Bioinformatics, 32(15), 2016, 2369–2371
doi: 10.1093/bioinformatics/btw146
Advance Access Publication Date: 24 March 2016
Applications Note

OXFORD

Genome analysis

BLESS 2: accurate, memory-efficient and fast error correction method

Yun Heo[1], Anand Ramachandran[1], Wen-Mei Hwu[1], Jian Ma[2] and Deming Chen[1,*]

[1]Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA and [2]Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA

*To whom correspondence should be addressed.

Associate Editor: Inanc Birol

Received on July 25, 2015; revised on March 7, 2016; accepted on March 12, 2016

Abstract

Summary: The most important features of error correction tools for sequencing data are accuracy, memory efficiency and fast runtime. The previous version of BLESS was highly memory-efficient and accurate, but it was too slow to handle reads from large genomes. We have developed a new version of BLESS to improve runtime and accuracy while maintaining a small memory usage. The new version, called BLESS 2, has an error correction algorithm that is more accurate than BLESS, and the algorithm has been parallelized using hybrid MPI and OpenMP programming. BLESS 2 was compared with five top-performing tools, and it was found to be the fastest when it was executed on two computing nodes using MPI, with each node containing twelve cores. Also, BLESS 2 showed at least 11% higher gain while retaining the memory efficiency of the previous version for large genomes.
Availability and implementation: Freely available at https://sourceforge.net/projects/bless-ec
Contact: dchen@illinois.edu
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Correcting errors in sequencing reads is a time-consuming and memory-intensive process. The occurrences of patterns (*k*-mers in many tools) in reads should be counted, and patterns with a small number of occurrences have to be replaced with ones having a large number of occurrences. Storing patterns requires a lot of memory, and searching for alternative patterns takes a long time for large genomes. Therefore, memory efficiency and fast runtime are as important as accuracy in error correction methods.

To provide a memory-efficient error correction method, BLESS, which uses a Bloom filter as the main data structure, was developed (Heo *et al.*, 2014). While BLESS could generate accurate results with a much smaller amount of memory than previous tools, it was too slow to be applied to reads from large genomes.

Recently, some new error correction methods that can correct errors in a large dataset in a short period of time have been developed (Li, 2015; Song *et al.*, 2014). However, to the best of our knowledge, none of the present tools satisfy all the three constraints (i.e. memory efficiency, runtime and accuracy).

To address the three requirements, we have developed a new version of BLESS. In BLESS 2, the accuracy of the error correction algorithm has been further improved over that of BLESS by adding new algorithmic steps. BLESS 2 corrects errors even in solid *k*-mers, *k*-mers that exist multiple times in reads, using the quality score distribution of input reads. Solid *k*-mers were originally treated as error-free *k*-mers. There is also a new algorithm introduced in BLESS 2 for trimming reads where errors cannot be corrected or corrections are ambiguous. In addition to quality improvements, the overall execution has been parallelized using hybrid MPI and OpenMP programming, which make BLESS 2 the fastest tool when executed on multiple computing nodes. All these improvements were made without hurting the memory efficiency of the predecessor.

2 Methods

BLESS 2 is parallelized using hybrid MPI and OpenMP programming. Therefore, the process can not only be parallelized on a server

with multiple CPU cores and shared memory, but it can also be accelerated further by running it on multiple servers.

The overall BLESS 2 architecture is shown in Figure 1. The grey boxes in the figure represent computing nodes in a cluster. First, Node 1 builds a quality score histogram that can be used for the error correction step. Then, all nodes start to fetch input reads to count the occurrences of *k*-mers. In order to accelerate the *k*-mer counting step, we applied MPI to KMC (Deorowicz *et al.*, 2015), which is one of the fastest and the most memory-efficient *k*-mer counters, and the modified KMC was integrated into BLESS 2. In the original KMC, *k*-mers are sent to one of 512 bins, and *k*-mers in each bin are counted separately. In BLESS 2, all of the *N* nodes invoke the modified KMC, and each node counts *k*-mers in 512/*N* bins.

