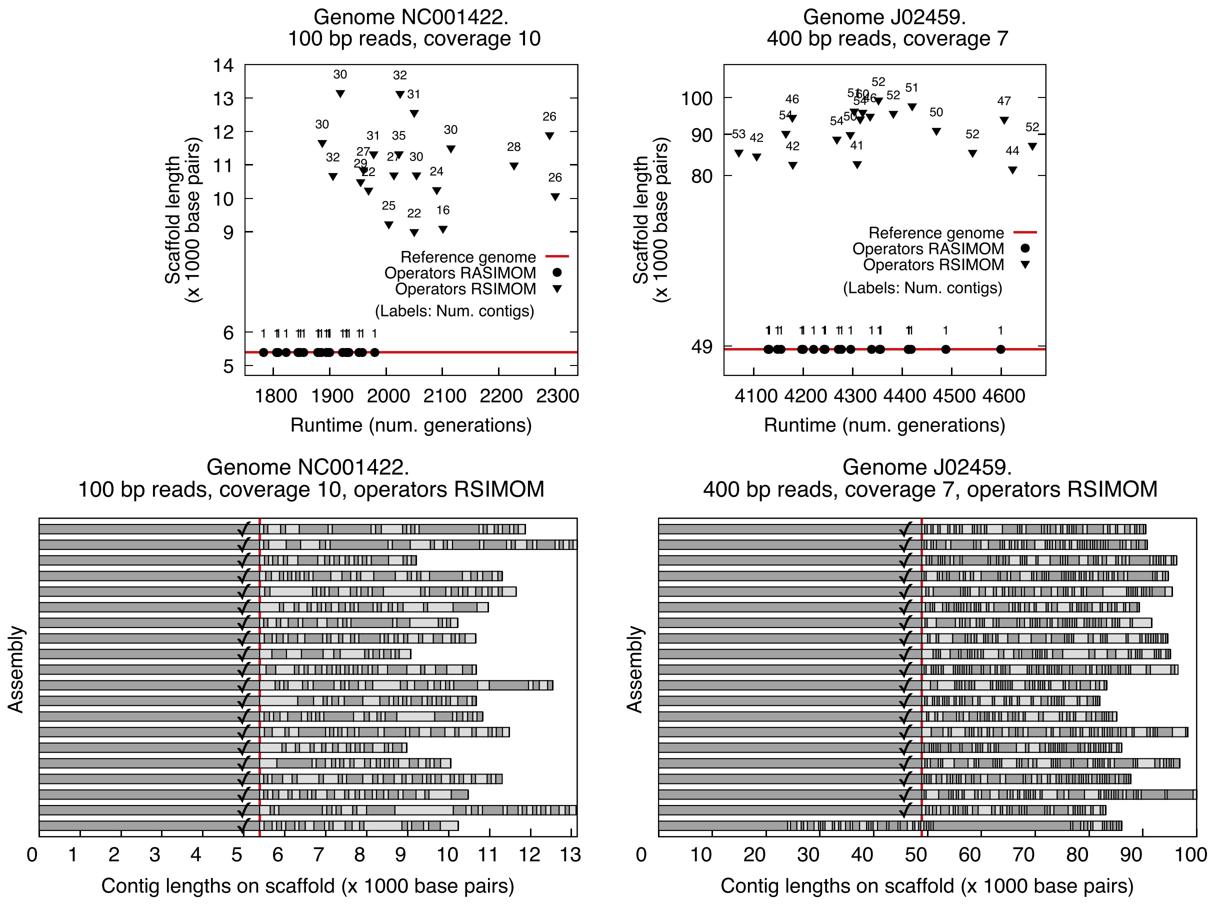

Fig. 8. (top) 20 assemblies with and 20 assemblies without the Align operator (over the same raw read library) for genome NC001422 with 100-bp raw reads and coverage 10, and for genome J02459 with 400-bp raw reads and coverage 7. (bottom) contig lengths for all the assemblies performed without the Align operator, for the same genomes; a long contig which is identical to the reference genome is marked with a $\surd$. In all cases, the length of the reference genome is shown as a red line (gray in print). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

genome with 100-bp reads, the same rate of optimality was seen among a sample of runs; on **KT589390** with 100-bp reads and the default RASIMOM operator order, there was a marginally better rate of optimality, as per Fig. 9.

