Bioinformatics, 38(15), 2022, 3717–3724
https://doi.org/10.1093/bioinformatics/btac404
Advance Access Publication Date: 22 June 2022
Original Paper

Sequence analysis

Figbird: a probabilistic method for filling gaps in genome assemblies

Sumit Tarafder[1,2], Mazharul Islam[1,2], Swakkhar Shatabda [2] and Atif Rahman [1,*]

[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1205, Bangladesh and [2]Department of Computer Science and Engineering, United International University, Dhaka 1212, Bangladesh

*To whom correspondence should be addressed.

Associate Editor: Can Alkan

Received on November 30, 2021; revised on June 12, 2022; editorial decision on June 13, 2022; accepted on June 17, 2022

Abstract

Motivation: Advances in sequencing technologies have led to the sequencing of genomes of a multitude of organisms. However, draft genomes of many of these organisms contain a large number of gaps due to the repeats in genomes, low sequencing coverage and limitations in sequencing technologies. Although there exists several tools for filling gaps, many of these do not utilize all information relevant to gap filling.

Results: Here, we present a probabilistic method for filling gaps in draft genome assemblies using second-generation reads based on a generative model for sequencing that takes into account information on insert sizes and sequencing errors. Our method is based on the *expectation-maximization* algorithm unlike the graph-based methods adopted in the literature. Experiments on real biological datasets show that this novel approach can fill up large portions of gaps with small number of errors and misassemblies compared to other state-of-the-art gap-filling tools.

Availability and implementation: The method is implemented using C++ in a software named 'Filling Gaps by Iterative Read Distribution (Figbird)', which is available at https://github.com/SumitTarafder/Figbird.

Contact: atif@cse.buet.ac.bd

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Genome sequencing of an organism, i.e. determining the sequence of nucleotides that make up the entire genome is often a prerequisite for performing experiments to study that organism and fundamental to understanding how different organisms relate to each other. With the advancement of sequencing technologies in the past decade and a half, the availability of sequencing data has increased drastically and genomes of many organisms have been sequenced. However, eukaryotic chromosomes can be hundreds of millions of base pairs long, and there is no sequencing technology on the market that can sequence an entire chromosome telomere to telomere. Therefore, instead of sequencing the complete genome, different sequencing technologies generate millions of smaller fragments called reads. The whole genome is then reconstructed from these read sequences through a process known as genome assembly.

The high throughput, low cost and error rates of second-generation technologies such as Illumina (Meyer and Kircher, 2010) have led to its use in a large number of sequencing projects, and many assembly tools such as ABySS (Simpson *et al.*, 2009), Velvet (Zerbino and Birney, 2008), Allpaths-LG (Butler *et al.*, 2008), SPAdes (Prjibelski *et al.*, 2020), etc. have been developed to sequence genomes using this technology. However, due to the short lengths of second-generation reads, assembly is performed in multiple steps. The first step of the assembly pipeline constitutes of stitching the read sequences into contiguous sequences called contigs and then in the next step, read pairs (paired-end or mate pair) or information from other technologies are used to orient and organize the contigs into scaffolds. Despite the development in methodologies for genome assembly, the draft assemblies constructed with these tools still contain thousands of intervening gaps within the assembled scaffolds due to repetitive regions in genomes and regions with low sequencing coverage. Filling these gaps with minimal introduction of error is a crucial step in genome assembly pipelines as an error-free complete genome can lead to better downstream analysis such as genotyping variants without errors, complete annotation of genes (Chaisson *et al.*, 2015), identification of effector and coregulated genes (Thomma *et al.*, 2016), and accurate statistical analysis (Domanska *et al.*, 2018).

To address this issue, a number of tools have been developed for gap filling using short reads. Many of the genome assemblers, such as ABySS and Allpaths-LG, include gap-filling modules in their pipeline. In addition, stand-alone tools such as GapCloser in the SOAPdenovo (Luo *et al.*, 2012) package, GapFiller (Boetzer and Pirovano, 2012), Gap2Seq (Salmela *et al.*, 2016), Sealer (Paulino *et al.*, 2015), etc. also exist for this purpose. GapCloser constructs a de Bruijn graph on the set of available reads to perform the local

assembly. Although it works well for smaller genomes, it is highly memory inefficient (Paulino *et al.*, 2015) for larger genomes and only considers read pairs with insert size <2000 base pairs. GapFiller, on the other hand, uses read pairs with one end aligned to the scaffold and the other end partially aligned to gap regions. The reads are then assembled using a k-mer-based method to fill the gaps. A major limitation of GapFiller is it uses only read pairs with one end fully aligned and the other end partially aligned (Supplementary Note S2.1), and does not utilize read pairs with one end fully unaligned. Thus a lot of sequence information is not taken into account during its gap-filling procedure. A recent computational approach for gap filling has been introduced in Salmela *et al.* (2016) where the problem is formulated as an exact path length problem, implemented in pseudo-polynomial time with some optimizations, and packaged in a tool called Gap2Seq. However, their approach does not scale to large genomes and is unable to fill large gaps due to the expensive computational approach. Lastly, a resource-efficient gap-filling software named Sealer (Paulino *et al.*, 2015) has been designed to close gaps in scaffolds by navigating de Bruijn graphs represented by a space-efficient data structure called Bloom filter and is scalable to large Giga base pair sized genomes. It uses an assembly utility within the ABySS package, called Konnector (Vandervalk *et al.*, 2014), as its engine to close intrascaffold gaps. The Konnector utility takes the flanking sequence pairs along with a set of reads with a high level of coverage redundancy as inputs and runs with a range of k-mer lengths to connect the flanking gap sequences. Sealer ignores size discrepancies between gaps and newly introduced sequences since gap sizes are often estimated from insert size distributions, and assemblers do not generally provide confidence intervals for every region of Ns. However, it does not consider the insert size of paired reads during gap filling.

Almost all of the methods for gap filling use a graph-based formulation of the problem, most commonly de Bruijn graph, and then try to reconstruct a path through the graph that corresponds to the gap sequence. But there can be multiple such paths present in the graph due to repetitive regions or sequencing errors (Zerbino and Birney, 2008), and only one of those paths corresponds to the true genomic sequence of the gap. Finding such a correct path can get complicated either due to the presence of repeats or because of the memory constraint due to the nature of the graph built from a large set of k-mers. Moreover, once the reads to be used for filling a gap are identified, most of the tools ignore distance information from the other end of the pair, i.e. insert size, which may help disambiguate among multiple sequences and solve repeat-related issues. So, searching for the actual gap sequence that can solve the above-stated problems is still an area to be explored in genome assembly.

