Briefings in Bioinformatics, 23(1), 2022, 1–12

https://doi.org/10.1093/bib/bbab405
Problem Solving Protocol

BlockPolish: accurate polishing of long-read assembly via block divide-and-conquer

Neng Huang, Fan Nie, Peng Ni, Xin Gao, Feng Luo and Jianxin Wang

Corresponding authors: Feng Luo, School of Computing, Clemson University, South Carolina, 29634, USA. E-mail: luofeng@clemson.edu; Jianxin Wang, Hunan Provincial Key Lab on Bioinformatics, Central South University, Changsha, Hunan 410083, P.R. China; School of Computer Science and Engineering, Central South University, Changsha, Hunan 410083, P.R. China. E-mail: jxwang@mail.csu.edu.cn

Abstract

Long-read sequencing technology enables significant progress in *de novo* genome assembly. However, the high error rate and the wide error distribution of raw reads result in a large number of errors in the assembly. Polishing is a procedure to fix errors in the draft assembly and improve the reliability of genomic analysis. However, existing methods treat all the regions of the assembly equally while there are fundamental differences between the error distributions of these regions. How to achieve very high accuracy in genome assembly is still a challenging problem. Motivated by the uneven errors in different regions of the assembly, we propose a novel polishing workflow named BlockPolish. In this method, we divide contigs into blocks with low complexity and high complexity according to statistics of aligned nucleotide bases. Multiple sequence alignment is applied to realign raw reads in complex blocks and optimize the alignment result. Due to the different distributions of error rates in trivial and complex blocks, two multitask bidirectional Long short-term memory (LSTM) networks are proposed to predict the consensus sequences. In the whole-genome assemblies of NA12878 assembled by Wtdbg2 and Flye using Nanopore data, BlockPolish has a higher polishing accuracy than other state-of-the-arts including Racon, Medaka and MarginPolish & HELEN. In all assemblies, errors are predominantly indels and BlockPolish has a good performance in correcting them. In addition to the Nanopore assemblies, we further demonstrate that BlockPolish can also reduce the errors in the PacBio assemblies. The source code of BlockPolish is freely available on Github (https://github.com/huangnengCSU/BlockPolish).

Key words: long reads; assembly polishing; block divide-and-conquer; neural network.

Introduction

Long-read sequencing technology, such as Oxford Nanopore Technologies and Pacific Biosciences, can produce long reads up to 10^6 bases. The long reads have dramatically advanced the genomics research field [1–4] because they can span repetitive regions and anchor repeat copies to a unique position in the genome. However, except for PacBio HiFi reads, the other long reads have high error rates. In particular, with the development of chemistry and base callers [5–7], Oxford Nanopore sequencing still has read error rates in the range of 5–15%. The errors in the reads usually lead to errors in assembly, which cause the downstream genome analysis [8–10] to be unreliable. In a *de novo*

Neng Huang is a PhD candidate in School of Computer Science and Engineering, Central South University, China. His current research interests include long reads correction.
Fan Nie is a PhD candidate in School of Computer Science and Engineering, Central South University, China. His current research interests include long-reads assembly.
Peng Ni is a PhD candidate in School of Computer Science and Engineering, Central South University, China. His current research interests include DNA methylation detection.
Xin Gao is the acting associate director of Computational Bioscience Research Center and a professor in School of Computer Science, King Abdullah University of Science and Technology, Saudi Arabia. His research lies at the intersection between computer science and biology.
Feng Luo is a professor in School of Computing, Clemson University, USA. His research areas include deep learning and application, high-throughput biological data analysis, data-intensive bioinformatics, network biology, computational genomics and genetics.
Jianxin Wang is the dean and a professor in the School of Computer Science and Engineering, Central South University, China, and a professor in HuNan Provincial Key Lab on Bioinformatics, Central South University, China. His current research interests include bioinformatics and algorithms.
Submitted: 19 July 2021; **Received (in revised form):** 13 August 2021

genome assembly, one can perform read error correction at the beginning of the process, or use reads-to-assembly alignment to improve the quality of assembly at the end of the process. The latter approach is called polishing.

The two strategies currently used for long-reads error correction are 'hybrid methods' and 'non-hybrid methods'. The hybrid error correction methods, including Hercules [11], HALC [12], Nanocorr [13], Nas [14], proovread [15], MaSuRCA [16] and PacBioToCA [17], take advantage of high accuracy of short reads. Errors in long reads are corrected based on the alignment between short reads and long reads. An obvious requirement of hybrid error correction methods is to have both short-read and long-read sequencing data of the same sample. The other strategy performs self-correction using long reads alone. The non-hybrid methods including FLAS [18], LoRMA [19], Canu error correction module [20] usually generate consensus sequences by constructing a graph representation from pairwise alignment of raw long reads.

However, the pairwise alignment in reads error correction is a time-consuming step accounting for around one-third of the entire assembly process. The latest assemblers such as Flye [21], Wtdbg2 [22] and Shasta [9] avoid the error correction step and directly assemble from erroneous raw reads and finally use a polishing workflow to increase the accuracy. Polishing is a step to correct errors in assemblies, and its performance directly affects the quality of the whole assembly result. The genome assembly polishing methods can be divided into three main categories. The first type of polishing tools, such as Pilon [23], NextPolish [24], ntEdit [25] and POLCA [26], aligns highly accurate short reads to the assembly and corrects the assembly result from reads alignment. These approaches have high polishing accuracy but require extra next-generation sequencing (NGS) data for the same sample. In addition, short-read polishing can be inaccurate in some repeats. The second type of polishing tools combines long reads and short reads for correcting errors in the assembly. Apollo [27] uses read sets from all available sequencing technologies to improve the accuracy of assembly with a profile hidden Markov model. However, Apollo's polishing speed is too slow, which makes it impractical for large genomes. The third type of polishing tools only requires long reads and usually has a fast speed. As an example, Racon [28] has a highly efficient polishing module. A partial-order alignment (POA) graph [29] is constructed according to the reads-to-assembly alignment, and then a path with the maximum score is found as the consensus sequence in the POA graph through dynamic programming. Medaka (https://github.com/nanoporetech/medaka) is developed by Oxford Nanopore Technologies (ONT) for creating consensus sequences and variant calls from Nanopore sequencing data. For each column of a pileup of sequencing reads against a draft assembly, Medaka counts the distribution of different bases and a recurrent neural network is used to predict the true nucleotide base at each position. In the workflow of Medaka, the draft assembly is polished by four rounds of Racon followed by one round of Medaka. NeuralPolish [30] is a polishing method based on alignment matrix construction and orthogonal Bi-directional Gated Recurrent Unit (Bi-GRU) networks. In the alignment matrix, each row stores a mapped read and each column represents the aligned bases at each position of the contig. Two orthogonal bidirectional GRU networks process the matrix by row and column, respectively, and predict the final polished sequence. MarginPolish & HELEN [9] is also a deep neural network-based polishing pipeline for improving the base-level quality of draft assembly. The pipeline is made up of two modules, graph-based MarginPolish and deep neural network-based HELEN. MarginPolish constructs a weighted run-length encoding (RLE) POA graph to represent the potential alternative local assemblies from RLE alignment of reads against the assembly. HELEN takes the weights of the MarginPolish RLE POA graph and uses a multitask recurrent neural network to predict a nucleotide base and the number of repetitions for each base.

The number of different nucleotide bases at each position from the reads-to-assembly alignment result is an important feature to determine the consensus sequence. However, since the inserted fragments from multiple reads are not aligned, we usually can not get an accurate number of nucleotide bases in the insertion region. Frequent insertions make the reads-to-assembly alignment more complex and reduce the polishing accuracy. Thus, it is necessary to improve the alignment of inserted fragments. However, the state-of-the-art polishing tools do not take this fact into consideration, which limits the polishing accuracy. In this paper, we present a novel polishing approach, BlockPolish, which only uses error-prone raw long reads. In BlockPolish, we divide each contig into trivial blocks where there are few insertions and complex blocks where there are frequent insertions. For complex blocks, we realign the read fragments to improve the alignment result using multiple sequence alignment (MSA) [31]. Then, two neural network models with different parameters are applied to trivial blocks and complex blocks, respectively. In the neural network, we apply a two-task, four-layer Bi-LSTM network to predict compressed consensus sequence and full-length consensus sequence. The task predicting compressed sequence only learns the order of different nucleotide bases that appear in the consensus sequence. Another task predicting full-length sequence learns not only the order of nucleotide bases but also the number of repetitions of each nucleotide base. The task predicting compressed sequence can help train the task predicting full-length consensus sequence. During training, two tasks are jointly trained. During inference, the final consensus sequence comes from the output of the task predicting full-length sequence. We compare BlockPolish with Racon, Medaka and MarginPolish & HELEN on the human sample NA12878. BlockPolish achieves more accurate assemblies than other polishing algorithms, and the genome assembly corrected by BlockPolish has higher gene completeness. For each assembly, the errors are dominated by indels, and BlockPolish has a significant improvement in solving indels compared with other tools. We also test BlockPolish on the assemblies produced by different base-callers and different assemblers, BlockPolish shows a stable ability to improve the accuracy of assembly results.

