Published online 20 August 2021 *Nucleic Acids Research, 2021, Vol. 49, No. 20* ***e117***
https://doi.org/10.1093/nar/gkab717

SWALO: scaffolding with assembly likelihood optimization

Atif Rahman[1,3,*] and Lior Pachter[1,2,4,*]

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, USA, [2]Departments of Mathematics and Molecular & Cell Biology, University of California, Berkeley, CA 94720, USA, [3]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1205, Bangladesh and [4]Departments of Biology and Computing & Mathematical Sciences, California Institute of Technology, Pasadena, CA 91103, USA

Received December 22, 2020; Revised June 16, 2021; Editorial Decision August 05, 2021; Accepted August 16, 2021

ABSTRACT

Scaffolding, i.e. ordering and orienting contigs is an important step in genome assembly. We present a method for scaffolding using second generation sequencing reads based on likelihoods of genome assemblies. A generative model for sequencing is used to obtain maximum likelihood estimates of gaps between contigs and to estimate whether linking contigs into scaffolds would lead to an increase in the likelihood of the assembly. We then link contigs if they can be unambiguously joined or if the corresponding increase in likelihood is substantially greater than that of other possible joins of those contigs. The method is implemented in a tool called SWALO with approximations to make it efficient and applicable to large datasets. Analysis on real and simulated datasets reveals that it consistently makes more or similar number of correct joins as other scaffolders while linking very few contigs incorrectly, thus outperforming other scaffolders and demonstrating that substantial improvement in genome assembly may be achieved through the use of statistical models. SWALO is freely available for download at https://atifrahman.github.io/SWALO/.

INTRODUCTION

The development of second generation sequencing technologies (1–4) has led to development of various assays to probe many aspects of interest in molecular and cell biology due to the low cost and high throughput. However, a prerequisite for running many of these assays, genome assembly is yet to be solved adequately using second generation sequencing reads. Long reads from the third generation sequencing technologies such as single molecule real time (SMRT) (5) and nanopore sequencing (6) are transforming genome assembly, and a number of tools have been developed to assemble genomes from long reads (7–10). However, the high cost, low sequencing coverage and high error rates of these technologies mean that many of the genomes are being assembled using reads from second generation technologies, or through a combination of second and third generation data. In addition, an extensive amount of second generation data exists already that can be better utilized. Furthermore, reference guided assembly tools such as RACA (11), Ragout (12) and MeDuSa (13) require an initial draft assembly which is often generated using second generation reads. In this paper, we demonstrate that considerable improvement in genome assembly using second-generation sequencing can be achieved through the application of statistical models for sequencing. The statistical approach we introduce may also be applied to genome assembly using long reads.

Genome assembly typically consists of two major steps. The first step is to merge overlapping reads into contigs which is commonly done using the *de Bruijn* or overlap graphs. In the second step, known as 'scaffolding', described in (14–16), contigs are oriented and ordered using various approaches such as paired-end or mate-pair reads (we use the term read pair to refer to either). Recently, methods have been developed to perform scaffolding using linked reads (17–19), long reads (20–23), and chromosomal conformation data from Hi-C (24–29) that lead to scaffolds with substantially better contiguity compared to scaffolds generated using read-pairs, and hence are the recommended technologies for scaffolding. However, read-pair information from second generation technologies is still widely used for scaffolding (30) and remains a critical part of the genome assembly process. It is hence built into most assemblers (31–36) and a number of stand-alone scaffolders such as Bambus2 (37,38), MIP (39), Opera (40,41), SCARPA (42), SOPRA (43), SSPACE (44), BESST (45) have also been developed to better utilize read-pair information and to resolve ambiguities due to repetitive regions.

*To whom correspondence should be addressed. Tel: +88 01752 872161; Email: atif@cse.buet.ac.bd
Correspondence may also be addressed to Lior Pachter. Tel: +1 626 395 4059; Email: lpachter@caltech.edu

Most of the scaffolding algorithms rely on heuristics or user input to determine parameters such as minimum number of read-pairs linking contigs to join them ignoring contig lengths, sequencing depth and sequencing errors. In an in-depth study, Hunt *et al.* evaluated scaffolding tools on real and simulated data and observed that although many of the scaffolders perform well on simulated datasets, they show inconsistent performance across real datasets and mapping tools (46). Their results demonstrate that SGA, SOPRA and ABySS are conservative and make very few scaffolding errors while SOAPdenovo identified more joins at the expense of greater number of errors indicating a scaffolding method achieving a better trade-off of the two may be possible.

Here, we present a scaffolding method called 'scaffolding with assembly likelihood optimization (SWALO)'. SWALO learns parameters automatically from the data and is largely free of user parameters making it more consistent than other scaffolders. It is also able to make use of multi-mapped read pairs through probabilistic disambiguation which most other scaffolding tools ignore. The method is grounded in rigorous probabilistic models yet proper approximations make the implementation efficient and applicable to practical datasets. We analyze the performance of SWALO using datasets used by Hunt *et al.* and find that SWALO makes more or similar number of correct joins as other scaffolders while making very few incorrect joins. We also compare SWALO with scaffolding modules built into various assemblers using GAGE datasets (47) and observe that final results obtained by applying SWALO on contigs generated by assemblers are generally better than applying the in-built scaffolding modules of those assemblers. Finally, we apply SWALO to a large dataset from the budgerigar genome and demonstrate that it scales to large datasets without compromising performance.

MATERIALS AND METHODS

Overview of SWALO

Our scaffolding method called SWALO is based on a generative model for sequencing (48). Figure 1 illustrates the main steps of SWALO. In the first step, reads are aligned to contigs, the insert size distribution and error parameters are learned using the reads that map uniquely and the likelihood of the set of contigs is computed using a generative model. We then construct the bi-directed *scaffold graph* which contains a vertex for each contig and there is an edge between contigs if joining them would result in an increase in the likelihood. It uses probabilistic models to estimate maximum likelihood gaps between contigs correcting for the issue that we may not observe inserts from the entire distribution of insert sizes due to gaps between contigs and lengths of contigs (Supplementary Figure S2) (49,50). It then approximates whether joining contigs would result in an increase in the genome assembly likelihood. We use the EM (expectation maximization) (51,52) algorithm to resolve multi-mapped read pairs. Contigs are then joined if the increase in the likelihood is significantly higher than that of all other conflicting joins as determined by a heuristic. Thus, SWALO takes a step towards maximum likelihood genome assembly (53). Moreover, we select multiple joins consistent with one another using the dynamic programming algorithm for the weighted interval scheduling problem. Each of these steps is described in more detail in the sections below and in Supplementary Notes 3.1-3.3. Our scaffolding method (i) learns parameters from the data making it largely parameter free (Supplementary Note 3.4), (ii) makes use of multi-mapped read pairs that are ignored in most scaffolders and (iii) is able to accurately estimate gaps between contigs facilitating gap-filling.