Recently, sequencing technologies have gone through a further revolution and 'third generation' single-molecule technologies, such as Pacific Bio-sciences and Oxford Nanopore have been developed which can generate read sequences of lengths tens to hundreds of kilo-base pairs and beyond. Among the long read and/or contig-based gap closing approaches existing in literature, GMcloser (Kosugi *et al.*, 2015), PBJelly (English *et al.*, 2012), gapFinisher (Kammonen *et al.*, 2019), TGS-GapCloser (Xu *et al.*, 2020), LR_Gapcloser (Xu *et al.*, 2019), PGcloser (Lu *et al.*, 2020), etc. are worth mentioning which use long reads or alternatively, assembled contig set from short-read libraries to fill gaps using sequence alignment and to determine consensus sequences. Although long reads lead to substantially better genome assemblies (Frank *et al.*, 2016), their use in the gap-filling process is still limited due to the high error rate in these technologies. Both PacBio and Nanopore technologies have higher error rates than second-generation Illumina technologies though long reads with low error rates such as PacBio HiFi reads are now emerging. To mitigate the effect of error-prone data, high coverage is required to ensure a low error rate in the generated assemblies. As gap filling is one of the last stages of genome assembly pipeline, any error introduced in this stage will carry over to subsequent genomic analysis and may lead to incorrect results. To this end, we have chosen comparatively accurate second-generation read sequences for gap-filling purposes.

In this work, we formulate gap filling as a parameter estimation problem and develop a probabilistic method for filling gaps in scaffolds using second-generation reads. The method is based on a generative model for sequencing proposed in CGAL (Rahman and Pachter, 2013) and subsequently used to develop a scaffolding tool SWALO (Rahman and Pachter, 2021). The model incorporates information such as distribution of insert size of read pairs, sequencing errors, etc. and can be used to compute the likelihood of an assembly with respect to a set of read pairs. We use this model to estimate the length of the gap and to find a sequence for each gap that maximizes the probability of the reads mapping to that gap region. We note that gap length estimation in this context is distinct from gap size estimation between contigs using read pairs with ends mapping to different contigs for scaffolding purposes (Chapman *et al.*, 2011; Sahlin *et al.*, 2012). In our case, only one end of the read pairs may be mapped, and thus the insert sizes of the read pairs may be completely unknown. So, we use an iterative approach based on the *expectation-maximization* (EM) algorithm (Dempster *et al.*, 1977), which has been used to solve many problems in computational biology including motif finding (Bailey and Elkan, 1995) and transcript abundance estimation from RNA-Seq (Pachter, 2011). Our method is implemented in a tool called Figbird and an extensive comparison with other standalone gap fillers is performed on datasets from the GAGE (Salzberg *et al.*, 2012) project as well as assemblies obtained from the widely used assembler SPAdes (Prjibelski *et al.*, 2020). Overall, our probabilistic method performs well consistently on six different metrics over a variety of real draft assemblies and is able to reduce the amount of gaps substantially while keeping misassemblies and errors low. Although the method is computationally intensive, the execution time is proportional to the total lengths of reads, the number of EM iterations as well as the number and the lengths of gaps, and is thus scalable and applicable to large genomes.

2 Materials and methods

In this section, we describe the methods behind Figbird. First, we present an overview of the method and subsequently discuss the gap-filling step in detail.

2.1 Overview of Figbird

A high-level overview of our gap-filling method is illustrated in the block diagram in Figure 1. The method consists of the following two major phases.

Pre-processing: In this phase, relevant read pairs are identified, and distributions are learned. At first, paired-end and mate-pair reads (we will refer to both these types as read pairs) are aligned to

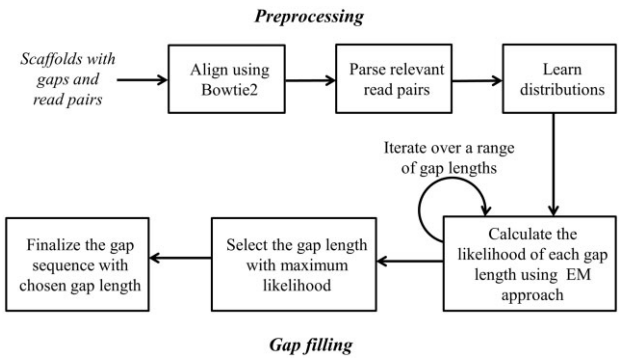

Fig. 1. Overview of Figbird. Figbird takes as input scaffolds containing gaps and sets of read pairs (paired-end or mate pairs) and aligns the read pairs to the scaffolds using Bowtie2. The output is then parsed to separate fully mapped as well as to assign one-end unmapped, and one-end partially mapped read pairs to specific gaps. Insert size and error distributions are then learnt using uniquely and fully mapped read pairs. In the gap-filling phase, one-end unmapped and partially mapped reads are used to fill gaps using the EM algorithm for a range of gap lengths. Finally, reads with probabilities below a threshold are filtered, and consensus sequence is formed for the gap length with maximum likelihood

the scaffold set using Bowtie2 (Langmead and Salzberg, 2012). Then the output of Bowtie2 is parsed to collect fully mapped, one-end unmapped and one-end partially mapped read pairs necessary for our method. The details of the read pairs as well as the read parsing criteria are described in Supplementary Note S2.1. Next, the insert size distribution and parameters for our error model are learned using uniquely and fully mapped read pairs (see Supplementary Note S2.2 for details).

Gap filling: In the gap-filling phase, read pairs with one-end unmapped or partially mapped are locally assembled using a maximum likelihood approach calculated using a model described in Section 2.2. As we do not know exactly where the unmapped end of the read pair should be placed within the gap, we use the EM algorithm (Dempster *et al.*, 1977) to iteratively find the placement of the read using the learned insert size distribution and the current probability estimates of the nucleotides in the gap sequence, and re-estimate the probabilities of the nucleotides using the current placements. The process is iterated over a range of gap estimates with different lengths, and the one with the maximum likelihood value is chosen to fill the gap region. Once the gap length and the corresponding sequence are estimated, the distribution of probability of fully mapped reads learnt from the previous step is used to decide whether the reads should be considered to fill that particular gap based on a cut-off value and reads with probability below the cut-off are discarded. Finally, a consensus is calculated based on the probabilistic placements of the chosen reads in gaps which are regarded as the final predicted gap sequence.