Methods

Training data set and testing data set

We train the BlockPolish model on Nanopore reads of the human sample HG002 chromosome 6–10. The raw reads of HG002 are basecalled with Guppy v3.4.4 and the ground-truth HG002 assembly is generated from Genome in a Bottle Consortium (GIAB) high-confidence variant set [32] against the GRCh38 reference sequence. Both basecalled reads and truth assembly are released on website https://console.cloud.google.com/storage/browser/kishwar-helen. We assemble the raw reads with Wtdbg2 and polish the initial assembly with one round of Racon because graph-based polishing can help to solve long-length structural errors in the draft assembly. Two state-of-the-art Nanopore assembly polishing workflows, Medaka and MarginPolish & HELEN, both adopt the similar framework, a

Table 1. Details of data sets

Data set	Accession	Detail
HG002-ONT	https://console.cloud.google.com/storage/browser/kishwar-helen	50X coverage, basecalled by Guppy 3.4.4
HG002-PacBio	https://humanpangenome.org/hg002/	44X coverage
HG002-Ground Truth	https://console.cloud.google.com/storage/browser/kishwar-helen	–
NA12878-ONT	https://github.com/nanopore-wgs-consortium/NA12878	35X coverage, basecalled by rgb 0.141,0.161,0.18Guppy 2.3.8 HAC
NA12878-Ground Truth	GenBank GCA_002077035.3	–
E. coli-PacBio	http://sysbio.gzzoc.com/MECAT/	84X coverage
E. coli-Ground Truth	Genbank GCF_000005845.2	–
S. cerevisiae-PacBio	http://sysbio.gzzoc.com/MECAT/	115X coverage
S. cerevisiae-Ground Truth	Genbank GCF_000146045.2	–
A. thaliana-PacBio	http://sysbio.gzzoc.com/MECAT/	155X coverage
A. thaliana-Ground Truth	Genbank GCA_000835945.1	–
D. melanogaster-PacBio	http://sysbio.gzzoc.com/MECAT/	110X coverage
D. melanogaster-Ground Truth	Genbank GCF_000001215.4	–

graph-based polishing followed by a neural network-based polishing. Then, the assembly is aligned to the truth assembly with minimap2 to obtain the label sequence. We divide the assembly results into trivial blocks and complex blocks, and form two training data sets. To evaluate the performance of different polishing tools, we use another individual human sample NA12878 as the testing data set. The Nanopore sequencing data of NA12878(rel6) is posted online at https://github.com/nanopore-wgs-consortium/NA12878. The truth assembly is from the publicly available PacBio high-quality assembly NA12878_prelim_3.0, which was assembled using Falcon [33] and corrected using Quiver [34] and Pilon. The NA12878_prelim_3.0 can be downloaded from GenBank with accession number GCA_002077035.3. All of the raw data and ground truths are available from public websites as shown in Table 1.

Overview of BlockPolish workflow

As shown in Figure 1A, the first few steps of BlockPolish are similar to Medaka. BlockPolish performs one round of Racon as the initial refinement for draft assembly. Then, the raw reads are aligned to the assembly with minimap2 [35] to obtain the read-to-assembly alignment. Besides minimap2, BlockPolish supports the other long-read aligners as long as the reads-to-assembly alignment results are stored in the BAM file format. After the two steps, BlockPolish scans the alignment result by column and counts the number of different nucleotide bases, insertions and deletions at each position of the contig to construct a consensus table [36]. According to the consensus table, each contig is divided into trivial blocks and complex blocks. To optimize the alignment result in complex blocks, an MSA is performed to realign the raw reads in complex blocks. Afterwards, the feature tensors including the number of different nucleotide bases, insertions and deletions extracted from trivial blocks and rearranged complex blocks are fed into two neural network models separately to generate consensus sequences of each block. Finally, BlockPolish threads the consensus sequences of blocks based on the positions in the contig and forms the polished sequence.

Constructing a consensus table and dividing blocks

At one position of the contig, there may be multiple reads with inserted fragments. However, there is no template sequence that can be used to map all inserted fragments to this template. The inserted fragments from multiple reads are not aligned in the reads-to-assembly alignment. Thus, frequent insertions can make the alignment result more complex and reduce the polishing accuracy. In order to divide the blocks on the contig with different alignment complexity, we construct a linear table to store the number of different nucleotide bases, insertions and deletions at each position. The number of different bases represents the statistics of the nucleotide bases from multiple reads at the current position of the contig. The number of insertions indicates how many reads have inserted fragments at the current position of the contig. The number of deletions indicates how many reads show nucleotide missing at the current position of the contig.

In this paper, we represent the reads-to-assembly alignment as follows:

$$\begin{cases} C = & (c_1, c_2, \dots c_j, \dots, c_{l_0}), \\ A_1 = & (a_{1,1}, a_{1,2}, \dots, a_{1,j}, \dots a_{1,l_1}), \\ A_2 = & (a_{2,1}, a_{2,2}, \dots, a_{2,j}, \dots a_{2,l_2}), \\ & \dots \\ A_i = & (a_{i,1}, a_{i,2}, \dots, a_{i,j}, \dots a_{i,l_i}), \\ & \dots \\ A_m = & (a_{m,1}, a_{m,2}, \dots, a_{m,j}, \dots a_{m,l_m}). \end{cases} \quad (1)$$

Here, C represents the contig in the alignment, and $A = \{A_1, A_2, \dots, A_m\}$ represents the reads aligned to the contig. m is the number of aligned reads, l_0 is the length of the contig in the alignment and $l_1, l_2, \dots, l_m$ are lengths of the aligned reads. Each element from the contig C or aligned reads A in the alignment comes from the set of symbols $\Sigma = \{A, C, G, T, *\}$. If $c_j = *$ and $a_{i,j} \neq *$, there is an inserted base in read A_i according to the contig. On the contrary, if $c_j \neq *$ and $a_{i,j} = *$, the read A_i shows a base deletion. The insertion region is a closed interval of the form $[h, k]$ such that $\forall j \in [h, k], c_j = *$.

$F = (f_1, f_2, \dots f_j, \dots, f_{l_0})$ is a sequence that marks each position in the alignment as a trivial or complex position. For position j in the alignment, we define the trivial position ($f_j = 0$) as the one meeting the following two criteria: (i) there is no significant insertion occurring ($I_j < \alpha$). If position j is in an insertion region $[h, k]$, $I_j = \max I_p, p \in [h, k]$ and $I_p = \| \{i | a_{i,p} \neq *, c_p = *, i \in \{1, 2, \dots, m\}\} \|$. If position j is not in any insertion region, $I_j = 0$; (ii) the position is consistent matches ($(M_j/(M_j + D_j) > \beta)$ or consistent deletions ($(D_j/(M_j + D_j) > \gamma)$, $M_j = \| \{i | a_{i,j} \neq *, i \in \{1, 2, \dots, m\}\} \|$, $D_j = \| \{i | a_{i,j} = *, i \in \{1, 2, \dots, m\}\} \|$.