We present here a brief description of methods behind SWALO. More details is available in Supplementary Notes 3.1-3.4.

Learning parameters and computing likelihood of contigs

The first step in SWALO is to estimate parameters and compute likelihood of contigs using the approach presented in (48) (Supplementary Note 3.1). The model incorporates insert size distribution, sequencing errors as well as randomness in read generation. We map reads to contigs and learn insert size distribution and error parameters using reads that map uniquely. The likelihood of contigs are computed with respect to a generative model using the learned parameters.

Given an assembly $\mathcal{A}$ and a set of reads pairs $\mathcal{R} = \{r_1, r_2 \ldots r_N\}$, assuming reads are generated independently the *likelihood* is given by

$$L(\mathcal{A}; \mathcal{R}) = \prod_{i=1}^{N} p(r_i|\mathcal{A})$$

If M_i is the number of times the read r_i is mapped to the assembly by an aligner, the probability that the read r_i is generated from the assembly is approximated by

$$p(r_i|\mathcal{A}) \approx \sum_{j=1}^{M_i} p_F(l_{i,j}) p_S(s_{i,j}) p_E(r_i|a_{i,j})$$

where $l_{i,j}$, $s_{i,j}$ and $a_{i,j}$ are fragment length, start site and assembly subsequence corresponding to jth mapping of ith read respectively. Therefore, the log likelihood is given by

$$l(\mathcal{A}; \mathcal{R}) \approx \sum_{i=1}^{N} \log \sum_{j=1}^{M_i} p_F(l_{i,j}) p_S(s_{i,j}) p_E(r_i|a_{i,j}). \qquad (1)$$

Here, p_F, p_S and p_E are fragment length, start site and error distributions respectively, as in (48). However, we use a smoothed and truncated version of the insert size distribution for scaffolding (Supplementary Figure S1, Supplementary Note 3.1). The probability that an insert of length l starts at s is

$$p_S(s) = \frac{1}{\tilde{T}(l)} = \frac{1}{\sum_{c \in \{contigs\}} (l_c - l + 1)}$$

where $\tilde{T}(l)$ is the total effective length, i.e. number of possible start sites for insert size l and l_c is the length of contig c.

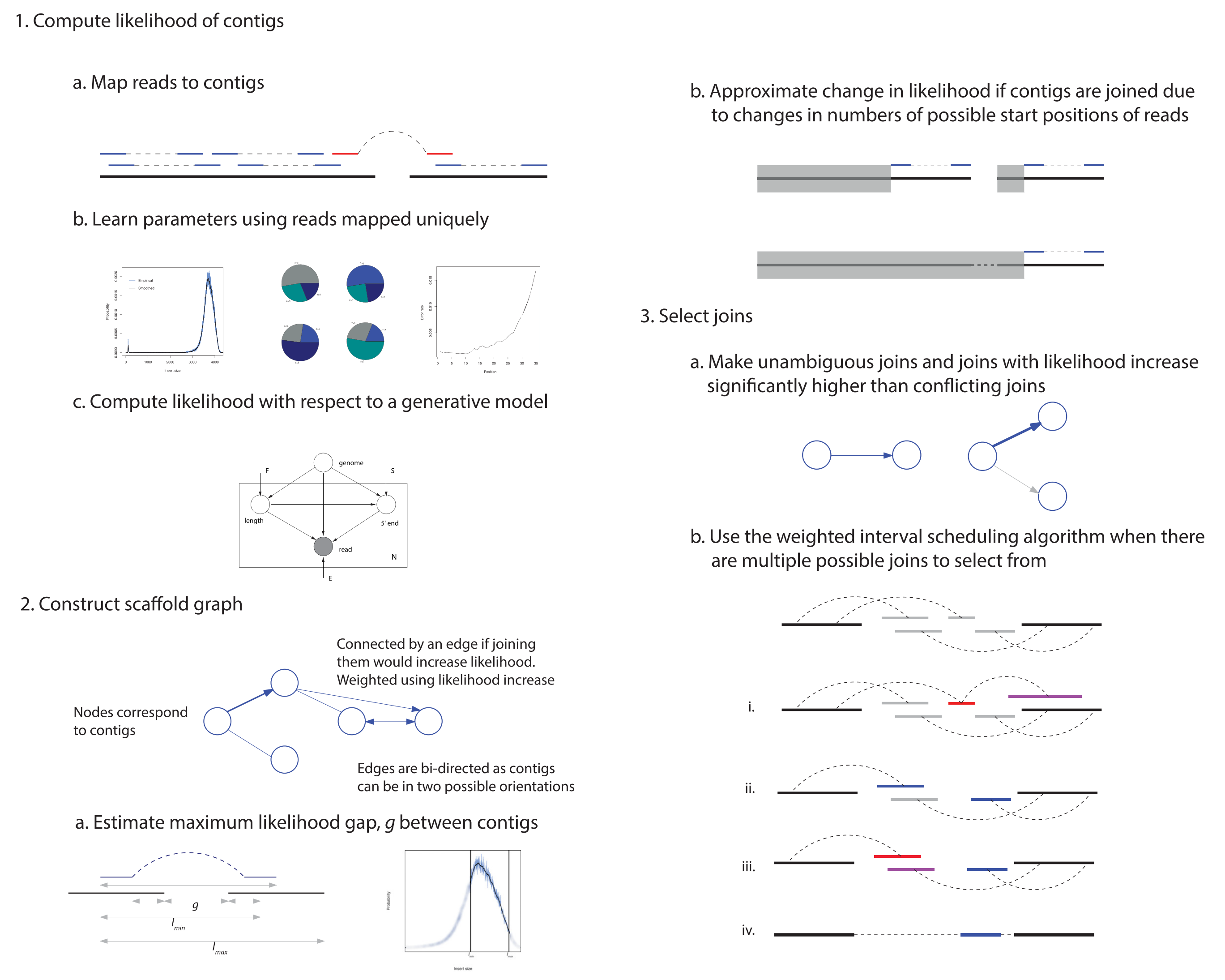

1. Compute likelihood of contigs
a. Map reads to contigs
b. Learn parameters using reads mapped uniquely
c. Compute likelihood with respect to a generative model
genome
F
S
length
5' end
read
N
E
2. Construct scaffold graph
Connected by an edge if joining them would increase likelihood. Weighted using likelihood increase
Nodes correspond to contigs
Edges are bi-directed as contigs can be in two possible orientations
a. Estimate maximum likelihood gap, g between contigs
g
l_{min}
l_{max}
b. Approximate change in likelihood if contigs are joined due to changes in numbers of possible start positions of reads
3. Select joins
a. Make unambiguous joins and joins with likelihood increase significantly higher than conflicting joins
b. Use the weighted interval scheduling algorithm when there are multiple possible joins to select from
i.
ii.
iii.
iv.