2.2 Gap filling using the EM algorithm

In the next phase, we fill the gaps using read pairs with one end mapped and the other end unmapped or partially mapped with a likelihood-based approach. Given a gap sequence $\mathcal{G}$ of length g and a set of reads $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$, the log-likelihood of $\mathcal{G}$ is given by

$$l(\mathcal{G}; \mathcal{R}) = \log \prod_{i=1}^{N} p(r_i|\mathcal{G}),$$

where $p(r_i|\mathcal{G})$ is the probability that r_i is generated from $\mathcal{G}$. We are interested in the gap sequence that maximizes this likelihood. To calculate this, we use the generative model described in CGAL (Rahman and Pachter, 2013). However, since one end of the read pair is mapped to a fixed position, we modify the model and define the probability of a read as follows:

$$p(r_i|\mathcal{G}) = p_F(L)p_E(r_i|\mathcal{G}, L),$$

where L is the insert size of the read pair, and p_F and p_E are insert size and sequence probabilities, respectively.

In this paper, we formulate gap filling as a parameter estimation problem. Given a gap of length g, we introduce the parameters to be estimated as

$$\theta_{j,c} \text{ for } 1 \leq j \leq g \text{ and } c \in \{A, C, G, T\},$$

where $\theta_{j,c}$ denotes the probability of nucleotide c at index j of a gap sequence. The gap filling problem then converts into a parameter estimation problem where the goal is to find the estimates of $\theta_{j,c}$'s that maximize the likelihood $l(\boldsymbol{\theta}_{g,c}; \mathcal{R})$.

However, we need to know the insert sizes exactly to estimate these parameters. Although the insert sizes are known for one-end partially mapped reads, we do not know these for one-end unmapped reads as sequencing experiments do not provide the exact distance between the two ends. It is worth noting that although we do not exactly know the insert size values, the read pairs follow an approximately normal distribution which can be learnt. This observation can reduce the possible positions of the unmapped end within a range of minimum and maximum value of insert size for that read pair as indicated in Figure 2. Here, we use a smoothed and truncated version of the empirical insert size distribution learnt using the uniquely mapped reads (Supplementary Note S2.2). To reduce bias toward small inserts, the user can optionally provide an estimate of the mean of the insert size distribution, which is set as a limit on the minimal size of scaffolds used to learn the distribution. However, the results in this paper are obtained without setting any limits.

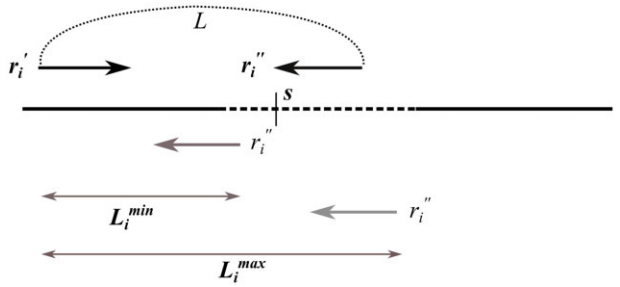

Fig. 2. Possible placement of the unmapped end r_i'' of read r_i in different gap positions

Now, if the insert size of a read pair was exactly known, we could have placed the read at the correct position within the gap and used the sequence to adjust the probabilities of the nucleotides occurring at those gap positions as shown in Figure 3A. On the other hand, if the gap sequence was known to us beforehand, then we could have aligned the read to that known sequence and obtained the most likely placement of the unmapped end as shown in Figure 3B.

But neither the exact insert sizes nor the sequence of the gap region are known beforehand, and to solve one of these problems we need the solution to the other. To solve this set of interlocking problems, we will use the EM algorithm. The EM algorithm can be applied to solve those class of problems that have some hidden observations as well as unknown model parameters. It proceeds by picking an initial set of model parameters to estimate the hidden observations with the assumption that the data come from a specific model. This is called the *E-step*. Then, using the newly estimated values of hidden observations, the parameters or initial hypothesis gets updated. This step is called the *M-step*. These two steps are iterated until the resulting values converge to a fixed point or the allocated time ends. In our method, the hidden observations are insert sizes of the read pairs, and the parameters are the probabilities of each nucleotide occurring at each gap position.

2.2.1 EM formulation

A schematic diagram of our EM approach for gap filling is shown in Figure 3C. We start with an initial hypothesis that each of the four bases $\{A, C, G, T\}$ is equally probable at each gap position j and initializes the estimation parameter $\theta_{j,c}$ with 0.25 for all gap positions and for all four possibilities of nucleotide c. Then, the E-step and M-step will be applied iteratively as follows:

E-step: In the E-step, we place the unmapped end of the read $r_i = c_1 c_2 \ldots c_l$ in all possible gap positions based on the set of allowable insert sizes $\mathcal{L}_i = [L_i^{min}, L_i^{max}]$, where L_i^{min} and L_i^{max} are the minimum and maximum threshold value of insert size for read r_i, respectively, and calculate the probability that the read is generated from that position using the current estimated probabilities $\boldsymbol{\theta}_{g,c}$. This posterior probability of read r_i having a particular insert size $L \in \mathcal{L}_i$, i.e. that it starts at s (Figure 2), is given by:

$$f_i(L) \propto p_F(L) \left[\prod_{j=s,k=1}^{j=s+l-1,k=l} (\theta_{j,c_k}(1 - p_{er}(k)) + \tau_{j,c_k} p_{er}(k)) \right],$$

where l is the length of the read and s is the start position of the read within the gap corresponding to the insert size L, $p_{er}(k)$ is the error probability at read position k and τ_{j,c_k} is the probability of getting nucleotide c at gap index j and read position k due to a sequencing error, which can be calculated using the following equation:

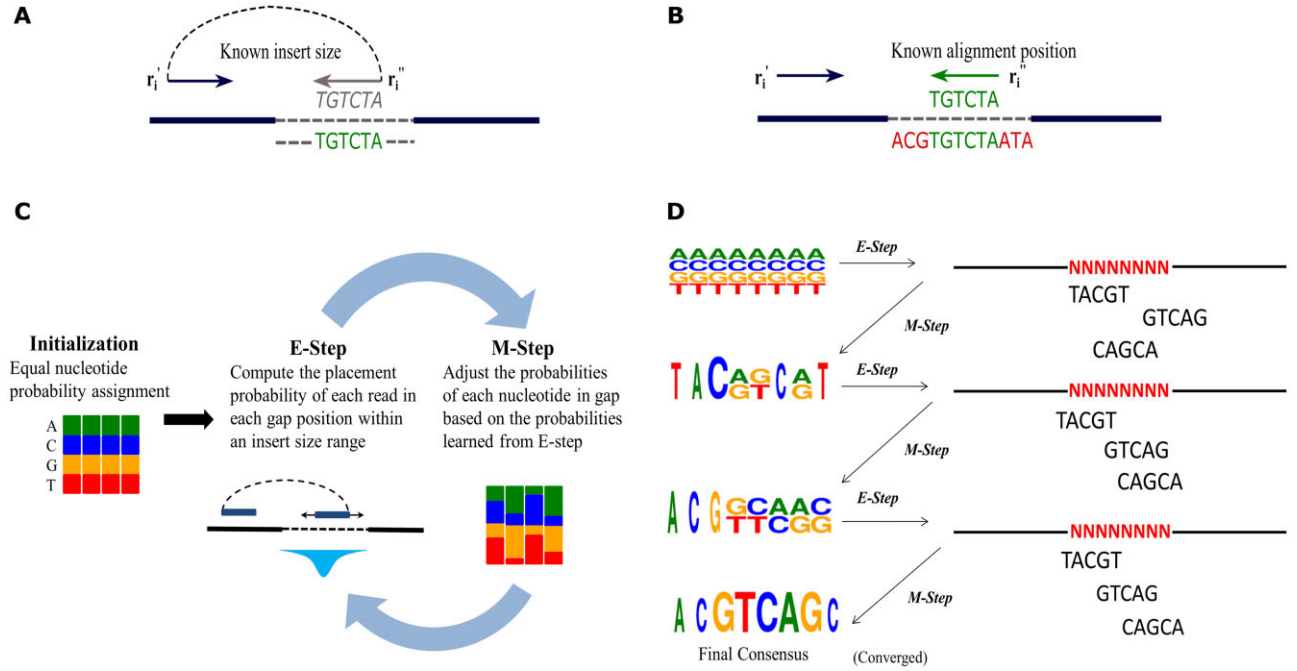

Fig. 3. Formulation of gap filling using the EM algorithm. (**A**) If the insert sizes are known exactly, the reads can be placed within the gap in the correct positions and the gap sequence can be inferred. (**B**) If the sequence of nucleotides in the gap is known, reads can be aligned to the sequence, and their placement and insert sizes can be estimated. (**C**) Figbird solves gap filling using the EM algorithm. It starts by initializing each nucleotide with equal probability. In the E-step, current probabilities of nucleotides and insert size distribution are used to calculate the placement probabilities of each read within the gap. Then in the M-step, the placement probabilities and read sequences are used to update the probabilities of nucleotides. These two steps are iterated until convergence. (**D**) A simplified simulation of the EM algorithm for gap filling

$$\tau_{j,c_k} = \sum_{k=1}^{5}\sum_{i=1}^{5} \theta_{j,i}\gamma_{i,c_k},$$

where γ is a 5×5 square matrix containing the probability of substitution of each of the four nucleotide characters and character 'N' into others.

M-step: In this step, we accumulate the probabilities calculated for all the reads in E-step in an intermediate matrix α with the same dimensions as θ. If s is the starting position of read $r_i \in \mathcal{R}$ in the gap with respect to an insert size $L \in \mathcal{L}_i$, α will be updated according to the following equation:

$$\alpha_{j,c} = \epsilon + \sum_{i=1}^{N}\sum_{L\in L_i} f_i(L)\mathbb{1}_{\mathrm{i}}(j-s+1,c),$$

where ϵ is a small value added to ensure the probability of characters do not become zero and $\mathbb{1}_{\mathrm{i}}(k,c)$ is a variable indicating whether the k-th character of r_i equals c, i.e.

$$\mathbb{1}_{\mathrm{i}}(k,c) = \begin{cases} 1 & \text{if } r_{i,k} = c \\ 0 & otherwise \end{cases}.$$

Finally, the estimation of our parameters θ using the intermediate values in α will be done as following:

$$\theta_{j,c} = \frac{\alpha_{j,c}}{\sum_{i=1}^{4}\alpha_{j,c_i}} \qquad \forall j \quad 1 \le j \le g.$$

Here, c_i denotes each of the four possible nucleotides at position j. Based on this updated hypothesis, we will continue our E- and M-steps until there is a convergence, i.e. the placement positions of reads do not change anymore and thus the hypothesis reaches a fixed set of values.

A simplified simulation of our EM algorithm is presented in Figure 3D. Initially, the probabilities of the nucleotides are equal across all positions as indicated in the top left of the figure. Based on this set of parameters, we calculate the probabilities of each read aligning at each gap position in the E-step and accumulate those probabilities in the M-step to determine an intermediate consensus with updated nucleotide probabilities. The relative height and frequency of nucleotides at different positions in the figure denote the corresponding information content and the relative probabilities at those positions. These two steps then iterate two more times and finally, at the end of the third iteration, we manage to obtain the true placements of the unmapped reads in the gap. The final consensus is constructed based on the final placement of reads using a majority voting approach, and the gap is filled with the predicted final consensus sequence.

2.3 Selecting the gap length

Once the EM converges for a particular gap length g, we compute the likelihood of the estimated parameters as follows:

$$l(\boldsymbol{\theta}_{g,c};\mathcal{R}) = \log\prod_{i=1}^{N} p(r_i|\boldsymbol{\theta}_{g,c}),$$

where $p(r_i|\boldsymbol{\theta}_{g,c})$ is the maximum over all placement probabilities of r_i, i.e.

$$p(r_i|\boldsymbol{\theta}_{g,c}) = \max_{L\in L_i} f_i(L). \tag{1}$$

However, since the gap length is often not known exactly, we iterate over a range of gap lengths and select the gap length that maximizes the above likelihood, i.e. $\max_g l(\boldsymbol{\theta}_{g,c};\mathcal{R})$. For each gap with length g in the scaffold file, we compute the likelihood for the range 0.5–2.5 g, and the gap estimate with the maximum likelihood will be selected as the final gap estimate. We observe that the actual gap length is within the specified range for most of the gaps. However, for long gaps, this becomes computationally expensive. So, we use a heuristic for deciding the range which is described in Supplementary Note S2.3. It is to be noted that gap length can be negative if the sequences preceding and succeeding the gap region overlap.