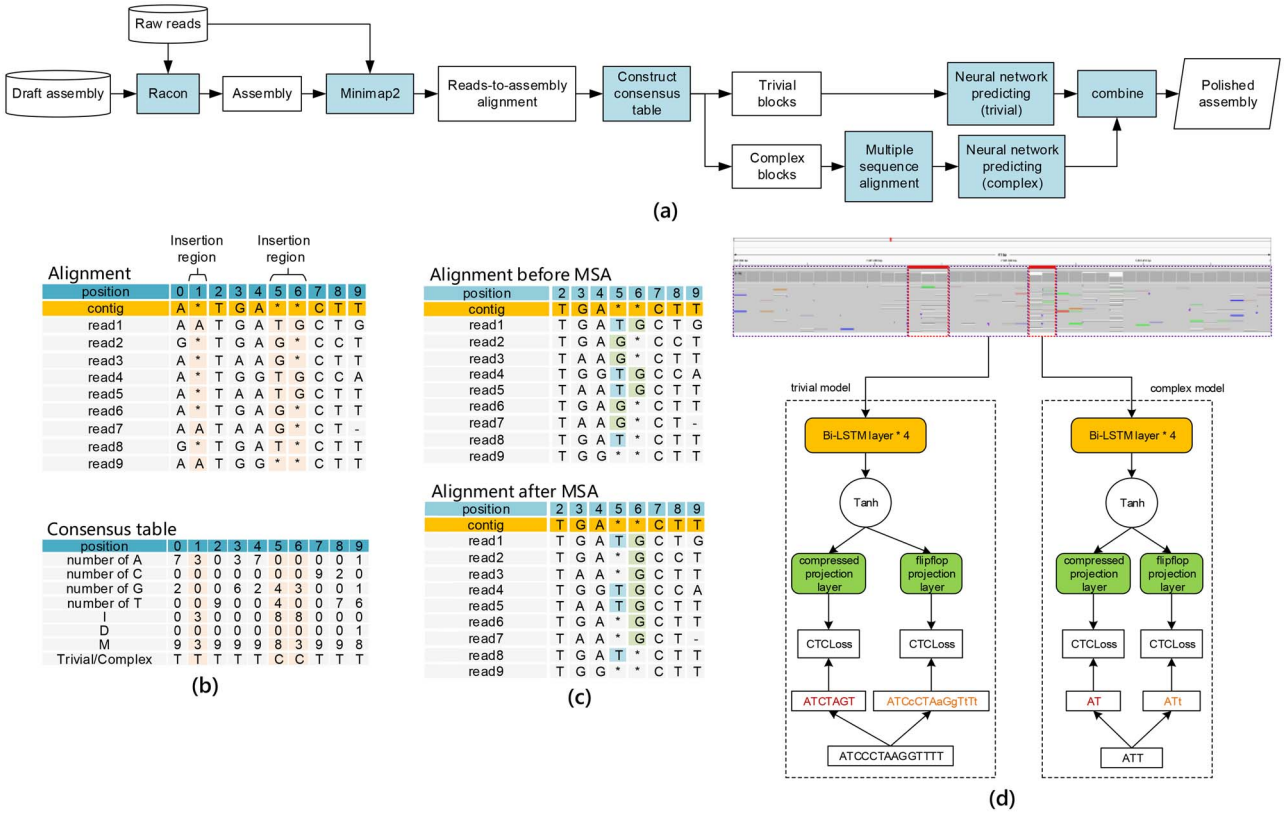

Figure 1. (**A**) Overview of the BlockPolish workflow. The inputs consist of a draft assembly and raw long reads. First, the draft assembly is polished with one round of Racon and the raw long reads are aligned to the assembly to get the reads-to-assembly alignment. Then, we construct a consensus table and divide each contig into blocks with low complexity and high complexity. The complex blocks are processed with a multiple sequence alignment to improve the alignment consistency. Afterwards, two neural network models are used to predict consensus sequences of the trivial blocks and complex blocks, separately. Finally, consensus sequences from all blocks are concatenated into the polished assembly. (**B**) The construction of a consensus table from reads-to-assembly alignment. The table records the number of different nucleotide bases, insertions (*I*) and deletions (*D*) at each position of the contig. *M* represents the total number of different nucleotide bases. (**C**) The reads-to-assembly alignment of a complex region before and after multiple sequence alignment. (**D**) The neural network architecture of BlockPolish. The network of BlockPolish consists of four Bi-LSTM layers, a compressed projection layer and a flip-flop projection layer. The Bi-LSTM layers take into both left and right alignment features when making decisions at each position of the contig. The compression layer is to learn the compressed consensus sequence without continuously repetitive nucleotide bases and the flip-flop is used to learn the full-length consensus sequence.

Here α, β and γ are three thresholds, and we set $\alpha = 6$, $\beta = 0.8$ and $\gamma = 0.8$. The position that does not meet the criteria of a trivial position is deemed as a complex position ($f_j = 1$). For each contig, we have divided each position into a trivial position or a complex position. Then, we can define the trivial blocks and complex blocks. The trivial block is a closed interval of the form $[u, v]$ such that $\forall j \in [u, v], f_j = 0$. Similarly, the complex block is a closed interval of the form $[u, v]$ such that $\forall j \in [u, v], f_j = 1$. For example, as shown in the upper part of Figure 1B, there are nine reads aligned to the contig. There are two insertion regions in the alignment of the format $[1, 1]$ and $[5, 6]$. At the positions in the insertion regions, the number of insertions, deletions and different bases are $I_1 = 3, D_1 = 0, M_1 = 3, I_5 = 8, D_5 = 0, M_5 = 8, I_6 = 8, D_6 = 0, M_6 = 3$. So, position one is a trivial position, and positions five and six are complex positions. Finally, the intervals $[0, 4]$ and $[7, 9]$ are trivial blocks, and the interval $[5, 6]$ is a complex block.

We test different values of thresholds and evaluate the performance of BlockPolish on the data set of HG002 chromosome 1. In Figure 2, as we progressively increase the value of α and decrease the value of β and the value of γ, the time consumption of the polishing process decreases and the error rate gradually increases. When we set $\alpha = 1$, $\beta = 0.85$ and $\gamma = 0.85$, most regions on the contig are divided into complex blocks. We perform MSA in each block of the complex blocks, so the polishing process takes the longest time, but the polishing result has the lowest error rate. When we set $\alpha = 26$, $\beta = 0.6$ and $\gamma = 0.6$, the constraints of trivial blocks are relaxed. Some low-quality regions from complex blocks are divided into trivial blocks. These regions need MSA to improve the alignment results during polishing, but there is no MSA in the polishing process of trivial blocks. Thus, the error rate of polishing results increases. For the trade-off between time consumption and accuracy, the thresholds in BlockPolish are $\alpha = 6$, $\beta = 0.8$ and $\gamma = 0.8$.

To understand the characteristics of trivial blocks and complex blocks, we construct consensus tables from the reads-to-assembly alignment of HG002 chromosome 6–10 and generate trivial blocks and complex blocks. Then, we extract the label sequence of each block from reference-to-assembly alignment. To calculate the error rates of trivial blocks and complex blocks, we align the contig sequence of each block to the label sequence. From the distribution of length in Figure 3, we can see that the lengths of trivial blocks range from 1 to 100 bp, whereas the lengths of complex blocks are concentrated within 10 bp. Since the lengths of complex blocks are relatively small, the computation of MSA in complex blocks does not take a long time.

Table 2. The error rates of trivial blocks and complex blocks on the data of HG002 chromosome 6–10.

Blocks	Insertion (%)	Deletion (%)	Substitution (%)	Total error rate (%)
Trivial blocks	0.0128	0.0712	0.0516	0.1356
Complex blocks	0.2932	1.9573	0.1106	2.3611

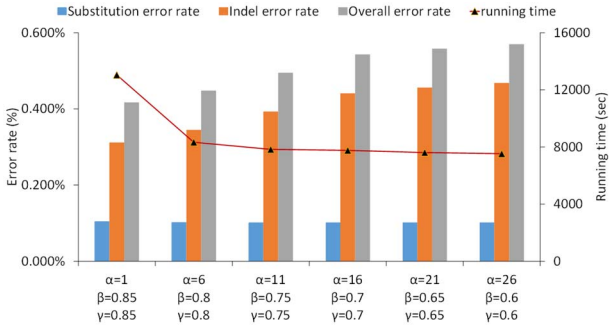

Figure 2. A comparison of runtime and accuracy of polishing for different thresholds in BlockPolish.