4.7. Compressed assemblies due to limitation

When using our method, the symptoms which indicate that the underlying genome has long repeats or stretches of identical base pairs consist of the inability of the algorithm to reach a consensus across read libraries. For example, assembling 10 read libraries sampled from another isolate of the HIV-1 virus, **JQ316128** [25] (not present in Table 5), using 400-bp reads, yields the numerical results in Fig. 10. While any one read library is seen there to give relatively consistent single-contig solutions across different runs of the assembler, the assembly results vary in length across read libraries, and all are overcompressed when compared to the reference genome. The cause is one of the features of the genome which limit this algorithm, due to our choice of fitness (Section 3.3): the genome has identical 142-bp end sequences, which the algorithm then overlaps to minimize the scaffold length.

4.8. High runtimes, low memory use

The method has low memory complexity: any worst-case auxiliary memory needed for representing the 100 candidate solutions in a generation of the algorithm is of the same order of growth as the input memory, i.e., the raw read library. On the other hand, the runtime of the algorithm cannot be easily predicted, and will grow with both the size of the read library and the length of the reads. An assembly run for the first test instance in Table 5 required 40 s, and the last test instance 40 h (on a 3.1 MHz computing core); the most computationally expensive test case was the **AP009180** genome (second to last in the table), at 125 core-hours. The current implementation uses Python, an interpreted rather than compiled program-

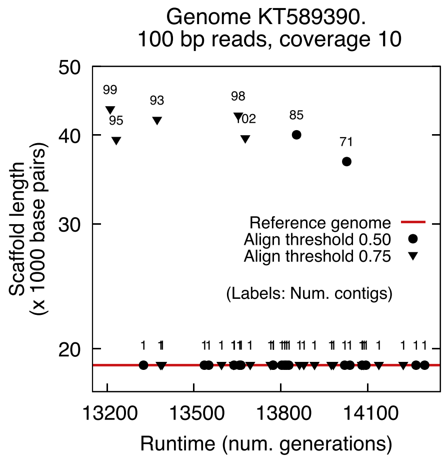

Fig. 9. Different thresholds for the Align operator bring a slight advantage. 20 assemblies of the same read library with the default 0.75 and 20 assemblies with the lower 0.5 threshold. The length of the reference genome is shown as a red line (gray in print). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

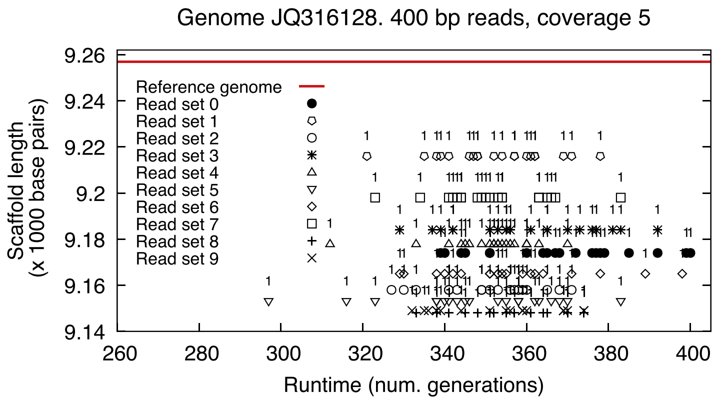

Fig. 10. 200 assemblies for the **JQ316128** genome, over 10 read libraries. Due to a long prefix identical to the suffix of the genome, the assemblies are overcompressed. The length of the reference genome is shown as a red line (gray in print). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

ming language, so speedups can be expected in future versions; as reflected in the high rate of optimal assemblies with the algorithm, the bottleneck in performance in this version is the runtime, rather than the accuracy of the algorithm.

5. Conclusions

This work presented a stochastic metaheuristic which approximates the solution to the problem of DNA-fragment assembly, under the assumption of accurately sequenced fragments and genomes without certain repeat patterns. It was shown empirically that, for genomes that are virus-sized, the method is accurate, with a 85 % chance of obtaining the correct genome for the most difficult test instance, a 159,662-bp complete endosymbiont genome with 1596 DNA fragments per library. This compares favourably with other assemblers, which show either less scalability with the size of the input, or a high likelihood of computing inaccurate, overcompressed assemblies.

Given the high likelihood of our method to compute the optimal solution, it can serve by itself as a means of computing the consensus genome for a de-novo assembly project; in case of genomes with repeats, assembling different libraries may make this apparent by not obtaining consensus across the libraries. The drawback of our method is its higher time complexity. Future work will attempt to remove the assumption of perfect sequencing accuracy, improve time complexity, and cater for genomes with long repeated sequences.

References

[1] E. Alba, G. Luque, A new local search algorithm for the DNA fragment assembly problem, in: Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, in: EvoCOP'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 1–12. http://dl.acm.org/citation.cfm?id=1761927.176192.
[2] M. Baker, De novo genome assembly: what every biologist should know, Nat. Meth. 9 (4) (2012) 333–337. http://dx.doi.org/10.1038/nmeth.1935.