Figure 1. Overview of SWALO. 1. Reads are aligned to contigs, uniquely mapped reads are used to learn the insert size distribution and error parameters, and then the likelihood of the set of contigs is computed (Supplementary Note 3.1, Supplementary Figure S1). 2. The scaffold graph is constructed by first estimating maximum likelihood gaps, *g* between contigs using the EM algorithm to resolve multimapped read-pairs taking into account that only inserts of sizes between l_{min} and l_{max} will be observed due to gaps between contigs and lengths of contigs (Supplementary Figure S2), and then approximating whether changes in number of possible start sites of reads (the regions shaded in grey) lead to an increase or decrease in the assembly likelihood (Supplementary Note 3.2). 3. Finally, we make the joins that are unambiguous or correspond to likelihood increase significantly higher than that of other conflicting joins. If there are multiple contigs (grey) that fit into the gap between contigs being joined we select from them using the dynamic programming algorithm for the weighted interval scheduling problem in following steps. i. Remove contigs (red) with inconsistent edges to other contigs. ii. Select consistent set of contigs (blue) that optimizes likelihood. iii. Remove selected contigs (red) with likelihood increase not significantly higher than conflicting ones not selected (purple). iv. Merge them into scaffolds (Supplementary Note 3.3).

Scaffold graph construction

We then construct the *scaffold graph* which is a bi-directed graph with a vertex for each contig and there is an edge between contigs if joining the contigs would lead to an increase in the assembly likelihood (Supplementary Note 3.2). The edges are weighted using this increase in likelihood. Edge weights are computed for each pair of contigs such that there are read pairs with two ends mapping to different contigs in the pair. This is done in two steps.

- First we estimate the maximum likelihood gap between the contigs using a generative model correcting for the issue that we may not observe inserts from the entire distribution (Supplementary Figure S2).
  Consider two contigs separated by a gap g. If the $5'$ end of an insert is at s, then we will not observe inserts smaller than l_{min} and greater than l_{max}. The probability that a read r is generated is then given by

$$p(r) \approx \sum p'_S(s)p'_F(l)p_E(r|a)$$

where p'_F and p'_S are the corrected insert size and start site distributions respectively. p'_F is given by

$$p'_F(l) = \frac{p_F(l)}{\sum_{k=l_{min}}^{l_{max}} p_F(k)}$$

and p'_S is approximated by

$$p'_S(s) \approx \frac{\text{p\{fragment starting at s and ending at t\}}}{\text{p\{fragment starting in A and ending at t\}}}.$$

We then find the gap g that maximizes likelihood of linking reads, $\max_g \prod_{r \in \{linking\}} p(r)$ for every pair of contigs with read pairs linking them.
If there are read pairs that map to multiple pairs of contigs, we resolve them using the expectation-maximization (EM) (51) algorithm.

- Then we check whether linking the contigs would result in an increase in the likelihood by computing probability of linking reads and adjusting probabilities of all other reads.
Adjusting the probabilities because of the change in effective length using Equation (1) would require iterating over all multi-mapped reads. In order to make this step practical we make the following approximation.

$$\begin{aligned} l(\mathcal{A};\mathcal{R}) &\approx \sum_{i=1}^{N} \log \frac{1}{\tilde{T}(\hat{l}_i)} \sum_{j=1}^{M_i} p_F(l_{i,j}) p_E(r_i|a_{i,j}) \\ &= \sum_{i=1}^{N} \log \frac{1}{\tilde{T}(\hat{l}_i)} \\ &\quad + \sum_{i=1}^{N} \log \sum_{j=1}^{M_i} p_F(l_{i,j}) p_E(r_i|a_{i,j}) \end{aligned}$$

where $\hat{l}_i$ is insert size corresponding to most probable mapping of read i. This allows us to count the number of reads corresponding to a particular insert size and efficiently calculate the new likelihood. If $n_{\hat{l}}$ is the number of reads with insert size $\hat{l}$, then

$$\begin{aligned} l_{new}(\mathcal{A};\mathcal{R}) &\approx l_{old}(\mathcal{A};\mathcal{R}) \\ &\quad - \sum_{\hat{l}} n_{\hat{l}} \left(\log \frac{1}{\tilde{T}_{old}(\hat{l})} - \log \frac{1}{\tilde{T}_{new}(\hat{l})} \right) \end{aligned}$$

where $\tilde{T}_{new}(l) = \tilde{T}_{old}(l) + l - 1 + g$ is the new effective length if both contigs are sufficiently large compared to insert sizes and can be precomputed for various gap sizes. The adjustments for cases where contigs are small can also be precomputed.
The likelihood of linking reads and likelihood adjustments of all other reads are combined to estimate the edge weight. We retain the edge and assign it the computed weight if it is positive and delete the edge otherwise.

Selecting joins

Once the scaffold graph is constructed, we first make unambiguous joins, i.e. join contigs connected by an edge with an increase in likelihood and such that one vertex has outdegree one and the other has indegree one. The other possible joins are sorted according to the estimated increase in the likelihood and contigs are joined if likelihood increase of the candidate join is significantly higher than other conflicting joins as determined by a heuristic (Supplementary Note 3.3). If there are other joins consistent with the candidate join, i.e. one or more contigs fit into the gap between the pair of contigs, we select from them using the dynamic programming algorithm for the weighted interval scheduling problem and remove conflicting ones. We choose a conservative approach during joining as unlinked contigs may later be merged using other datasets but incorrect joins would usually remain undetected for *de novo* assembly and might lead to errors in downstream analysis.

Implementation

The methods have been implemented in a tool called 'scaffolding with assembly likelihood optimization (SWALO)' using C/C++. The read alignement and gap estimation phases are parallelized to speed up computation. SWALO is available for download freely at http://atifrahman.github.io/SWALO/.

Data availability

We use the data and analysis scripts used in (46,47) and (54). Scripts to install tools, download data and generate the results used in this paper are available at https://github.com/atifrahman/SWALO/tree/scripts.

The scaffolds generated are also available at the same url. There may be minor variations in results due to thread race during mapping of reads unaligned by Bowtie as random subsets of unaligned reads are mapped.