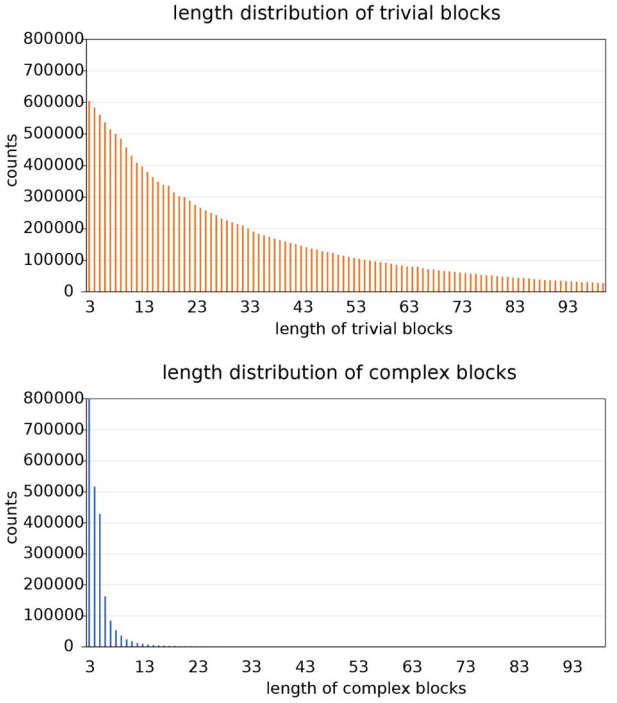

Figure 3. A comparison of length distributions of trivial blocks and complex blocks on the data of HG002 chromosome 6–10.

We calculate the total length of all trivial blocks and that of all complex blocks. The total length shows that 90% of the regions on the contig are divided into trivial blocks, and 10% of the regions are divided into complex blocks. The error rates of trivial blocks and complex blocks are presented in Table 2. The error rate of trivial blocks is 0.1356%, and the error rate of complex blocks is 2.3611%. The higher error rate indicates that polishing complex blocks is more difficult. To optimize the alignment result and improve the polishing accuracy of complex blocks, we apply a fast MSA tool Kalign [31] to realign the reads in each complex block. When performing MSA, we extend the complex block to the left and right by three positions, respectively. As shown in Figure 1C, at position four on the contig, the inserted fragments are only arranged from left to right and not aligned. The MSA effectively realigns the inserted fragments. Then, we calculate the features including the number of different bases, insertions and deletions from the rearranged alignment in complex blocks again. Finally, the features extracted from trivial blocks and complex blocks are fed into two neural networks with different parameters, respectively, to generate consensus sequences.

Neural network architecture

In this section, we will show the architecture of the BlockPolish network. Let $X = \{x_1, x_2, \ldots, x_T\}$ be a sequence of input features (T is the length of the block and each step $x_t \in X$ contains six features: the number of different nucleotide bases {A,C,G,T}, the number of insertions and the number of deletions). As shown in Figure 1D, the network of BlockPolish consists of four Bi-LSTM layers, a compressed projection layer and a flip-flop projection layer. The hidden size of each Bi-LSTM layer is 256, and the output size of last Bi-LSTM layer is 512. The Bi-LSTM layers can take both left and right features when making decisions at any position of the contig. The output from LSTM layers $H = \{h_1, h_2, \ldots h_T\}$ is a tensor with the shape of $T \times 512$. Because of the insertions and deletions in reads-to-assembly alignment, the length of the feature sequence may vary from the length of the label sequence. To overcome the inconsistency, we apply a Connectionist Temporal Classification (CTC) layer [37] in the model architecture. CTC is a sequence prediction method that does not have the limitation that the length of input should be the same as the length of the target.

The compressed projection layer is a linear layer and the input size of the compressed projection layer is 1024. We couple the compressed projection layer with a CTC layer to transform the output H from the previous LSTM layer into a sequence of nucleotide bases without continuous repetition of a single base. In other words, the consecutively repeated nucleotide bases in the sequence are compressed into a single base and the length of the sequence after compression is shorter than before (e.g. the sequence 'CCCTAA' is compressed into 'CTA'). This module only learns the order of different nucleotide bases that appear in the consensus sequence. The output of compressed projection $O_c = \{c_1, c_2, \ldots, c_T\}$ is a tensor with the shape of $T \times 6$. The output symbols of compressed projection contain {A,C,G,T,D,-}. 'D' means a deletion and '-' represents the blank symbol in CTC to separate two adjacent prediction characters. Afterwards, we use a greedy search algorithm to pick the symbol with the largest probability at each time step of the output O_c. By removing the blank symbol '-', deletion 'D' and compressing consecutively repeated symbols, the compressed output sequence is obtained.

Besides the compressed projection layer, there is a flip-flop projection layer in the network. This idea of flip-flop comes from the recognition of consecutively repeated bases in Nanopore flip-flop basecaller (https://github.com/nanoporetech/flappie).

Flip-flop projection is also a linear layer as the compressed projection layer. The input size of the flip-flop projection layer is 1024. The flip-flop projection layer followed by a CTC layer transforms the previous LSTM output H into a sequence in which the continuously repeated nucleotide bases are flip-flopped. In BlockPolish, the flip-flop operation alternately represents the continuously repeated bases using uppercase and lowercase characters, but the length of the sequence is unchanged (e.g. the sequence 'AAAAA' becomes 'AaAaA' after flip-flopped). The output of flip-flop projection layer $O_p = \{p_1, p_2, \ldots, p_T\}$ is a tensor with the shape of $T \times 10$. The output characters of flip-flop projection layer are {A,a,C,c,G,g,T,t,D,-}. To achieve the final output sequence, the greedy search algorithm is used to generate the predicted sequence. The symbols 'D' and '-' in the sequence are removed, and the lowercase symbols are converted to uppercase symbols.

In the neural network, we use a multitask learning framework to optimize the compressed prediction task and the full-length flip-flop prediction task. During training, the two tasks are jointly trained. The losses of these two parts are calculated by two CTC loss functions, and two losses are added as the total loss. During inference, since the output of the compressed prediction does not have the information about the number of repetitions of each base, we cannot recover the consensus sequence from the output sequence of the compressed prediction task. So, the final consensus sequence is the output sequence of full-length flip-flop prediction. Besides, the two tasks make their own decisions and the result of the compressed prediction task can help to validate that of the flip-flop prediction task. We compress the repeated nucleotide bases in the sequence S_p from flip-flop prediction into a new sequence S'_p and then compare the new sequence S'_p with the sequence S_c from compressed prediction. If the sequences S'_p and S_c are equal, the result of the flip-flop prediction task is more likely to be correct.

Model training

The whole network is trained on Nanopore reads of the human sample HG002 chromosomes 6–10 (90% for training and 10% for evaluation). We assemble the raw reads with Wtdbg2 and then polish the draft assembly with one round of Racon. Afterwards, both the Nanopore raw reads and the truth assembly from GIAB are aligned to the Racon polished assembly, respectively. According to the reads-to-assembly alignment, the consensus table is constructed, and we divide each contig into trivial blocks and complex blocks. The feature tensors from the alignment result of trivial blocks including the number of different bases, insertions and deletions can be extracted. Based on the reference-to-assembly alignment, the truth sequence of each trivial block is also obtained. For complex blocks, the read fragments are realigned by an MSA. Then, we extract the feature tensors including the number of different bases, insertions and deletions from the rearranged alignment result of complex blocks. The truth sequence of each complex block is also provided by the reference-to-assembly alignment.

So far, we have the feature tensors and truth sequences for training two models for trivial blocks and complex blocks, respectively. BlockPolish is a two-task model, and the two tasks have the same input feature tensor but different label sequences. To achieve the label of the compressed prediction task, we compress the repeated bases in the truth sequence into a single base. To obtain the label of the flip-flop prediction task, we alternately represent the repeated bases in the truth sequence using uppercase and lowercase characters. Afterward, we train the two prediction tasks using the same feature tensor and the different label sequences. The parameters of the neural network are optimized by Adam optimizer [38]. The initial learning rate is set to 0.001 and the decay ratio of the learning rate is 0.7. To prevent over-fitting, we use the dropout [39] to randomly set the values of some neural cells to zero. The dropout ratio is set to 0.3. To prevent gradient explosion during training, we apply a gradient clipping strategy and set the maximum gradient to 200. The whole network is implemented in Python with PyTorch 1.4.