After KMC is finished, Node 1 collects the outputs of *N* nodes and constructs a *k*-mer occurrence histogram. This histogram is used to determine the threshold for solid *k*-mers. Each node separates *k*-mers in its private bins that have occurrences larger than the threshold, and programs them into a local Bloom filter.

Each node's Bloom filter thus contains solid *k*-mers private to the corresponding node. Bloom filter data in each node is broadcast to all the other nodes, and each node reduces all the received data, along with its own Bloom filter data into a single Bloom filter that represents all the solid *k*-mers in the entire input read set using a bitwise OR operation. Each node then corrects *R*/*N* reads where *R* is the total number of input reads, using the local copy of the Bloom filter. Corrected reads from all nodes are concurrently written, using MPI, to a final output file.

The accuracy of BLESS 2 has been significantly improved over that of its predecessor. There could be many solid *k*-mers that have errors, and they were not corrected in BLESS, because all such *k*-mers were considered error-free. The new algorithm checks whether a solid *k*-mer needs to be considered an erroneous *k*-mer when it has a base with a quality score below the fifth percentile. In addition, BLESS 2 can remove errors even when it cannot find right solutions for correcting them. In such a case, the new algorithm checks the locations of weak *k*-mers in a read and tries to determine the minimum number of bases that need to be trimmed to remove as many weak *k*-mers as possible.

3 Results and discussion

In order to evaluate the performance of BLESS 2, we compared it with five state-of-the-art error correction tools. All the experiments were done on servers, each with two six-core Xeon X5650 CPUs and 24 GB of memory.

ERR194147, which is a read set from NA12878, was used as an evaluation input. D1 is reads from chr1-3 and D2 is the entire read set. D3 is reads from SRR065390 *C.elegans*, the same dataset that was used in the BFC paper (Li, 2015). How the inputs were prepared and how all the error correction tools were run, and how results were evaluated can be found in the supplementary document.

The input reads were corrected using BLESS, BLESS 2, BFC-KMC (Li, 2015), Lighter (Song *et al.*, 2014), Musket (Liu *et al.*, 2013), QuorUM (Marçais *et al.*, 2013), SGA (Simpson and Durbin, 2011). The results are summarized in Table 1. We prepared D1 because Musket, QuorUM and SGA could not handle D2 with 24 GB of memory on our server. BLESS could not finish correcting errors in D2 in 70 h, which is the maximum allocated run time in our cluster. For D1, BLESS 2 generated the most accurate result; its gain was higher than those of the others by at least 11 and 20% on the average.

Accuracies of BLESS 2, BFC-KMC, and Lighter for D2 were similar to their accuracies for D1, which means the D1 results can represent the accuracy of an error correction tool for the entire human genome. For D3, BLESS 2 again showed the best accuracy

Table 1. Error correction results

Data	Software	Gain	Memory (GB)	Runtime (min)
D1	BLESS 2 (1 node)	0.565	3.7	65
	BLESS 2 (2 nodes)	0.565	3.7	41
	BLESS 2 (3 nodes)	0.565	3.7	33
	BLESS 2 (4 nodes)	0.565	3.7	30
	BLESS	0.510	1.1	2,893
	BFC-KMC	0.505	11.2	60
	Lighter	0.439	3.2	54
	Musket	0.454	13.2	133
	QuorUM	0.477	10.5	120
	SGA	0.448	12.0	396
D2	BLESS 2 (1 node)	0.563	5.6	320
	BLESS 2 (2 nodes)	0.563	5.6	203
	BLESS 2 (3 nodes)	0.563	5.6	160
	BLESS 2 (4 nodes)	0.563	5.6	139
	BFC-KMC	0.496	20.5	274
	Lighter	0.434	13.9	231
D3	BLESS 2 (1 node)	0.833	3.7	7
	BLESS	0.801	0.1	331
	BFC-KMC	0.800	9.8	9
	Lighter	0.776	1.4	4
	Musket	0.751	2.9	24
	QuorUM	0.821	4.1	12
	SGA	0.751	1.7	55

TP, erroneous bases that are correctly modified; FP, all bases that are incorrectly modified; FN, erroneous bases that are not modified; Gain: (TP - FP)/(TP + FN); Twelve threads were used in a node for all the tools.