RESULTS AND DISCUSSION

Comparison with stand-alone scaffolders

To compare performance of SWALO with other stand-alone scaffolders, we use the datasets used by Hunt *et al.* to evaluate scaffolding tools (46). In addition to the scaffolders considered in the study, we include subsequently published versions of Opera (Opera-LG (41)) and BESST (55). The datasets include four simulated datasets from *S. aureus* and six real datasets from *Staphylococcus aureus*, *Rhodobacter sphaeroides*, *Plasmodium falciparum* and human chromosome 14 (Supplementary Table S1). Among these the *S. aureus*, *R. sphaeroides* and human chromosome 14 datasets were also part of the GAGE project (47). The contigs were generated using Velvet (31) which were then aligned to the reference and split at locations of error to ensure misassembly free contigs. Please see (46) for more details on the datasets. We use Bowtie (56) and Bowtie 2 (57) for mapping reads, analyze results using the scripts provided in (46) and when applicable use the same parameter values for mapping and scaffolding as used in the paper by Hunt *et al.* (all parameters used are given in Supplementary Table S2). Similar to their paper, we use number of correct joins and number of incorrect joins for comparison as contiguity statistics such as N50 scaffold length biases evaluation towards scaffolders making more joins regardless of whether they are correct or incorrect while corrected N50 scaffold length leads to favorable assessment of scaffolders with more correct joins even if that is at the expense of many more incorrect joins compared to other scaffolders.

Supplementary Table S3 summarizes performance of scaffolding tools on simulated datasets. We find that SWALO makes no incorrect joins for any of the datasets. For 100kb contigs SWALO was able to make 100% of the correct joins using either library and all aligners. When the insert size library of mean 500 bp was used to scaffold 3 kb contigs, SWALO made 99.0%, 99.3% and 99.0% correct joins using Bowtie 2, Bowtie with 0 (-v 0) and 3 (-v 3) mismatches respectively. The only scaffolder that makes more than 99.3% correct joins is Opera at 99.8% when used in conjunction with BWA but this is at the cost of making 0.2% incorrect joins. For 3 kb contigs and 3 kb insert size library, SWALO made 99.6%, 99.8% and 99.6% correct joins for three mapping modes. No other scaffolder made more than 99.6% correct joins. It is worth pointing out that SWALO was able to make more correct joins when used with Bowtie -v 0 compared to Bowtie -v 3 and Bowtie 2 which may be due to reads not being mapped to some regions for the latter two.

Performance of SWALO in comparison to other scaffolders for real datasets is illustrated in Figure 2, Supplementary Figure S2 and Tables S4–S7. For the *S. aureus* dataset from GAGE, we find that SWALO made more correct joins than all other scaffolders while making 1, 1 and 2 incorrect joins in the three runs corresponding to the three ways of mapping reads. However, closer inspection reveals that one join labeled incorrect in each case is in fact a join from the end to the start of a circular sequence and is actually correct. Similarly for the *R. sphaeroides* dataset, more correct joins are made by SWALO than all other scaffolders when used in conjunction with Bowtie 2. Again three joins that are marked as incorrect are joins linking the ends to the starts of circular chromosomes or plasmids. we observe that the sequencing error rate for this dataset is high compared to the *S. aureus* dataset. So, number of reads mapped by Bowtie is quite low (46) resulting in lesser number of joins made by SWALO and other scaffolders when Bowtie is used compared to Bowtie 2.

The *P. falciparum* genome is known to be hard to assemble due its low GC content. In this case although SWALO does not make more correct joins than all other scaffolders as in other cases, the numbers of correct joins made are only slightly less than that of SOPRA, MIP and SCARPA while number of incorrect joins is less than or similar to what SOPRA made and much less than the numbers for SCARPA and MIP. We observe that many of the contigs have strings of consecutive 'A's or 'T's where very few reads are mapped to by aligners leading to poor gap estimates which may explain the comparatively smaller numbers of links by SWALO.

Finally, for the combined human chromosome 14 dataset, SWALO makes more correct joins than all other scaffolders except SOAP2 and Opera-LG both of which make more than three times incorrect joins compared to the highest number of incorrect links by SWALO and more than six times the best result by SWALO. Supplementary Figure S1 shows that the long jumping library is in fact a mixture of inserts of two sizes. When they are mapped and used to estimate gaps separately before scaffolding, and the fosmid library is applied on the output, the results improve both in terms of increase in the number of correct joins and decrease in the number of incorrect joins.

Comparison with other scaffolding modules

While Hunt *et al.* performed a comprehensive evaluation of stand-alone scaffolding tools, scaffolding modules built into some assemblers such as ALLPATHS-LG (33,58), MaSuRCA (59), CABOG (60) were left out as they cannot be run independently. In order to assess performance of SWALO in comparison to scaffolding modules of these assemblers, we ran SWALO on the contigs generated by the assemblers obtained from the GAGE project and compared the results with final results of contig assembly and scaffolding by each of these assemblers. We also include two widely used assemblers SPAdes (36) and Megahit (61) published after the GAGE study.

The results are shown in Table 1. It reveals that SWALO makes fewest number of incorrect joins in all cases except for SPAdes while making more or similar number of correct joins as ALLPATHS-LG and CABOG. For the human chromosome 14 dataset, there are 17 more joins in scaffolds generated by ALLPATHS-LG compared to SWALO output. However, nine more incorrect joins are made by ALLPATHS-LG than SWALO. Although MaSuRCA makes more correct joins than SWALO, this is at the expense of more incorrect joins which is drastically high for human chromosome 14. We observe that SWALO is able to make more correct joins than SPAdes while retaining similar incorrect to correct join ratio overall. Since the Megahit assembler does not have a scaffolding module, the result cannot be compared with SWALO results. However, we find that the contigs generated by Megahit can be reliably scaffolded using SWALO.

It may be noted that information such as actual position of reads in contigs and distance between contigs in assembly graph that were available to the assemblers could only be inferred by SWALO by mapping the reads back to the contigs using a short aligner such as Bowtie. We believe that if this information were made available by assemblers, the results could be further improved.

Time and memory requirements

SWALO uses statistical models to estimate gaps between contigs and the change in genome assembly likelihood achieved if contigs are joined. As a result it is more computationally intensive than some other scaffolders. However, we make necessary approximations to make SWALO fast, memory efficient and scalable to large datasets. Figure 3 and Supplementary Table S8 show running times and memory usage of SWALO using 32 cores on a machine with Intel Xeon E5 2.70 GHz processors to scaffold Hunt *et al.* datasets.