Results

Evaluation of BlockPolish on polishing Wtdbg2 assembly

We evaluated the performances of Racon, Medaka, MarginPolish & HELEN and BlockPolish on ONT data of NA12878. Since BlockPolish and MarginPolish & HELEN were both trained on the same human sample HG002, the test data set was new for these two polishing tools. The raw reads of NA12878 chromosome 1–10 were extracted by Samtools [40], respectively, after mapping all of the reads to the reference GRCh38. Then, the raw reads of each chromosome were assembled using Wtdbg2 and polished using four different polishing methods. For Racon polishing, each assembly was polished by four rounds of Racon. For Medaka polishing, four rounds of Racon followed by one round of Medaka was used to polish the draft assemblies. For MarginPolish & HELEN polishing, each draft assembly was corrected by MarginPolish and HELEN successively. For BlockPolish polishing, we polished the assemblies with a round of Racon followed by a round of BlockPolish. The base-level error rates of the polished assemblies were reported by the Pomoxis toolkit (https://github.com/nanoporetech/pomoxis). The substitution, deletion and insertion error rates are defined as the number of mismatched, deleted and inserted bases divided by the number of bases in the reference sequence. The indels error rate is the sum of insertion error rate and deletion error rate. As can be seen from Figure 4, BlockPolish gets the lowest overall error rate in each chromosome. The order of accuracy in four polishing tools is BlockPolish, MarginPolish & HELEN, Medaka and Racon. From the error types, BlockPolish fixes much more indel errors than other polishing tools in each chromosome. For all assemblies, the errors are dominated by indels, which leads to the overall lowest error rate for BlockPolish. In terms of substitution errors, MarginPolish & HELEN has the best performance followed by BlockPolish. Besides, we calculated the relative improvement rate of BlockPolish compared with other tools, which is computed as $(e_i - e_B)/e_B$, $e_i \in \{e_H, e_M, e_R\}$. Here, e_B, e_H,e_M and e_R are the overall error rates of the assemblies polished by BlockPolish, MarginPolish & HELEN, Medaka and Racon, respectively. From the results of the relative improvement rate, we can see that BlockPolish has relatively improved by 2.67% over MarginPolish & HELEN, 7.65% over Medaka and 15.70% over Racon on average.

Polishing performance generalizes to a new assembler

We trained a BlockPolish model on the training data assembled by Wtdbg2. To evaluate the generalization power of BlockPolish on the assembly generated by a new assembler, we used Flye to assemble the ONT reads of NA12878 chromosome 1–10. The draft assemblies were polished by Racon, Medaka, MarginPolish & HELEN and BlockPolish, and the error rates of polishing results were evaluated by Pomoxis. According to the base-level error report shown in Figure 5, we found that the assembly polished

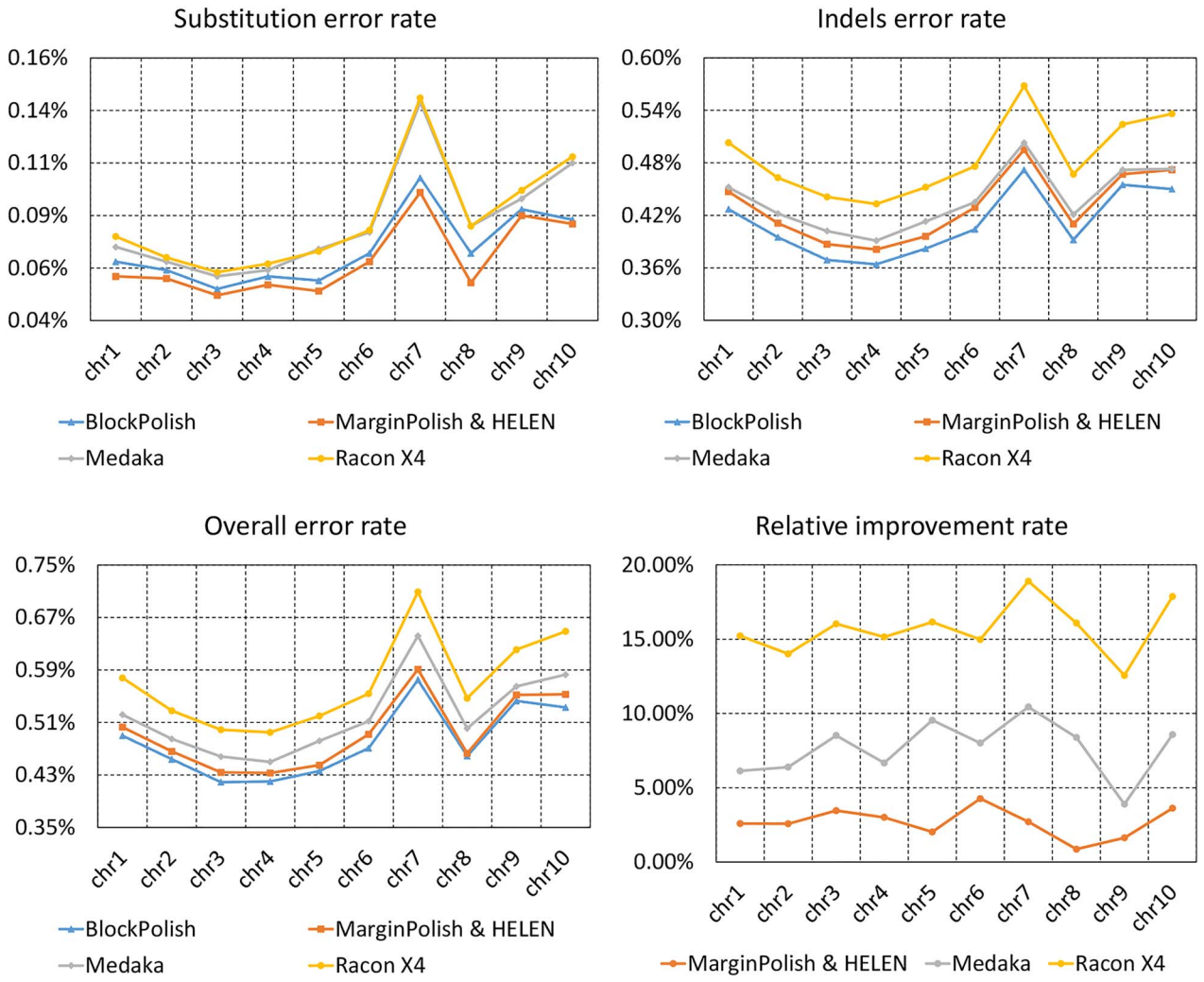

Figure 4. Base-level error rates in polished Wtdbg2 assemblies of NA12878 chromosome 1–10.

by BlockPolish still gets the lowest overall error rate in each chromosome. Compared with MarginPolish & HELEN, BlockPolish has greatly improved in solving indel errors. Compared with Medaka and Racon, BlockPolish has lower error rate both in substitutions and indels. From the results of the relative improvement rate, we can see that BlockPolish has relatively improved by 3.63% over MarginPolish & HELEN, 7.69% over Medaka and 13.86% over Racon on average. In these comparisons, BlockPolish has better performance than state-of-the-art polishing pipelines, even though the draft assemblies were generated by the unseen assembler Flye. Therefore, we do not need to train a specific model for every assembler, which makes BlockPolish more robust and generalizable.

Accuracy, gene completeness and contiguity of polished whole-genome assembly of NA12878

In the previous section, we evaluated the performance of BlockPolish on certain chromosomes of NA12878. In this section, we will assess the performance of BlockPolish on ONT NA12878 whole-genome assembly. First, we assembled the whole-genome sequencing reads of NA12878 with Flye. The draft assembly was polished by different polishing tools separately. The error rates of assemblies reported by Pomoxis are listed in Figure 6. The assembly polished by BlockPolish achieves the lowest error rate, which is mainly due to the reduction of indel errors. MarginPolish & HELEN has the lowest substitution error rate followed by BlockPolish. For the draft assembly, errors are dominated by indel errors. Substitution errors are 6.4 times fewer than indels. In addition to accuracy, gene completeness is an important indicator for evaluation assembly results. To evaluate the gene completeness of polished whole-genome assembly of NA12878, we calculated an asmgene score [41] using minimap2 and paftools. First, the complementary DNA of Homo sapiens from EnsEMBL was aligned to polished assemblies and high-quality truth assembly using minimap2, and the single-copy genes in the truth assembly were selected. Then, we used paftools to compare the alignment result of the single-copy genes in the truth assembly to those in the polished assembly. The asmgene scores of assemblies polished by different technologies are shown in Table 3. Complete asmgene score represents the percentage of single-copy genes in the truth assembly that are mapped as intact single-copy genes in the polished assembly. Duplicated asmgene score gives the percentage of single-copy genes in truth assembly that become multi-copy in the polished assembly. As can be seen from the figure, NA12878 assembly polished by BlockPolish has the highest complete gene score (96.87%) over all the compared