[3] R. Baos, F. Manzano-Agugliaro, F. Montoya, C. Gil, A. Alcayde, J. Gmez, Optimization methods applied to renewable and sustainable energy: a review, Renew. Sustain. Energy Rev. 15 (4) (2011) 1753–1766, doi:10.1016/j.rser.2010.12.008.
[4] K.R. Bradnam, et al., Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species, GigaScience 2 (1) (2013) 10, doi:10.1186/2047-217X-2-10.
[5] D. Bucur, De Novo DNA Assembly with a Genetic Algorithm Finds Accurate Genomes Even with Suboptimal Fitness, Springer International Publishing, 2017, pp. 67–82. http://dx.doi.org/10.1007/978-3-319-55849-3_5.
[6] D. Bucur, Towards accurate de novo assembly for genomes with repeats, in: 2017 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (accepted), 2017, pp. 1–8.
[7] Y. Cherukuri, S.C. Janga, Benchmarking of de novo assembly algorithms for nanopore data reveals optimal performance of olc approaches, BMC Genom. 17 (7) (2016) 507, doi:10.1186/s12864-016-2895-8.
[8] B. Dorronsoro, E. Alba, G. Luque, P. Bouvry, A self-adaptive cellular memetic algorithm for the DNA fragment assembly problem, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), 2008, pp. 2651–2658, doi:10.1109/CEC.2008.4631154.
[9] J.S. Firoz, M.S. Rahman, T.K. Saha, Bee algorithms for solving DNA fragment assembly problem with noisy and noiseless data, in: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, in: GECCO '12, ACM, New York, NY, USA, 2012, pp. 201–208, doi:10.1145/2330163.2330192.
[10] A.L. Garret, Inspyred: A framework for creating bio-inspired computational intelligence algorithms in Python, 2017, https://pypi.python.org/pypi/inspyred.
[11] J. Hughes, S. Houghten, G.M. Malln-Fullerton, D. Ashlock, Recentering and restarting genetic algorithm variations for dna fragment assembly, in: 2014 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology, 2014, pp. 1–8, doi:10.1109/CIBCB.2014.6845500.
[12] J.A. Hughes, S. Houghten, D. Ashlock, Restarting and recentering genetic algorithm variations for {DNA} fragment assembly: the necessity of a multi-strategy approach, Biosystems 150 (2016) 35–45, doi:10.1016/j.biosystems.2016.08.001.
[13] Illumina, Illumina sequencing technology, 2016, http://www.illumina.com/documents/products/techspotlights/techspotlight_sequencing.pdf.
[14] L. Li, S. Khuri, A comparison of DNA fragment assembly algorithms, in: Proc. of the Intl Conf. on Mathematics and Engineering Techniques in Medicine and Biological Sciences, CSREA Press, 2004, pp. 329–335.
[15] R. Luo, et al., SOAPdenovo2: An empirically improved memory-efficient short-read de novo assembler, Gigascience 1 (1) (2012) 18, doi:10.1186/2047-217X-1-18.
[16] G.M. Mallén-Fullerton, G. Fernández-Anaya, DNA fragment assembly using optimization, in: 2013 IEEE Congress on Evolutionary Computation, 2013, pp. 1570–1577, doi:10.1109/CEC.2013.6557749.
[17] G.M. Mallén-Fullerton, J.E. Quiroz-Ibarra, A. Miranda, G. Fernández-Anaya, Modified classical graph algorithms for the DNA fragment assembly problem, Algorithms 8 (2015) 754–773.
[18] G. Minetti, G. Leguizamón, E. Alba, An improved trajectory-based hybrid metaheuristic applied to the noisy DNA fragment assembly problem, Inf. Sci. 277 (2014) 273–283. http://dx.doi.org/10.1016/j.ins.2014.02.020
[19] S. Nasser, G.L. Vert, M. Nicolescu, A. Murray, Multiple sequence alignment using fuzzy logic, in: 2007 IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology, 2007, pp. 304–311, doi:10.1109/CIBCB.2007.4221237.
[20] NCBI, Candidatus Carsonella ruddii PV DNA, complete genome, 2017a, https://www.ncbi.nlm.nih.gov/nuccore/AP009180.1.
[21] NCBI, Enterobacteria phage lambda, complete genome, 2017b, https://www.ncbi.nlm.nih.gov/nuccore/J02459.
[22] NCBI, Enterobacteria phage phiX174 sensu lato, complete genome, 2017c, https://www.ncbi.nlm.nih.gov/nuccore/NC_001422.1.
[23] NCBI, Hepatitis C virus genotype 3, genome, 2017d, https://www.ncbi.nlm.nih.gov/nuccore/NC_009824.1.