2.4 Finalizing the gap sequence

At this stage of our pipeline, we will finalize our gap sequence based on an error model and learnt cut-off value. This step is needed because every unmapped read parsed during the preprocessing phase does not truly belong to that gap region due to the fact that one or both ends of a read pair might be very error prone, and the aligner sometimes fails to map these reads to the scaffolds and thus they remain unmapped. So it is essential not to consider such reads in the gap-filling process. To prune these reads out, we perform the following steps. Firstly, we generate an intermediate consensus sequence $\mathcal{C}$ based on the output of the final M-step of our method for the gap length corresponding to the highest likelihood. Then each unmapped read r_i will be slid across the allowable consensus positions based on the insert size, and the error probability of r_i placed on consensus position s will be calculated using the following equation:

$$p(r_i|\mathcal{C}) = \max_{L \in \mathcal{L}} e(L).$$

If $\mathbb{1}_i(k, \mathcal{C}_j)$ denotes whether the consensus character at position j, i.e. $\mathcal{C}_j$ matches with r_{ik} which is the k^{th} character of read r_i, then

$$e(L) = \prod_{j=s,k=1}^{j=s+l-1,k=l} \mathbb{1}_i(k, \mathcal{C}_j)(1 - p_{er}(k) - p_{in}(k) - p_{del}(k)) + (1 - \mathbb{1}_i(k, \mathcal{C}_j))(p_{er}(k)\gamma(\mathcal{C}_j, r_{i,k})),$$

where p_{er}, p_{in} and p_{del} are error, insertion and deletion distributions, respectively, for different read positions calculated using the parsed CIGAR information from the SAM alignment output. If the error probability $p(r_i|\mathcal{C})$ of r_i is less than a cut-off error probability, only then we will consider the read for gap filling, otherwise it will be discarded from consensus consideration. The cut-off probability value is precalculated using the distribution of probabilities of uniquely and fully mapped reads. It is the value above which the probabilities of 80% of such reads lie.

Finally, we place each read above the cut-off threshold value at their most likely position according to Equation (1) and construct a final consensus sequence based on a majority voting approach. This will be our final predicted sequence $\mathcal{G}$ for that particular gap.

2.5 Implementation

The method is implemented using C++ and is available for download freely at https://github.com/SumitTarafder/Figbird under GNU General Public License v3.0. In order to reduce the runtime and ensure accuracy, we apply a number of heuristics in the implementation, which are described in Supplementary Note S2.5. We have also prepared a script to run Figbird iteratively with various libraries in different modes, which is described in Supplementary Note S2.5.

3 Results

3.1 Experimental data and setup

To assess the performance of Figbird and to compare it with existing gap-filling tools, we use the GAGE dataset (Salzberg *et al.*, 2012), which is a standard dataset generated to critically evaluate the genome assemblers and has been used to assess gap-filling tools (Salmela *et al.*, 2016). In this experiment, we use the data for two bacterial species *Staphylococcus aureus*, *Rhodobacter sphaeroides* as well as *Homo sapiens* Chromosome 14, for which reference genomes are available. We collect a wide array genome assemblies for the three datasets generated using various methods as part of the GAGE project as described in Supplementary Table S1. In addition, we have used the SPAdes (Prjibelski *et al.*, 2020) assembler v3.15.1 to generate draft scaffolds for the three genomes specified above and evaluated the performance of the gap-filling tools on them. We perform gap filling using Figbird on all the assemblies and evaluate the results by comparing the filled sequences with the reference sequences.

As part of the GAGE datasets, second-generation sequencing reads from two libraries are available for each of these three datasets. For our experiment, we use both the fragment (paired-end) as well as the short jump (mate pair) libraries. The details about the short-read libraries used in this experiment are listed in Supplementary Table S2. More details about the different assemblies and read sets are available at the official GAGE website. For sequences from the short jump library, we use Quake (Kelley *et al.*, 2010) corrected versions of the reads for better accuracy due to its conservative nature of error correction mechanism (Fujimoto *et al.*, 2014). We run Bowtie2 version 2.2.3 to align these read pairs to the gapped scaffolds for our experiment and then fill the gaps using Figbird v0.1.0.

We compare the performance of our tool Figbird with the four state-of-the-art tools available for filling gaps using short reads which are, SOAPdenovo's stand-alone tool GapCloser v1.12-r6 (Luo *et al.*, 2012), GapFiller v1.10 (Boetzer and Pirovano, 2012), Gap2Seq v1.0 (Salmela *et al.*, 2016) and Sealer (Paulino *et al.*, 2015). For GapFiller, both BWA (Li and Durbin, 2009) and Bowtie (Langmead *et al.*, 2009) aligners are used. We also considered two recent gap-filling tools GAPPadder (Chu *et al.*, 2019) and GapPredict (Chen *et al.*, 2021), but we were unable to run them on most assemblies due to runtime errors and long execution time, respectively.

All experiments are run using 24 cores on a machine with Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70 GHz processors. Our method has been parallelized to allow the filling of individual gaps on separate threads (discussed in Supplementary Note S2.4). We used the Unix command 'time' to measure the time taken for each of these tools and used the Python script Memusg to benchmark peak memory usage. Due to the prohibitive time and memory requirements of Gap2seq, the performance metrics for all the assemblies of Human Chromosome 14 (HC14) except SPAdes have been taken from the bar plots in (Salmela *et al.*, 2016) using the tool WebPlotDigitizer.

3.2 Evaluation criteria

As gap filling is one of the last steps in the genome assembly pipeline, a wrongly introduced sequence may affect subsequent analysis, especially if the gap region falls in coding regions of the genome. Therefore, we focus on errors introduced by gap-filling tools in addition to the amounts of gaps filled by them. We use QUAST v2.3 (Gurevich *et al.*, 2013) to compare the outputs of the tools with the reference. We then use a Python script provided by Salmela *et al.* (2016) for analysis of the results and classification into 'misassemblies' and other errors. QUAST uses NUCmer (Kurtz *et al.*, 2004) to find alignments between the gap-filled assembly and the reference sequence.

To assess the quality of the filled sequence and the robustness of the methods, we use six different metrics, (i) misassemblies, (ii) erroneous length, (iii) unaligned length, (iv) NGA50, (v) number of gaps and (vi) total gap length, which are explained in details in Supplementary Note S2.6.