Although a comparison of running times is not appropriate since SWALO was run on a different machine to other scaffolding tools, we would like to note that SWALO took from approximately a minute for *S. aureus* datasets to

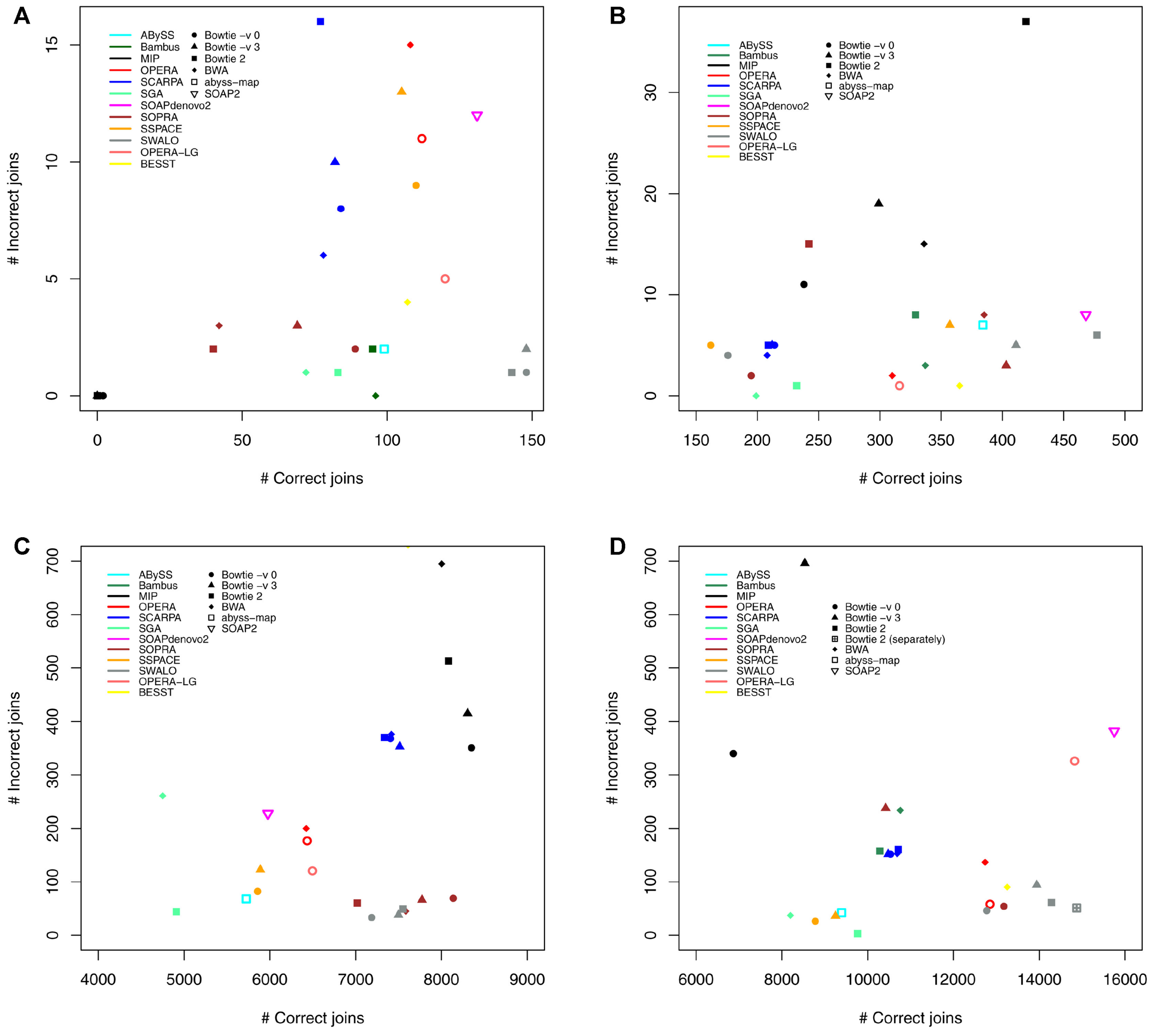


Figure 2. Performance of scaffolders. Scatter plots showing number of correct joins vs incorrect joins made by SWALO and other scaffolders on (**A**) *S. aureus* data, (**B**) *R. sphaeroides* data, (**C**) *P. falciparum* combined short and long insert data, and (**D**) human chromosome 14 combined long insert and fosmid library data. Up to 1 and 3 joins in (A) and (B), respectively made by SWALO (and possibly other scaffolders) labeled incorrect are joins from ends to starts of circular sequences and are therefore correct. Values for all scaffolders except SWALO, Opera-LG and BESST are from (46).

around 70 minutes for combined human chromosome 14 dataset to run (excluding the time required for mapping). The memory usage ranges from around 40MB for *S. aureus* to 437MB for combined human chromosome 14 dataset. We find that SWALO can scaffold 19936 contigs from human chromosome 14 using 25.1 million reads in about 70 min and using 437MB of memory.

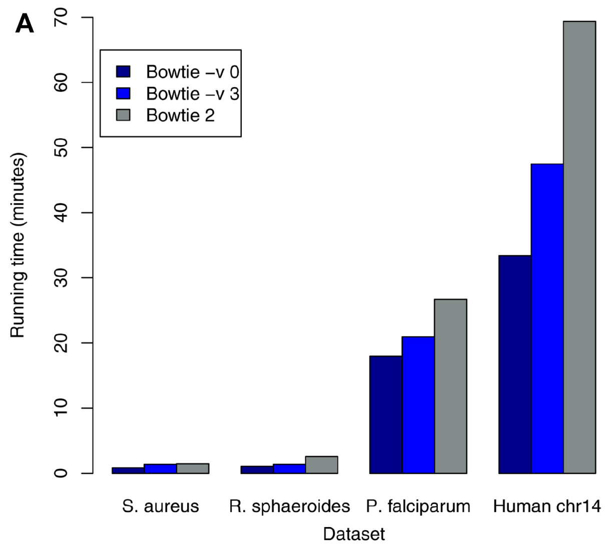
Application to a large dataset

Finally, we apply SWALO to a large dataset from a bird genome, the budgerigar (*Melopsittacus undulatus*) dataset from Assemblathon 2 (54). We selected the contigs generated by SOAPdenovo and scaffolded them using the mate-pair libraries. The SOAPdenovo contig file provided included 245 857 contigs with a total length of ∼1.1 Gb and the five mate-pair libraries had more than 730 million read pairs in total. We chose the SOAPdenovo contigs as it is widely used for assembling large genomes including the original assembly of the budgerigar genome by Ganapathy *et al.* (62). SOAPdenovo has been observed to make large numbers of joins. Although the aggressive scaffolding often leads to many incorrect joins, this allows us to compare the number of joins made by SWALO to that of SOAPdenovo along with their accuracy in absence of a complete reference genome.