[24] NCBI, HIV-1 isolate 99HYH2, complete genome, 2017e, https://www.ncbi.nlm.nih.gov/nuccore/JQ316129.1.
[25] NCBI, HIV-1 isolate HP-9:03KDE11, complete genome, 2017f, https://www.ncbi.nlm.nih.gov/nuccore/JQ316128.1.
[26] NCBI, Human apolipoprotein B-100 mRNA, complete cds, 2017g, https://www.ncbi.nlm.nih.gov/nuccore/M15421.
[27] NCBI, Human MHC class III region DNA with fibronectin type-III repeats, 2017h, https://www.ncbi.nlm.nih.gov/nuccore/X60189.
[28] NCBI, Neurospora crassa DNA linkage group II BAC clone B10K17, 2017i, https://www.ncbi.nlm.nih.gov/nuccore/BX842596.
[29] NCBI, Zaire ebolavirus isolate Ebola virus/H.sapiens-wt/SLE/2014/Makona-201403164, complete genome, 2017j, https://www.ncbi.nlm.nih.gov/nuccore/KT589390.1.
[30] A. Nebro, G. Luque, F. Luna, E. Alba, DNA Fragment assembly using a grid-based genetic algorithm, Comput. Oper. Res. 35 (9) (2008) 2776–2790, doi:10.1016/j.cor.2006.12.011. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization.
[31] R. Parsons, S. Forrest, C. Burks, Genetic algorithms for DNA sequence assembly, in: Proceedings. International Conference on Intelligent Systems for Molecular Biology, 1, 1993, pp. 310–318. http://europepmc.org/abstract/MED/7584352.
[32] R. Parsons, M.E. Johnson, DNA sequence assembly and genetic algorithms - new results and puzzling insights, in: Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, Cambridge, United Kingdom, July 16–19, 1995, 1995, pp. 277–284.
[33] R. Parsons, M.E. Johnson, A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly, Am. J. Math. Manage. Sci. 17 (3–4) (1997) 369–396, doi:10.1080/01966324.1997.10737444.
[34] R.J. Parsons, S. Forrest, C. Burks, Genetic algorithms, operators, and DNA fragment assembly, Mach. Learn. 21 (1–2) (1995) 11–33, doi:10.1007/BF00993377.
[35] W. Paszkowicz, Genetic algorithms, a nature-inspired tool: survey of applications in materials science and related fields, Mater. Manuf. Processes 24 (2) (2009) 174–197, doi:10.1080/10426910802612270.
[36] K.-J. Räihä, E. Ukkonen, The shortest common supersequence problem over binary alphabet is NP-complete, Theor. Comput. Sci. 16 (2) (1981) 187–198, doi:10.1016/0304-3975(81)90075-X.
[37] S.L. Salzberg, A.M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T.J. Treangen, M.C. Schatz, A.L. Delcher, M. Roberts, G. Marçais, M. Pop, J.A. Yorke, GAGE: A critical evaluation of genome assemblies and assembly algorithms, Genome Res. 22 (3) (2012) 557–567, doi:10.1101/gr.131383.111.
[38] F. Sanger, G.M. Air, B.G. Barrell, N.L. Brown, A.R. Coulson, J.C. Fiddes, C.A. Hutchison, P.M. Slocombe, M. Smith, Nucleotide sequence of bacteriophage PhiX174 DNA, Nature 265 (1977) 687–695, doi:10.1038/265687a0.
[39] R.H. Sheikh, M.M. Raghuwanshi, A.N. Jaiswal, Genetic algorithm based clustering: a survey, in: 2008 First International Conference on Emerging Trends in Engineering and Technology, 2008, pp. 314–319, doi:10.1109/ICETET.2008.48.
[40] R.L. Warren, G.G. Sutton, S.J.M. Jones, R.A. Holt, Assembling millions of short DNA sequences using SSAKE, Bioinformatics 23 (4) (2007) 500–501, doi:10.1093/bioinformatics/btl629.
[41] D.R. Zerbino, G.K. McEwen, E.H. Margulies, E. Birney, Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler, PLoS ONE 4 (12) (2009) 1–9, doi:10.1371/journal.pone.0008407.

Doina Bucur works on algorithms for computationally hard problems. She received a Ph.D. in Computer Science from University of Aarhus, Denmark (2008). Between 2008 and 2010, she held a postdoctoral position with the Computing Laboratory at University of Oxford, UK, working on formal verification for networked embedded systems. Between 2010 and 2012, she was a scientific researcher at INCAS[3], a research institute in The Netherlands. From 2012, she is an assistant professor in Computer Science, first with University of Groningen, and then with University of Twente, The Netherlands.