3.3 Comparison with gap-filling tools

In this section, we present the performance comparison of Figbird with four other state-of-the-art gap-filling tools that use short reads as discussed in Section 3.1. The detailed evaluation results from QUAST are provided in Supplementary Tables S3–S5. Each table shows the percentage increase or decrease in the six evaluation metrics achieved by the five gap-filling tools along with the original values before gap filling. For each assembly, these relative percentages are determined using the differences in evaluated values between original assembly and gap-filled assembly using QUAST. The results over all the assemblies are presented in the bottom of the tables in rows named 'Average', which is obtained by determining the average for a particular metric over all the assemblies for each gap-filling tool. The standard deviations in the percent averages are also shown in the tables. The overall percent reduction in gap length as well as percent changes in misassembly and errors for all three datasets are also shown in Figure 4. We focus on these three metrics as unaligned length and NGA50 are related to these, while filling a gap partially

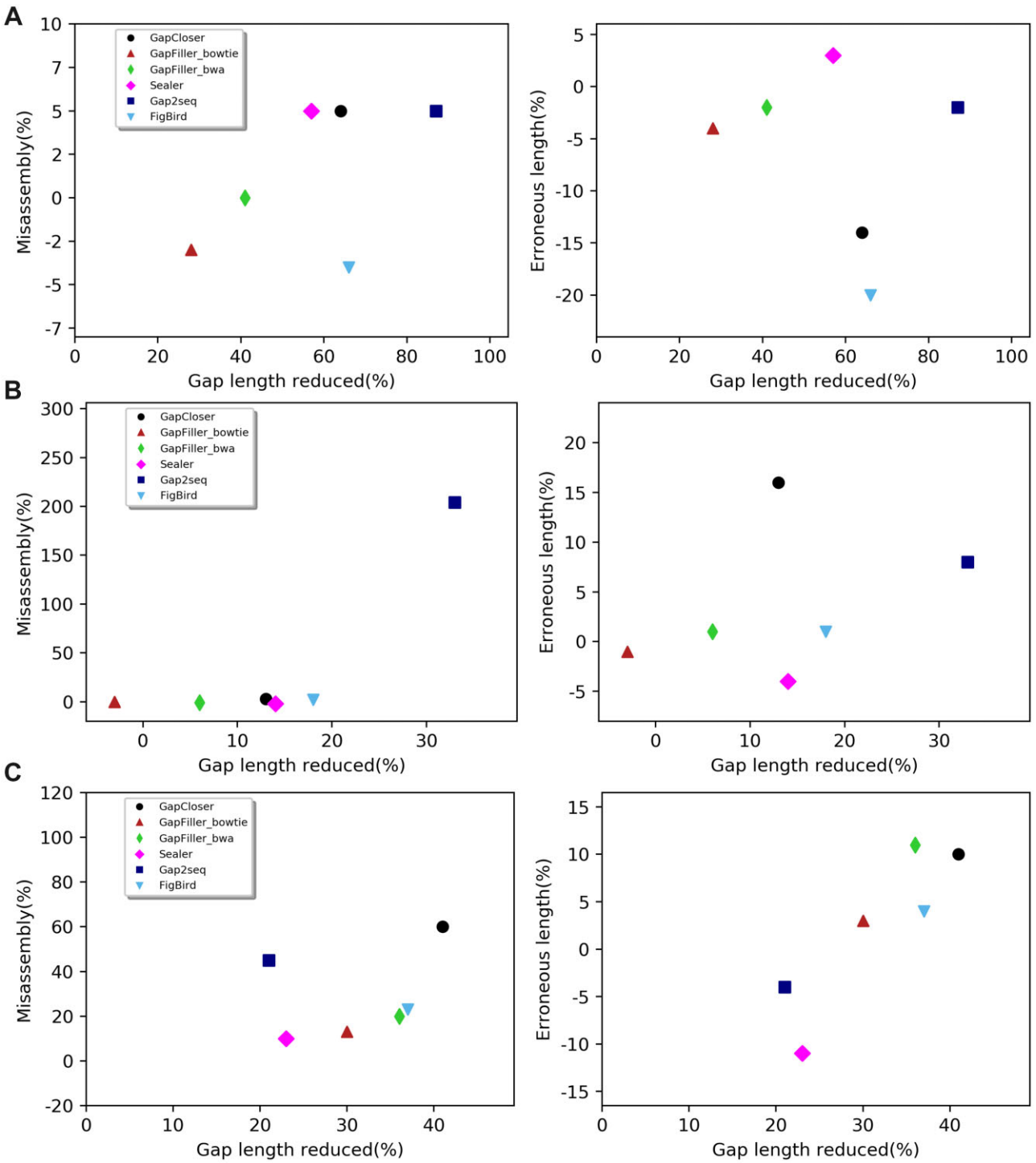

Fig. 4. Scatter plots showing the performance of Figbird compared to other gap-filling tools on (**A**) *S.aureus*, (**B**) *R.sphaeroides* and (**C**) HC14 datasets in terms of the average percentage of misassemblies and erroneous length against the average percentage of nucleotides filled by each tool. The values are computed by determining the percent change in misassembly and erroneous length, and percent reduction in gap length obtained by each tool for all the *de novo* draft genome assemblies from GAGE and SPAdes, and then averaging over the values for all those assemblies

may lead to an increase in the number of gaps which we believe is misleading. From the overall results, we observe that Figbird is able to close considerably high portion of gap regions yet manage to keep the erroneous length and misassembly low compared to the other state-of-the-art tools.

For the *S.aureus* dataset, we observe from the scatter plots in Figure 4A that none of the other tools have managed to fill more nucleotides than Figbird while achieving better accuracy in terms of error or misassembly. There are no points in the scatter plots that are to the right and below the points representing Figbird i.e. it is Pareto optimal. We can also see from Supplementary Table S3 that Figbird has reduced the total misassembly rate by 4%, which is the best among all other tools, and reduced erroneous length by 20% thus outperforming the next best tool GapCloser in this respect by

6%. The only tool which is able fill a higher portion of gaps than Figbird is Gap2Seq, which is at the expense of higher rate of misassembly and substantially higher amount of erroneous sequences.