Table 2 shows a summary of the mate-pair libraries and the performance of SWALO on this dataset as well as the time and memory usage. We observe that SWALO makes a total of 80 669 joins in comparison to 94 464 joins made

Table 1. Comparison of performance of SWALO with scaffolding modules built into assemblers using GAGE datasets. Comparison of results obtained by running SWALO on contigs generated by various assemblers with final results obtained by those assemblers after scaffolding

Dataset	Contig stats		Original scaffold stats		SWALO stats	
Assembler	Number	Error	Correct	Incorrect	Correct	Incorrect
S. aureus						
ALLPATHS-LG	60	15	48	**0**	**49**	**0**
MSR-CA	94	25	**74**	3	64	**2**
SPAdes	106	11	50	**2**	**69**	3
Megahit	91	31	-	-	30	1
R. sphaeroides						
ALLPATHS-LG	204	41	170	**0**	**186**	**0**
MSR-CA	395	17	**347**	5	228	**3**
CABOG	322	33	187	5	**239**	**1**
SPAdes	592	33	334	**2**	**444**	3
Megahit	605	45	-	-	309	2
Human chromosome 14						
ALLPATHS-LG	4529	2706	**4259**	45	4242	**36**
MSR-CA	30091	1901	**27521**	1145	20242	**167**
CABOG	3361	3076	2845	37	**2980**	**32**
SPAdes	27583	1876	9662	**165**	**18465**	253
Megahit	13150	3770	-	-	8326	193



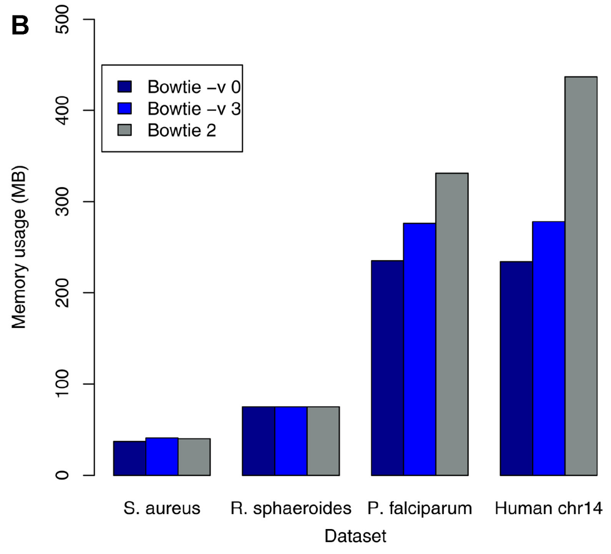


Figure 3. Running time and memory usage of SWALO. Barplots showing (**A**) running times and (**B**) memory usage of SWALO using 32 cores on a machine with Intel Xeon E5 2.70 GHz processors to scaffold Hunt *et al.* datasets.

by SOAPdenovo. We find that that SWALO automatically switches to a conservative mode (Supplementary Note 3.4) on libraries with insert size means of 10k, 20k and 40k, to keep the number of incorrect joins low.

We also find that SWALO takes less than 44 hours using 32 threads and a peak memory of 5.09 GB to scaffold 245 857 contigs using more than 730 million read pairs. It is worth noting that the times shown in Table 2 exclude the time taken by Bowtie 2 to map the reads which takes substantially more time than the time needed by SWALO to scaffold. This shows that SWALO is efficient and scalable, and thus applicable to scaffolding large genomes.

To assess the correctness of the scaffolds generated by SWALO in comparison to those by SOAPdenovo, we used the chromosome-level assembly of the budgerigar (*Melopsittacus undulatus*) genome by O'Connor *et al.* (63) generated through computational and lab-based approaches. The quality of the scaffolds as well as the original SOAPdenovo contigs were assessed using the quality assessment tool QUAST (64). Table 3 summarizes the numbers of correct joins and mis-assemblies made by SWALO and SOAPdenovo. We observe that SWALO makes 78 501 and 2168 correct and incorrect joins respectively in comparison to 90 389 and 4075 by SOAPdenovo i.e. SWALO makes only 13% less correct joins while making 47% less mis-assemblies than SOAPdenovo.

CONCLUSION

The results show that SWALO performs well consistently and is able to identify many correct joins while keeping the number of incorrect joins very low. It also shows pareto-optimal performance in the datasets we have analyzed, i.e. there is a run of SWALO such that no other scaffolder in any of their

Table 2. Performance of SWALO on a large dataset from a bird genome. Description of five mate-pair libraries from the budgerigar (*Melopsittacus undulatus*) dataset from Assemblathon 2 (54), time and memory requirements of SWALO, and the number of joins made by it. The first two libraries were used in the combined mode. For the last three libraries, SWALO switches to a conservative mode automatically and so the hierarchical approach is used as recommended

Accessions	# read pairs	Orientation	Insert size	Mode	Scaffolding Time (hh:mm:ss)	Peak memory (GB)	# joins
ERR244148-150	264 708 963	Mate-pair	2000	Combined	27:36:12	5.09	70 634
ERR244151-152	194 240 419	Mate-pair	5000				
ERR244153	94 282 287	Mate-pair	10 000	Hierarchical, Conservative (auto)	5:02:57	2.09	5372
ERR244154	89 722 180	Mate-pair	20 000	Hierarchical, Conservative (auto)	5:26:47	2.14	2957
ERR244155	87 489 651	Mate-pair	40 000	Hierarchical, Conservative (auto)	5:48:02	2.41	1706
Total	730 443 500				43:53:58	5.09	80 669

Table 3. Comparison of results of SWALO and SOAPdenovo on the bird genome dataset. Comparison of number of scaffolds, total number of joins, and number of mis-assemblies introduced while scaffolding by SWALO and SOAPdenovo using five mate-pair libraries from the budgerigar (*Melopsittacus undulatus*) dataset from Assemblathon 2 (54). In addition, the original numbers of contigs and mis-assemblies in the contigs generated by SOAPdenovo are also shown

Assembly	Number of contigs/ scaffolds	Number of correct joins	Number of mis-assemblies (additional)
SOAPdenovo contigs	245 857	-	5658
SOAPdenovo scaffolds	151 393	90 389	4075
SWALO scaffolds	166 894	78 501	2168

runs was able to make more correct joins while making less than the number of incorrect links by SWALO. We observe that consistent results are achieved when SWALO is used with Bowtie 2. However, when reads are largely error free results achieved using Bowtie with no mismatches can be better possibly due to reads being mapped to more regions compared to Bowtie 2.

Overall we find that SWALO outperforms all other scaffolders on real and simulated datasets. This indicates that genome assembly may be substantially improved through the use of statistical models. The method may further be improved by modifying the heuristic used to select among multiple candidate joins and by considering global properties of the scaffold graph. The methods may also be extended to scaffolding with long reads generated by SMRT and nanopore sequencing. The improvement in scaffolding achieved by a practical method based on assembly likelihoods opens up the possibility that other problems related to genome assembly such as reference guided assembly, misassembly correction, copy number estimation, gap-filling may also be amenable to this approach.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

ACKNOWLEDGEMENTS

We thank Dan Rokhsar, Páll Melsted, Harold Pimentel, Shannon McCurdy and Nicolas Bray for helpful conversations during the development of SWALO.

FUNDING

NIH [R01 HG006129 to L.P., in part]; Fulbright Science & Technology Fellowship [15093630 to A.R., in part]. Funding for open access charge: NIH [R01 HG006129].
Conflict of interest statement. None declared.