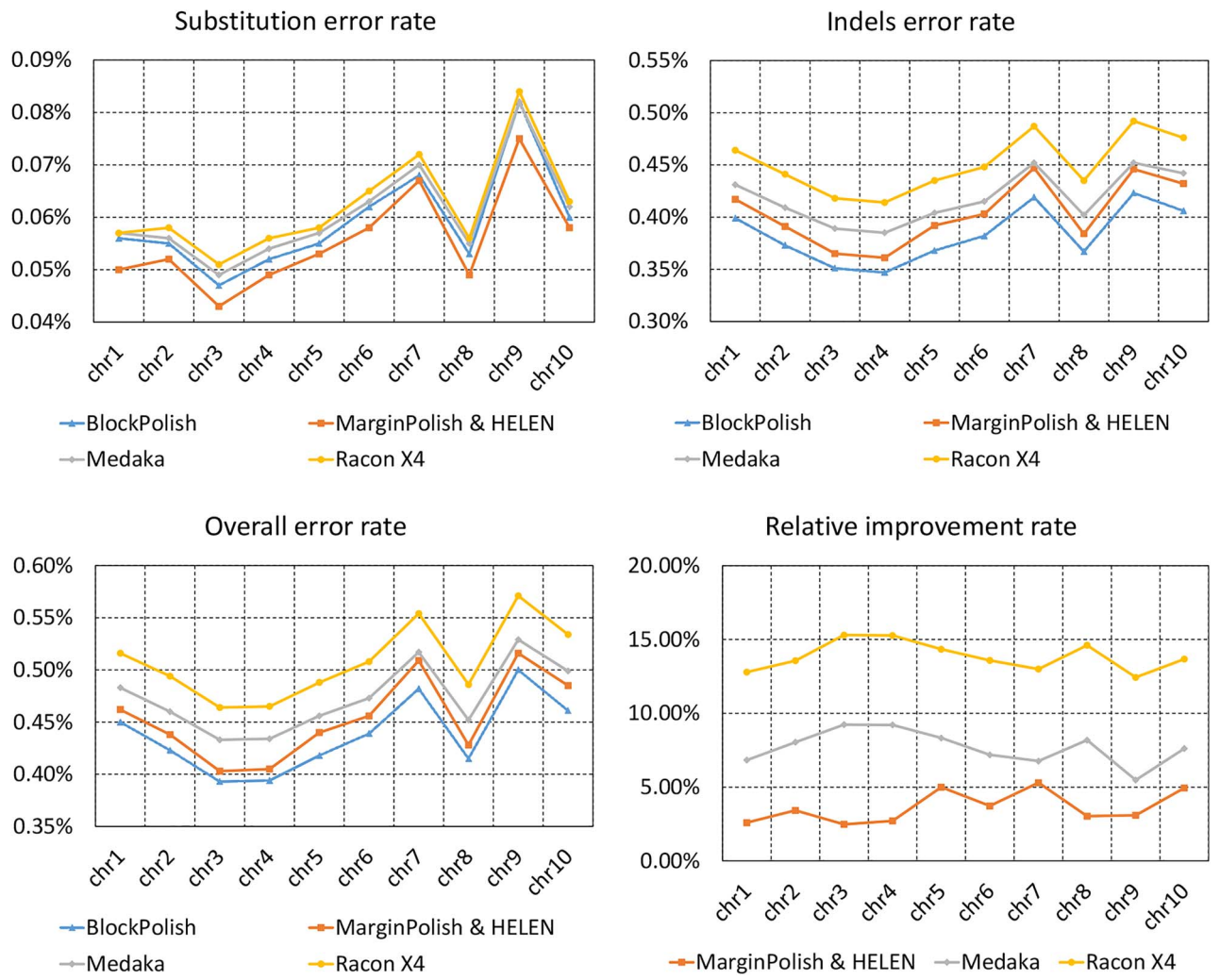

Figure 5. Base-level error rates in polished Flye assemblies of NA12878 chromosome 1–10.

tools. BlockPolish recovered 194 additional complete genes in polished assembly, which is the largest number of recoveries among all the polishing tools. To quantify contiguity, we further assessed the assemblies of NA12878 using Quast [42]. From Table 4, the assembly polished by BlockPolish has the highest N50 of 20.79M. The maximum length of BlockPolish polished assembly is longer than that of assemblies polished by other tools. In terms of genome fraction, Medaka polished assembly has the highest genome fraction, followed by BlockPolish polished assembly.

Evaluation of BlockPolish on assemblies of varying accuracies

At present, there are many base-callers for Nanopore sequencing, and their basecalling accuracies are different. As a result, the error rate of the assembly result is also different. This requires the polishing tools to have the ability to deal with the raw reads and assembly results with different error rates. To evaluate BlockPolish on the assemblies of different error rates, we used multiple base-callers to generate several read sets of NA12878 chromosome 21. The basecalling tools include several versions of Albacore and Guppy. The time span of these tools is from

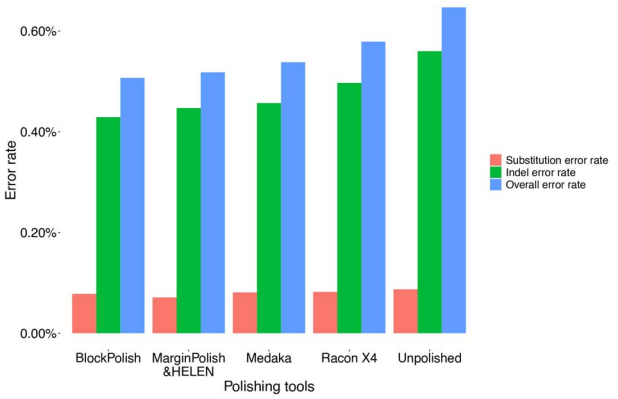

Figure 6. Error rates in the whole-genome assemblies of NA12878 before and after polishing with BlockPolish, MarginPolish & HELEN, Medaka, Racon.

January 2017 to December 2020. Afterwards, we used Flye to assemble these read sets and polish the draft assemblies using BlockPolish. The accuracies of read sets and the error rates of assemblies are shown in Figure 7. The accuracies of reads sets range from 83.78 to 92.52%, with a difference of 8.74%. And the

Table 3. The score of gene completeness in the whole-genome assembly of NA12878 before and after polishing

Polishing tools	Complete gene score	Duplicate gene score	Total gene
BlockPolish	96.87%(34 558)	0.36%(130)	**34 688**
MarginPolish & HELEN	96.64%(34 478)	0.31%(112)	34 590
Medaka	96.71%(34 502)	0.44%(157)	34 659
Racon ×4	96.48%(34 421)	0.41%(147)	34 568
Unpolished	96.32%(34 364)	0.50%(180)	34 544

The bold in the table means the best results.

Table 4. Quast evaluation of the contiguity of polished NA12878 assemblies

Polishing tools	N50 (Mb)	Max Len (Mb)	Total Len (Mb)	Genome fraction (%)
BlockPolish	20.79	72.69	2799	97.105
MarginPolish & HELEN	20.78	72.68	2797	97.035
Medaka	20.63	72.66	2800	97.116
Racon x 4	20.63	72.65	2800	97.104

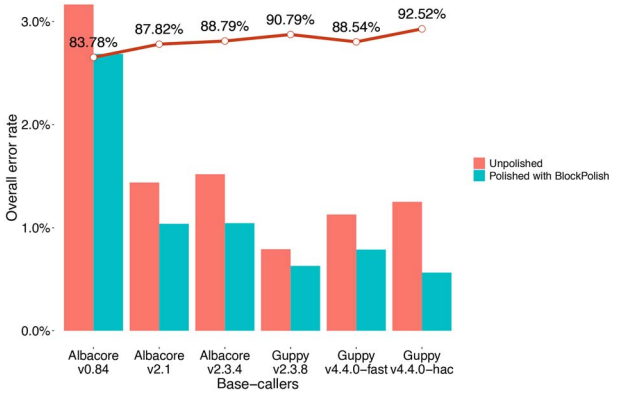

Figure 7. The performance of BlockPolish on polishing assemblies of different qualities. The assemblies are assembled from raw long reads basecalled by different base-callers. The red polyline represents the median identities of reads sets.