In the case of our second bacterial dataset *R.Sphaeroides*, a similar trend can be observed from Figure 4B like the previous dataset. None of the other tools have outperformed Figbird, which has managed to close the second-highest amount of 'N's, only behind Gap2Seq. However, Gap2Seq increases misassembly and erroneous length by 204% and 8%, respectively, which are substantially higher than the 2% misassembly and 1% erroneous length increase by Figbird, as shown in Supplementary Table S4. It is worth highlighting that the sequencing error rate for *R.sphaeroides* dataset is higher compared to that of *S.aureus* dataset and the sequencing reads being more error-prone, mapping tools find it difficult to align reads properly to the scaffolds (Hunt *et al.*, 2014), and also leads to errors during gap filling. This shows that Gap2Seq lacks robustness to sequencing errors and may introduce a large amount of errors, whereas Figbird still performs in a balanced way without introducing massive amounts of misassemblies and errors.

Finally, detailed results of QUAST evaluation on the HC14 dataset are summarized in Supplementary Table S5. Despite the highly repetitive nature of the HC14 dataset as suggested in Luo *et al.* (2012), three of the tools, Figbird, GapCloser and GapFiiler filled more than 30% nucleotides. From the overall comparison presented in Figure 4C, it can be observed that Figbird is second to GapCloser in terms of the amount of gap filled. However, the additional 4% gap filled by GapCloser comes at the expense of 37% and 6% more misassemblies and erroneous length, respectively, compared to Figbird. Moreover, GapCloser fills a smaller amount of gaps while making more misassemblies and errors than Figbird in the other datasets. On the other hand, Gap2Seq, which is able to reduce the gap length by the highest amount in the other two datasets, fills 16% fewer gaps than Figbird despite 22% more misassemblies.

Overall, our EM-based method shows Pareto optimal performance with respect to amount of gap filled, misassemblies and erroneous sequence introduced in all three datasets, i.e. no other tool is able to fill more nucleotides while making less misassemblies or errors than Figbird. Figbird has achieved best or near-best performance score for all the evaluation metrics in every dataset used in evaluation suggesting the usefulness of our approach in gap filling. However, it is worth noting that Sealer outperforms Figbird while filling gaps in assemblies generated by the widely used genome assembler SPAdes.

3.4 Time and memory usage

We have compared the performance of Figbird in terms of run time and peak memory usage with four other tools mentioned in Section 3.1. The detailed commands and parameters used to run each of these tools can be found in Supplementary Note S2.7. Table 1 shows the time and memory usage of the tools on the HC14 dataset. The comparisons on the other two datasets are summarized in Supplementary Tables S6 and S7. From Table 1, we can see that GapCloser and Sealer are the two fastest tools in terms of run time across all the assemblies while Gap2Seq is the slowest among all. Figbird and GapFiller-bwa have moderate time requirements. In the case of memory usage, Gap2seq and Sealer require the highest amount of memory due to their problem formulation, whereas GapFiller requires the lowest. The memory requirement of Figbird is almost similar to that of GapCloser, taking less memory in case of Allpaths-LG, CABOG, MSR-CA, etc. while taking slightly more memory in case of Velvet, ABySS, etc. Overall, we find that although there are gap-filling tools with lower time and memory usage than Figbird, it falls within the time and memory requirement range of the state-of-the-art tools. The increased run time can be regarded as a trade-off with the improved performance. Moreover, the running time of Figbird scale linearly with the number and lengths of gaps, and the number of reads (Supplementary Note S2.9) making it applicable to large datasets unlike some of the other gap-filling tools. To further scale our method to large genomes, several future directions for improvement are possible. The majority of the time taken by Figbird is in predicting the length of the gap. Although we have implemented an approach to make it scalable (described in Supplementary Note S2.3), the approach can be redesigned by gaining a preliminary idea about the gap length through graph-based approaches or gap size estimation methods (Chapman *et al.*, 2011; Rahman and Pachter, 2021; Sahlin *et al.*, 2012), and then exploring the range with our likelihood estimation model to identify the correct gap length. Secondly, as we are placing the unmapped reads in every possible gap position in each EM iteration, the probable position of those reads can be stored to minimize searches in subsequent iterations.

4 Conclusion

In this paper, we presented a probabilistic method based on the EM algorithm to fill gaps in genome assemblies using paired-end and mate-pair reads from second-generation sequencing technology with relatively low error rate. As gap filling is one of the last stages in genome assembly pipelines and the comparatively easy regions of the genome have already been reconstructed by de novo assemblers in previous stages, only complex regions are left to fill up at this stage. So, the main objective was to incorporate essential information such as insert size distribution, sequencing errors, etc. to correctly estimate the true lengths of gaps and fill them with the introduction of a minimal amount of errors making downstream genome analysis easier. The results from experiments on multiple

Table 1. Gap-closing performance of all the tools on 10 draft genome assemblies of HC14

Assembly	Gap-filling software									
	Time (h)					Memory (GB)				
	GapCloser	GapFiller-bwa	Sealer	Gap2Seq[a]	Figbird	GapCloser	GapFiller-bwa	Sealer	Gap2Seq[a]	Figbird
ABySS	0.14	1.7	0.3	55.2	2.2	6.5	0.11	40	22	7.5
ABySS2	0.16	2.9	0.35	155.5	3.5	6.4	0.14	40	17	7.2
Allpaths-LG	24	6.3	0.55	4×10^3	8	8.2	0.17	40	21	5.5
Bambus2	0.19	13.2	0.31	1×10^4	17	6.8	0.12	40	25	5.4
CABOG	0.08	4.4	0.45	167	5.4	6.5	0.11	40	14	5.3
MSR-CA	0.4	13.7	0.9	6×10^3	29	7.6	0.14	40	23	6.1
SGA	0.3	13.1	0.6	14×10^3	32	6.5	0.11	40	23	8.9
SOAPdenovo	0.25	5.2	0.7	1×10^4	9	6.8	0.3	40	25	7.1
SPAdes	0.1	0.4	0.5	2.2	1	6.7	0.15	40	33	0.7
Velvet	1.2	41.7	1.1	2×10^4	38	7.7	0.2	40	27	11.5

[a]The performance metric values for Gap2seq on all the assemblies except SPAdes are values taken from bar plots in the corresponding paper (Salmela *et al.*, 2016) using WebPlotDigitizer: Version 4.5 as the tool could not be evaluated due to the high run time and memory constraints.

real datasets show that our method achieves a balanced performance across a variety of assembly pipelines managing to close a larger number of gaps improving the overall contiguity while still managing to keep the amount of misassembly and the length of erroneously introduced sequence low compared to existing tools. Specifically, it demonstrates Pareto optimal performance in terms of the portion of gap filled and the numbers of misassemblies and errors introduced overall, although there are gap filling tools better suited to specific genome assemblers.