REFERENCES

1. Margulies,M., Egholm,M., Altman,W.E., Attiya,S., Bader,J.S., Bemben,L.A., Berka,J., Braverman,M.S., Chen,Y.-J., Chen,Z. *et al.* (2005) Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**, 376–380.
2. Harris,T.D., Buzby,P.R., Babcock,H., Beer,E., Bowers,J., Braslavsky,I., Causey,M., Colonell,J., Dimeo,J., Efcavitch,J.W. *et al.* (2008) Single-molecule DNA sequencing of a viral genome. *Science*, **320**, 106–109.
3. Valouev,A., Ichikawa,J., Tonthat,T., Stuart,J., Ranade,S., Peckham,H., Zeng,K., Malek,J.A., Costa,G., McKernan,K. *et al.* (2008) A high-resolution, nucleosome position map of *C. elegans* reveals a lack of universal sequence-dictated positioning. *Genome Res.*, **18**, 1051–1063.
4. Rothberg,J.M., Hinz,W., Rearick,T.M., Schultz,J., Mileski,W., Davey,M., Leamon,J.H., Johnson,K., Milgrew,M.J., Edwards,M. *et al.* (2011) An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, **475**, 348–352.
5. Eid,J., Fehr,A., Gray,J., Luong,K., Lyle,J., Otto,G., Peluso,P., Rank,D., Baybayan,P., Bettman,B. *et al.* (2009) Real-time DNA sequencing from single polymerase molecules. *Science*, **323**, 133–138.
6. Branton,D., Deamer,D.W., Marziali,A., Bayley,H., Benner,S.A., Butler,T., Di Ventra,M., Garaj,S., Hibbs,A., Huang,X. *et al.* (2008) The potential and challenges of nanopore sequencing. *Nat. Biotechnol.*, **26**, 1146–1153.
7. Koren,S., Walenz,B.P., Berlin,K., Miller,J.R., Bergman,N.H. and Phillippy,A.M. (2017) Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res.*, **27**, 722–736.
8. Chin,C.-S., Peluso,P., Sedlazeck,F.J., Nattestad,M., Concepcion,G.T., Clum,A., Dunn,C., O'Malley,R., Figueroa-Balderas,R., Morales-Cruz,A. *et al.* (2016) Phased diploid genome assembly with single-molecule real-time sequencing. *Nat. Methods*, **13**, 1050–1054.
9. Li,H. (2016) Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, **32**, 2103–2110.
10. Kolmogorov,M., Yuan,J., Lin,Y. and Pevzner,P.A. (2019) Assembly of long, error-prone reads using repeat graphs. *Nat. Biotechnol.*, **37**, 540–546.
11. Kim,J., Larkin,D.M., Cai,Q., Zhang,Y., Ge,R.-L., Auvil,L., Capitanu,B., Zhang,G., Lewin,H.A., Ma,J. *et al.* (2013) Reference-assisted chromosome assembly. *Proc. Natl. Acad. Sci. U.S.A.*, **110**, 1785–1790.
12. Kolmogorov,M., Raney,B., Paten,B. and Pham,S. (2014) Ragout—a reference-assisted assembly tool for bacterial genomes. *Bioinformatics*, **30**, i302–i309.