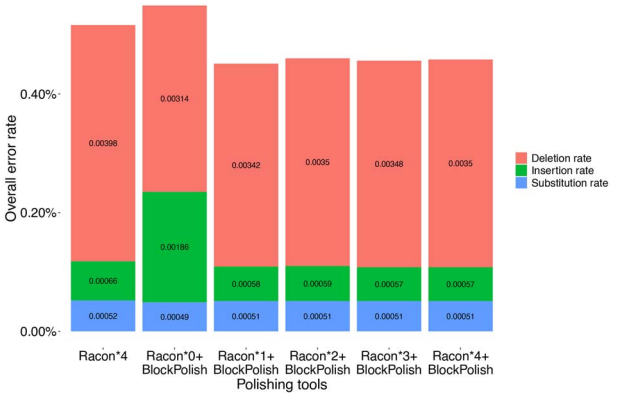

Figure 8. Evaluating the impact of different rounds of Racon in the BlockPolish workflow.

error rates of assembly results range from 0.792 to 3.166%. In all data sets, BlockPolish has improved the accuracy of assembly results, although the error rate of draft assembly varies greatly. Especially, when the raw reads were produced by the latest base-caller Guppy v4.4.0, the assembly polished by BlockPolish had a relative improvement rate of 54.7% over the initial assembly in terms of the error rate.

Evaluation of the role of Racon in the BlockPolish pipeline

Currently, two state-of-the-art Nanopore assembly polishing workflows use the similar framework, a graph-based polishing followed by a neural network-based polishing. The workflow of Medaka polishing consists of four rounds of Racon and one round of Medaka. The workflow of MarginPolish & HELEN polishing is made up of one round of MarginPolish and one round of HELEN. MarginPolish is similar to Racon in function, both using the POA graph [29] to correct the draft assembly initially. HELEN is similar to Medaka in that both of them use deep neural networks to make predictions from the statistical information of reads-to-assembly alignment. In BlockPolish, we also applied the same workflow as the existing polishing tools. Here, we evaluated the polishing performance of different rounds of Racon combined with BlockPolish. The test data set was from ONT data of NA12878 chromosome 1 assembled by Flye. We tested five BlockPolish workflows in which the rounds of Racon range from zero to four. From the base-level error rates result shown in Figure 8, we can see that the error rate of the workflow combined with Racon and BlockPolish is much lower than that of only Racon polishing or only BlockPolish polishing. This is because the graph-based polishing method can help solve long-length structural errors in the assembly, thereby improving the reads-to-assembly alignments. Then, the neural network approach solves fine base-level errors from the more accurate alignment results. Neither of the two types of errors can be ignored, and only solving one of them will not achieve the best polishing performance. For this reason, the state-of-the-art polishing workflows all combine the graph method with the deep learning method. For the workflow of BlockPolish, we executed one round of Racon followed by one round of BlockPolish, because more rounds of Racon brought more computational expense without much performance improvement. It is worth noting that the polishing result generated by two rounds of Racon plus one round of BlockPolish is worse than that generated by one round of Racon plus one round of BlockPolish. The difference between the two experiments is that one performs a round of Racon, and the other performs two rounds. More

Table 5. The runtime of polishing NA12878 chromosome 1 with different polishers

Polishing tools	CPU runtime (s)	GPU runtime (s)	Total runtime (s)
BlockPolish	2829	4920	7749
MarginPolish & HELEN	5040	557	5597
Medaka	2781	6780	9561
Racon ×4	**2781**	**0**	**2781**

The bold in the table means the best results.

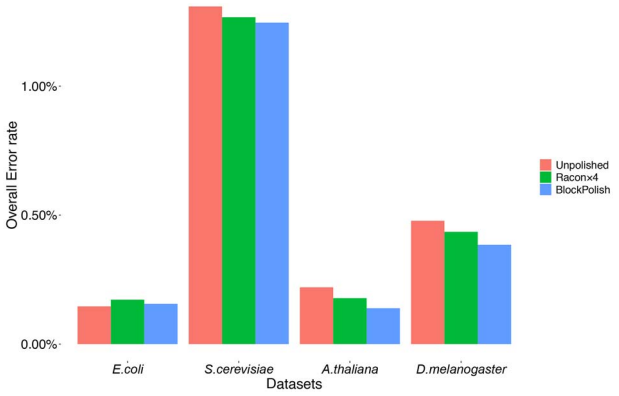

Figure 9. Error rates in PacBio assemblies of *Escherichia coli*, *Saccharomyces cerevisiae*, *Arabidopsis thaliana* and *Drosophila melanogaster* before and after polishing with BlockPolish.

rounds of Racon increase the error rate of polishing results. This is because the iterative running of Racon cannot guarantee that the result of each iteration has a higher accuracy than that of the previous round. The improvement in the accuracy of the iterative polishing is fluctuating.

Evaluation of BlockPolish on PacBio long-read assembly

The distributions of errors in PacBio reads and Nanopore reads are different. PacBio reads show a high insertion error rate, whereas Nanopore reads show similar substitution, insertion and deletion rates [43]. BlockPolish is a polishing tool designed for Nanopore assembly, but it is also applicable for PacBio assembly from the perspective of algorithm. To enable BlockPolish to polish PacBio assemblies, we trained a model of BlockPolish on PacBio data of HG002 (44X). Wtdbg2 was used to assemble the PacBio reads and the truth assembly was generated from GIAB high-confidence variant set against the GRCh38 reference sequence. We tested the performance of BlockPolish on the PacBio data of *Escherichia coli* (84X), *Saccharomyces cerevisiae* (115X), *Arabidopsis thaliana* (155X) and *Drosophila melanogaster* (110X) assembled by Flye. The error rates of polished assemblies are shown in Figure 9. In three of four assembly results, BlockPolish can reduce the errors in the draft assembly. In the assembly of *E. coli*, there are fewer deletion errors. However, BlockPolish mistakenly tries to recover the deletion errors, which causes the lower performance. How to find a better strategy to balance among different types of errors is one future work of our study.

Computational setup and runtime

To evaluate the runtime of different polishing pipelines, we ran all the polishing tools on the Server with 40 Central Processing Units (CPU) (Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz), a Nvidia GeForce RTX 2080Ti Graphics Processing Unit (GPU) and 256GB Memory. BlockPolish consists of three main calculation steps: polishing draft assembly with one round of Racon, generating features of trivial blocks and complex blocks, and predicting the polished sequence by a neural network. The first two steps are executed on CPU, and the last step is executed on GPU. Medaka and MarginPolish & HELEN are similar to BlockPolish where part of the calculation is on CPU and the rest on GPU. Thus, we recorded the CPU runtime and GPU runtime separately, and then added the two to get the final runtime. The runtime of polishing ONT assembly of NA12878 chromosome 1 is shown in Table 5. As can be seen from the table, Racon polishing takes the least time and Medaka polishing takes the longest time. In the polishing technologies based on neural network, MarginPolish & HELEN is the fastest, followed by BlockPolish, and finally Medaka.

Conclusion

In this study, we proposed a novel genome polishing algorithm named BlockPolish. Due to the different accuracy and complexity of regions in the contig, we adopted different strategies for correcting these regions. In our proposed polishing methods, we divided each contig into blocks with low complexity and high complexity according to the consensus table constructed from reads-to-assembly alignment. For complex blocks, we used MSA to realign the raw reads in blocks to optimize the alignment result. Then, two neural network models with different parameters were applied for trivial blocks and complex blocks to predict the consensus sequences, respectively. In terms of network architecture, we used a multitask Bi-LSTM network to predict compressed consensus sequence and full-length flip-flop consensus sequence. The compressed prediction task only learns the order of the different nucleotide bases appear in the consensus sequence. Another task predicting full-length consensus sequence learns not only the order of nucleotide bases but also the number of repetition of each base.

When polishing the ONT data of NA12878, the assembly polished by BlockPolish achieves higher accuracy than Racon, Medaka and MarginPolish & HELEN. In terms of error types, BlockPolish fixes more indels errors, which are dominated in all assemblies. We tested BlockPolish on the assembly produced by a different assembler, which is not seen during training; the assembly polished by BlockPolish still achieves the lowest error rate. It means that we do not need to train a specific model for every assembler, which makes BlockPolish more robust and generalizable. Due to the quality of raw reads, the accuracy of the assembly results may vary greatly. We used several base-callers to generate the raw reads of ONT NA12878 chromosome 20 and produced assemblies with different error rates. Afterwards, we polished the assemblies using BlockPolish, each of the assemblies has an improvement in accuracy. BlockPolish is designed for polishing erroneous ONT assembly, but it is also capable of polishing PacBio assemblies. In four data sets of PacBio assemblies, BlockPolish reduces the errors in three of them.