We believe our method for gap sequence prediction in draft genome assemblies can be adapted to other applications. For example, long reads with high error rates are often corrected with the help of short reads through hybrid error correction techniques such as Ratatosk (Holley *et al.*, 2021). Error models for short and long reads may be used to find the most probable paths corresponding to long reads in de Bruijn graphs constructed from short reads. This may improve error correction in long reads generated from near-identical repeats. Furthermore, there are still a number of medically relevant genes that are challenging to analyze due to their presence in complex repetitive and hard-to-assess regions (Wagner *et al.*, 2022). An EM-based approach may also improve the accuracy of such variant calls in repeat-rich regions and resolve short tandem repeats more accurately in human and other genomes. Finally, another interesting future direction is to explore whether this approach to gap filling can be extended to third-generation sequencing data to accurately fill gaps that cannot be done using only second-generation reads.

Acknowledgements

The authors thank Lior Pachter for providing feedback and computational resources to perform some of the experiments. The authors also thank the reviewers for their comments.

Data availability

The data underlying this article are from the GAGE (Genome Assembly Gold-standard Evaluations) study (Salzberg *et al.*, 2012). The datasets were derived from sources in the public domain: https://gage.cbcb.umd.edu/.

Funding

This research work is partially supported by the Institute of Advanced Research (IAR) of United International University (UIU), Dhaka, Bangladesh under the Research Grant UIU-RG-162002.

Conflict of Interest: The authors declare that they do not have any conflict of interest.

References

Bailey,T.L. and Elkan,C. (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learn.*, **21**, 51–80.

Boetzer,M. and Pirovano,W. (2012) Toward almost closed genomes with GapFiller. *Genome Biol.*, **13**, R56.

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

Chaisson,M.J. *et al.* (2015) Genetic variation and the de novo assembly of human genomes. *Nat. Rev. Genet.*, **16**, 627–640.

Chapman,J.A. *et al.* (2011) Meraculous: de novo genome assembly with short paired-end reads. *PLoS One*, **6**, e23501.

Chen,E. *et al.* (2021) GapPredict—a language model for resolving gaps in draft genome assemblies. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **18**, 2802–2808.

Chu, C. *et al.* (2019) GAPPadder: a sensitive approach for closing gaps on draft genomes with short sequence reads. *BMC Genomics*, **20**, 1–10.

Dempster, A. P. *et al.* (1977) Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc. B (Methodol.)*, **39**, 1–38.

Domanska, D. *et al.* (2018) Mind the gaps: overlooking inaccessible regions confounds statistical testing in genome analysis. *BMC Bioinformatics*, **19**, 1–9.

English,A.C. *et al.* (2012) Mind the gap: upgrading genomes with Pacific Biosciences RS long-read sequencing technology. *PLoS One*, **7**, e47768.

Frank,J.A. *et al.* (2016) Improved metagenome assemblies and taxonomic binning using long-read circular consensus sequence data. *Sci. Rep.*, **6**, 1–10.

Fujimoto, M. S. *et al.* (2014) Effects of error-correction of heterozygous next-generation sequencing data. *BMC Bioinformatics*, **15**, 1–8.

Gurevich,A. *et al.* (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

Holley, G. *et al.* (2021), Ratatosk: hybrid error correction of long reads enables accurate variant calling and assembly. *Genome Biol.*, **22**, 1–22.

Hunt, M. *et al.* (2014) A comprehensive evaluation of assembly scaffolding tools. *Genome Biol.*, **15**, R42.

Kammonen,J.I. *et al.* (2019) gapFinisher: a reliable gap filling pipeline for SSPACE-LongRead scaffolder output. *PLoS One*, **14**, e0216885.

Kelley,D.R. *et al.* (2010) Quake: quality-aware detection and correction of sequencing errors. *Genome Biol.*, **11**, R116.

Kosugi,S. *et al.* (2015) GMcloser: closing gaps in assemblies accurately with a likelihood-based selection of contig or long-read alignments. *Bioinformatics*, **31**, 3733–3741.

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

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, 1–10.

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

Lu,P. *et al.* (2020) PGcloser: fast parallel gap-closing tool using long-reads or contigs to fill gaps in genomes. *Evol. Bioinform. Online*, **16**, 1176934320913859.

Luo, R. *et al.* (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, **1**, 18.

Meyer, M. and Kircher, M. (2010) Illumina sequencing library preparation for highly multiplexed target capture and sequencing. *Cold Spring Harb. Protoc.*, **2010**, pdb.prot5448.

Pachter,L. (2011) Models for transcript quantification from RNA-Seq. *arXiv preprint arXiv:1104.3889*. https://doi.org/10.48550/ARXIV.1104.3889.

Paulino, D. *et al.* (2015) Sealer: a scalable gap-closing application for finishing draft genomes. *BMC Bioinformatics*, **16**, 1–8.

Prjibelski,A. *et al.* (2020) Using SPAdes de novo assembler. *Curr. Protoc. Bioinformatics*, **70**, e102.

Rahman,A. and Pachter,L. (2013) CGAL: computing genome assembly likelihoods. *Genome Biol.*, **14**, R8.

Rahman,A. and Pachter,L. (2021) SWALO: scaffolding with assembly likelihood optimization. *Nucleic Acids Res.*, **49**, e117.

Sahlin,K. *et al.* (2012) Improved gap size estimation for scaffolding algorithms. *Bioinformatics*, **28**, 2215–2222.

Salmela,L. *et al.* (2016) Gap filling as exact path length problem. *J. Comput. Biol.*, **23**, 347–361.

Salzberg,S.L. *et al.* (2012) GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.

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

Thomma,B.P. *et al.* (2016) Mind the gap; seven reasons to close fragmented genome assemblies. *Fungal Genet. Biol.*, **90**, 24–30.

Vandervalk,B.P. *et al.* (2014) Konnector: connecting paired-end reads using a bloom filter de bruijn graph. In: *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Belfast, UK*, IEEE. pp. 51–58.

Wagner,J. *et al.* (2022) Curated variation benchmarks for challenging medically relevant autosomal genes. *Nat. Biotechnol.*, **40**, 672–680.

Xu,G.-C. *et al.* (2019) LR_Gapcloser: a tiling path-based gap closer that uses long reads to complete genome assembly. *GigaScience*, **8**, giy157.

Xu,M. *et al.* (2020) TGS-GapCloser: a fast and accurate gap closer for large genomes with low coverage of error-prone long reads. *GigaScience*, **9**, giaa094.

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