13. Bosi,E., Donati,B., Galardini,M., Brunetti,S., Sagot,M.-F., Lió,P., Crescenzi,P., Fani,R. and Fondi,M. (2015) MeDuSa: a multi-draft based scaffolder. *Bioinformatics*, **31**, 2443–2451.
14. Fleischmann,R.D., Adams,M.D., White,O., Clayton,R.A., Kirkness,E.F., Kerlavage,A.R., Bult,C.J., Tomb,J.-F., Dougherty,B.A., Merrick,J.M. *et al.* (1995) Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science*, **269**, 496–512.
15. Weber,J.L. and Myers,E.W. (1997) Human whole-genome shotgun sequencing. *Genome Res.*, **7**, 401–409.
16. Huson,D.H., Reinert,K. and Myers,E.W. (2002) The greedy path-merging algorithm for contig scaffolding. *J. ACM*, **49**, 603–615.
17. Yeo,S., Coombe,L., Warren,R.L., Chu,J. and Birol,I. (2018) ARCS: scaffolding genome drafts with linked reads. *Bioinformatics*, **34**, 725–731.
18. Coombe,L., Zhang,J., Vandervalk,B.P., Chu,J., Jackman,S.D., Birol,I. and Warren,R.L. (2018) ARKS: chromosome-scale scaffolding of human genome drafts with linked read kmers. *BMC Bioinformatics*, **19**, 234.
19. Weisenfeld,N.I., Kumar,V., Shah,P., Church,D.M. and Jaffe,D.B. (2017) Direct determination of diploid genome sequences. *Genome Res.*, **27**, 757–767.
20. Boetzer,M. and Pirovano,W. (2014) SSPACE-LongRead: scaffolding bacterial draft genomes using long read sequence information. *BMC Bioinformatics*, **15**, 211.
21. Warren,R.L., Yang,C., Vandervalk,B.P., Behsaz,B., Lagman,A., Jones,S.J. and Birol,I. (2015) LINKS: scalable, alignment-free scaffolding of draft genomes with long reads. *GigaScience*, **4**, 35.
22. Wick,R.R., Judd,L.M., Gorrie,C.L. and Holt,K.E. (2017) Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS Comput. Biol.*, **13**, e1005595.
23. Cao,M.D., Nguyen,S.H., Ganesamoorthy,D., Elliott,A.G., Cooper,M.A. and Coin,L.J. (2017) Scaffolding and completing genome assemblies in real-time with nanopore sequencing. *Nat. Commun.*, **8**, 14515.
24. Dudchenko,O., Batra,S.S., Omer,A.D., Nyquist,S.K., Hoeger,M., Durand,N.C., Shamim,M.S., Machol,I., Lander,E.S., Aiden,A.P. *et al.* (2017) De novo assembly of the Aedes aegypti genome using Hi-C yields chromosome-length scaffolds. *Science*, **356**, 92–95.
25. Burton,J.N., Adey,A., Patwardhan,R.P., Qiu,R., Kitzman,J.O. and Shendure,J. (2013) Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. *Nat. Biotechnol.*, **31**, 1119–1125.
26. Kaplan,N. and Dekker,J. (2013) High-throughput genome scaffolding from in vivo DNA interaction frequency. *Nat. Biotechnol.*, **31**, 1143–1147.
27. Putnam,N.H., O'Connell,B.L., Stites,J.C., Rice,B.J., Blanchette,M., Calef,R., Troll,C.J., Fields,A., Hartley,P.D., Sugnet,C.W. *et al.* (2016) Chromosome-scale shotgun assembly using an in vitro method for long-range linkage. *Genome Res.*, **26**, 342–350.
28. Ghurye,J., Pop,M., Koren,S., Bickhart,D. and Chin,C.-S. (2017) Scaffolding of long read assemblies using long range contact information. *BMC Genomics*, **18**, 527.
29. Ghurye,J., Rhie,A., Walenz,B.P., Schmitt,A., Selvaraj,S., Pop,M., Phillippy,A.M. and Koren,S. (2019) Integrating Hi-C links with assembly graphs for chromosome-scale assembly. *PLoS Comput. Biol.*, **15**, e1007273.
30. Ghurye,J. and Pop,M. (2019) Modern technologies and algorithms for scaffolding assembled genomes. *PLoS Comput. Biol.*, **15**, e1006994.
31. Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.
32. Simpson,J.T., Wong,K., Jackman,S.D., Schein,J.E., Jones,S.J. and Birol,I. (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.
33. Butler,J., MacCallum,I., Kleber,M., Shlyakhter,I.A., Belmonte,M.K., Lander,E.S., Nusbaum,C. and Jaffe,D.B. (2008) ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Res.*, **18**, 810–820.
34. Luo,R., Liu,B., Xie,Y., Li,Z., Huang,W., Yuan,J., He,G., Chen,Y., Pan,Q., Liu,Y. *et al.* (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Gigascience*, **1**, 18.
35. Simpson,J. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res*, **22**, 549–556.
36. Bankevich,A., Nurk,S., Antipov,D., Gurevich,A.A., Dvorkin,M., Kulikov,A.S., Lesin,V.M., Nikolenko,S.I., Pham,S., Prjibelski,A.D. *et al.* (2012) SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, **19**, 455–477.
37. Pop,M., Kosack,D. and Salzberg,S. (2004) Hierarchical scaffolding with Bambus. *Genome Res*, **14**, 149–159.
38. Koren,S., Treangen,T. and Pop,M. (2011) Bambus 2: scaffolding metagenomes. *Bioinformatics*, **27**, 2964–2971.
39. Salmela,L., Makinen,V., Valimaki,N., Ylinen,J. and Ukkonen,E. (2011) Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, **27**, 3259–3265.
40. Gao,S., Sung,W.-K. and Nagarajan,N. (2011) Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *J. Comput. Biol.*, **18**, 1681–1691.
41. Gao,S., Bertrand,D., Chia,B.K. and Nagarajan,N. (2016) OPERA-LG: efficient and exact scaffolding of large, repeat-rich eukaryotic genomes with performance guarantees. *Genome Biol.*, **17**, 1.
42. Donmez,N. and Brudno,M. (2013) SCARPA: scaffolding reads with practical algorithms. *Bioinformatics*, **29**, 428–434.
43. Dayarian,A., Michael,T. and Sengupta,A. (2010) SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, **11**, 345.
44. Boetzer,M., Henkel,C., Jansen,H., Butler,D. and Pirovano,W. (2011) Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, **27**, 578–579.
45. Sahlin,K., Vezzi,F., Nystedt,B., Lundeberg,J. and Arvestad,L. (2014) BESST-efficient scaffolding of large fragmented assemblies. *BMC bioinformatics*, **15**, 1.
46. Hunt,M., Newbold,C., Berriman,M. and Otto,T. (2014) A comprehensive evaluation of assembly scaffolding tools. *Genome Biol.*, **15**, R42.
47. Salzberg,S.L., Phillippy,A.M., Zimin,A., Puiu,D., Magoc,T., Koren,S., Treangen,T.J., Schatz,M.C., Delcher,A.L., Roberts,M. *et al.* (2011) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.
48. Rahman,A. and Pachter,L. (2013) CGAL: computing genome assembly likelihoods. *Genome Biol.*, **14**, R8.
49. Chapman,J.A., Ho,I., Sunkara,S., Luo,S., Schroth,G.P. and Rokhsar,D.S. (2011) Meraculous: *De Novo* genome assembly with short paired-end reads. *PLoS ONE*, **6**, e23501.
50. Sahlin,K., Street,N., Lundeberg,J. and Arvestad,L. (2012) Improved gap size estimation for scaffolding algorithms. *Bioinformatics*, **28**, 2215–2222.
51. Dempster,A.P., Laird,N.M. and Rubin,D.B. (1977) Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc. B (Methodological)*, **39**, 1–38.
52. Trapnell,C., Williams,B.A., Pertea,G., Mortazavi,A., Kwan,G., Van Baren,M.J., Salzberg,S.L., Wold,B.J. and Pachter,L. (2010) Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat. Biotechnol.*, **28**, 511–515.
53. Medvedev,P. and Brudno,M. (2009) Maximum likelihood genome assembly. *J. Computat. Biol.*, **16**, 1101–1116.
54. Bradnam,K., Fass,J., Alexandrov,A., Baranay,P., Bechner,M., Birol,I., Boisvert,S., Chapman,J., Chapuis,G., Chikhi,R. *et al.* (2013) Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *Gigascience*, **2**, 10.
55. Sahlin,K., Chikhi,R. and Arvestad,L. (2016) Assembly scaffolding with PE-contaminated mate-pair libraries. *Bioinformatics*, **32**, 1925–1932.
56. Langmead,B., Trapnell,C., Pop,M. and Salzberg,S. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25.
57. Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.
58. Gnerre,S., MacCallum,I., Przybylski,D., Ribeiro,F.J., Burton,J.N., Walker,B.J., Sharpe,T., Hall,G., Shea,T.P., Sykes,S. *et al.* (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl. Acad. Sci. U.S.A.*, **108**, 1513–1518.

59. Zimin,A.V., Marçais,G., Puiu,D., Roberts,M., Salzberg,S.L. and Yorke,J.A. (2013) The MaSuRCA genome assembler. *Bioinformatics*, **29**, 2669–2677.
60. Miller,J.R., Delcher,A.L., Koren,S., Venter,E., Walenz,B.P., Brownley,A., Johnson,J., Li,K., Mobarry,C. and Sutton,G. (2008) Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**, 2818–2824.
61. Li,D., Liu,C.-M., Luo,R., Sadakane,K. and Lam,T.-W. (2015) MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, **31**, 1674–1676.
62. Ganapathy,G., Howard,J.T., Ward,J.M., Li,J., Li,B., Li,Y., Xiong,Y., Zhang,Y., Zhou,S., Schwartz,D.C. *et al.* (2014) High-coverage sequencing and annotated assemblies of the budgerigar genome. *GigaScience*, **3**, 11.
63. O'Connor,R.E., Farré,M., Joseph,S., Damas,J., Kiazim,L., Jennings,R., Bennett,S., Slack,E.A., Allanson,E., Larkin,D.M. *et al.* (2018) Chromosome-level assembly reveals extensive rearrangement in saker falcon and budgerigar, but not ostrich, genomes. *Genome Biol.*, **19**, 171.
64. Gurevich,A., Saveliev,V., Vyahhi,N. and Tesler,G. (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.