Key Points

- We present a novel polishing workflow, BlockPolish, to divide the assembly into trivial blocks and complex blocks and apply different polishing strategies for correcting the errors in these two kinds of blocks.
- For complex blocks, we realign the reads in these regions to improve the alignment results with an MSA.

- We train two different Bi-LSTM networks to predict polished sequences for two types of blocks, respectively.
- The results show that in the whole-genome assemblies of NA12878 assembled by Wtdbg2 and Flye, BlockPolish has a higher polishing accuracy than other state-of-the-arts.

Acknowledgments

The authors thank Jain *et al*. [3] for releasing NA12878 Nanopore sequencing data set. The authors thank Shafin *et al*. [9] for releasing Nanopore basecalled data of HG002 and the truth assembly. The authors thank Human Pangenome Coordinating Center for releasing the PacBio data of HG002. The authors thank Xiao *et al*. [36] for releasing the PacBio data of *E. coli*, *S. cerevisiae*, *A. thaliana* and *D. melanogaster*.

Funding

This work was supported in part by the National Natural Science Foundation of China under grants (Nos. U1909208 and 61772557); 111 Project (No. B18059); Hunan Provincial Science and Technology Program (No. 2018wk4001 to J.W.); the US National Institute of Food and Agriculture (NIFA) under grant (2017-70016-26051 to F.L.) and the US National Science Foundation (NSF) under grant (ABI-1759856to F.L.); the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research (OSR) under Award No. (FCC/1/1976-26-01, URF/1/3412-01-01, URF/1/4098-01-01, REI/1/4742-01-01 and REI/1/4473-01-01 to X.G.).

References

1. Pendleton M, Sebra R, Pang AWC, et al. Assembly and diploid architecture of an individual human genome via single-molecule technologies. *Nat Methods* 2015; **12**(8): 780–6.
2. Chen Y, Nie F, Xie SQ, et al. Efficient assembly of nanopore reads via highly accurate and intact error correction. *Nat Commun* 2021; **12**(1): 1–10.
3. Jain M, Koren S, Miga KH, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat Biotechnol* 2018; **36**(4): 338–45.
4. Ni P, Huang N, Zhang Z, et al. Deepsignal: detecting dna methylation state from nanopore sequencing reads using deep-learning. *Bioinformatics* 2019; **35**(22): 4586–95.
5. Teng H, Cao MD, Hall MB, et al. Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. *GigaScience* 2018; **7**(5):giy037.
6. Boža V, Perešíni P, Brejová B, et al. Deepnano-blitz: a fast base caller for minion nanopore sequencers. *Bioinformatics* 2020; **36**(14): 4191–2.
7. Huang N, Nie F, Ni P, et al. Sacall: a neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism. *IEEE/ACM Trans Comput Biol Bioinform* 2020.
8. Cheng H, Concepcion GT, Feng X, et al. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nat Methods* 2021; **18**(2): 170–5.
9. Shafin K, Pesout T, Lorig-Roach R, et al. Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nat Biotechnol* 2020;1–10.
10. Liao X, Li M, Hu K, et al. A sensitive repeat identification framework based on short and long reads. *Nucleic Acids Res* 2021.
11. Firtina C, Bar-Joseph Z, Alkan C, et al. Hercules: a profile hmm-based hybrid error correction algorithm for long reads. *Nucleic Acids Res* 2018; **46**(21): e125–5.
12. Bao E, Lan L. Halc: High throughput algorithm for long read error correction. *BMC bioinformatics* 2017; **18**(1): 1–12.
13. Goodwin S, Gurtowski J, Ethe-Sayers S, et al. Oxford nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome Res* 2015; **25**(11): 1750–6.
14. Madoui MA, Engelen S, Cruaud C, et al. Genome assembly using nanopore-guided long and error-free dna reads. *BMC Genomics* 2015; **16**(1): 327.
15. Hackl T, Hedrich R, Schultz J, et al. proovread: large-scale high-accuracy pacbio correction through iterative short read consensus. *Bioinformatics* 2014; **30**(21): 3004–11.
16. Zimin AV, Marçais G, Puiu D, et al. The masurca genome assembler. *Bioinformatics* 2013; **29**(21): 2669–77.
17. Koren S, Schatz MC, Walenz BP, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat Biotechnol* 2012; **30**(7): 693–700.
18. Bao E, Xie F, Song C, et al. Flas: fast and high-throughput algorithm for pacbio long-read self-correction. *Bioinformatics* 2019; **35**(20): 3953–60.
19. Salmela L, Walve R, Rivals E, et al. Accurate self-correction of errors in long reads using de bruijn graphs. *Bioinformatics* 2017; **33**(6): 799–806.
20. Koren S, Walenz BP, Berlin K, et al. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res* 2017; **27**(5): 722–36.
21. Kolmogorov M, Yuan J, Lin Y, et al. Assembly of long, error-prone reads using repeat graphs. *Nat Biotechnol* 2019; **37**(5): 540–6.
22. Ruan J, Li H. Fast and accurate long-read assembly with wtdbg2. *Nat Methods* 2020; **17**(2): 155–8.
23. Walker BJ, Abeel T, Shea T, et al. Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PloS one* 2014; **9**(11): e112963.
24. Hu J, Fan J, Sun Z, et al. Nextpolish: a fast and efficient genome polishing tool for long read assembly. *Bioinformatics* 2020.
25. Warren RL, Coombe L, Mohamadi H, et al. ntedit: scalable genome sequence polishing. *Bioinformatics* 2019; **35**(21): 4430–2.
26. Zimin AV, Salzberg SL. The genome polishing tool polca makes fast and accurate corrections in genome assemblies. *PLoS Comput Biol* 2020; **16**(6):e1007981.
27. Firtina C, Kim JS, Alser M, et al. Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm. *Bioinformatics* 2020; **36**(12): 3669–79.
28. Vaser R, Sović I, Nagarajan N, et al. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Res* 2017; **27**(5): 737–46.
29. Lee C. Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics* 2003; **19**(8): 999–1008.
30. Huang N, Nie F, Ni P, et al. Neuralpolish: a novel nanopore polishing method based on alignment matrix

construction and orthogonal bi-gru networks. *Bioinformatics* 2021.
31. Lassmann T. *Kalign 3: multiple sequence alignment of large datasets*, 2020.
32. Zook JM, McDaniel J, Olson ND, et al. An open resource for accurately benchmarking small variant and reference calls. *Nat Biotechnol* 2019; **37**(5): 561–6.
33. Chin CS, Peluso P, Sedlazeck FJ, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nat Methods* 2016; **13**(12): 1050–4.
34. Chin CS, Alexander DH, Marks P, et al. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nat Methods* 2013; **10**(6): 563–9.
35. Li H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 2018; **34**(18): 3094–100.
36. Xiao CL, Chen Y, Xie SQ, et al. Mecat: fast mapping, error correction, and de novo assembly for single-molecule sequencing reads. *Nat Methods* 2017; **14**(11): 1072.
37. Graves A, Fernández S, Gomez F, et al. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: *Proceedings of the 23rd international conference on Machine learning*, 2006, 369–76.
38. Kingma DP, Ba J. Adam: A method for stochastic optimization arXiv preprint arXiv:14126980. 2014.
39. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* 2017; **60**(6): 84–90.
40. Li H, Handsaker B, Wysoker A, et al. The sequence alignment/map format and samtools. *Bioinformatics* 2009; **25**(16): 2078–9.
41. Cheng H, Concepcion GT, Feng X, et al. Haplotype-resolved de novo assembly with phased assembly graphs arXiv preprint arXiv:200801237. 2020.
42. Gurevich A, Saveliev V, Vyahhi N, et al. Quast: quality assessment tool for genome assemblies. *Bioinformatics* 2013; **29**(8): 1072–5.
43. Dohm JC, Peters P, Stralis-Pavese N, et al. Benchmarking of long-read correction methods. *NAR Genomics and Bioinformatics* 2020; **2**(2):lqaa037.