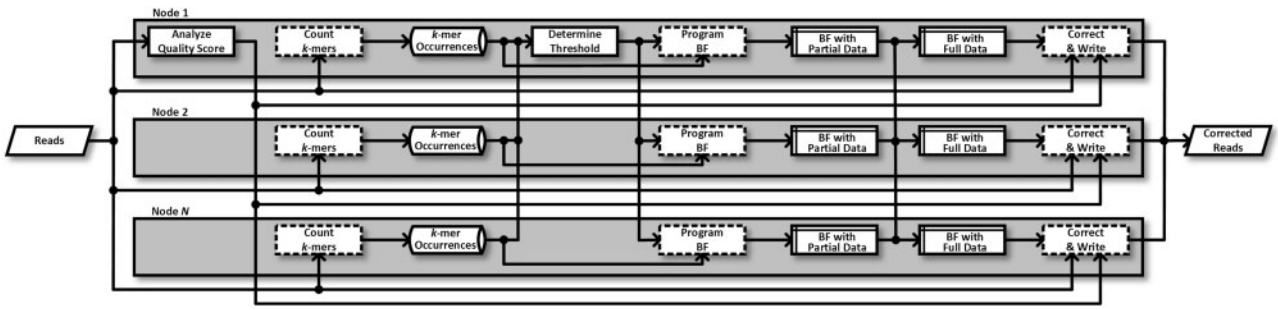

Fig. 1. Overview of the BLESS 2 structure. Rectangles with dotted lines are processes that are parallelized using OpenMP

among the compared tools. The percentage of trimmed bases rose from 0.9% for D1 and D2 to 1.6% for D3. Since trimming increases the chances that a read aligns to multiple locations, the aligned locations of BLESS 2's output reads were compared with those of the uncorrected reads. The result showed that 99.8% of D3 reads were aligned the same location after error correction. This is discussed in more detail in Section 5.1 of supplementary document.

The effect of error correction on DNA assembly was also assessed. D3 and its error correction results were assembled using Gossamer (Conway *et al.*, 2012) and the assembly results were compared using Quast (Gurevich *et al.*, 2013). All the error correction tools helped improve the NG50 length, and the improvement was the most when using BLESS 2. Details are presented in Section 5.2. of supplementary document.

While BLESS 2 consumed the smallest amount of memory for D2, Lighter used the least memory for D1. This is because KMC that is invoked in BLESS 2 requires a constant 4 GB of memory irrespective of genome size. For D2, the size of the Bloom filter in BLESS 2 was larger than 4 GB and KMC was no longer a memory bottleneck.

The runtime of BLESS 2 on one computing node was comparable to that of the other methods, and BLESS 2 became the fastest tool when more nodes were available. When four nodes were used, BLESS 2 became 2.3 times faster than when one node was used. The current version of BLESS 2 reads input read files three times (i.e. for analyzing quality scores, counting *k*-mers using KMC and correcting errors), and reading compressed input read files consumes a significant amount of time. Since there is no efficient way to read a compressed file in parallel, this part cannot be accelerated even when the number of available nodes increases. In the next version, KMC source code will be embedded in BLESS 2 and we could analyze quality scores while counting *k*-mers using KMC, giving further speedup.

Conflict of Interest: none declared.

References

Conway,T. *et al.* (2012) Gossamer – a resource-efficient de novo assembler. *Bioinformatics*, **28**, 1937–1938.

Deorowicz,S. *et al.* (2015) KMC 2: Fast and resource-frugal k-mer counting. *Bioinformatics*, **31**, 1569–1576.

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

Heo,Y. *et al.* (2014) BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, **30**, 1354–1362.

Li,H. (2015) BFC: correcting Illumina sequencing errors. *Bioinformatics*, **31**, 2885–2887.

Liu,Y. *et al.* (2013) Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, **29**, 308–315.

Marçais,G. *et al.* (2013) *QuorUM: An error corrector for Illumina reads*. arXiv preprint arXiv:*1307.3515*.

Simpson,J. and Durbin,R. (2011) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, **22**, gr.126953.126111-126556.

Song,L. *et al.* (2014) Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biol.*, **15**, 509.