Bioinformatics, 31(17), 2015, 2885–2887
doi: 10.1093/bioinformatics/btv290
Advance Access Publication Date: 6 May 2015
Applications Note

Sequence analysis

BFC: correcting Illumina sequencing errors

Heng Li

Medical Population Genetics Program, Broad Institute, Cambridge, MA 02142, USA

Associate Editor: Inanc Birol

Received on February 12, 2015; revised on April 17, 2015; accepted on May 2, 2015

Abstract

Summary: BFC is a free, fast and easy-to-use sequencing error corrector designed for Illumina short reads. It uses a non-greedy algorithm but still maintains a speed comparable to implementations based on greedy methods. In evaluations on real data, BFC appears to correct more errors with fewer overcorrections in comparison to existing tools. It particularly does well in suppressing systematic sequencing errors, which helps to improve the base accuracy of *de novo* assemblies.
Availability and implementation: https://github.com/lh3/bfc
Contact: hengli@broadinstitute.org
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Error correction is a process to fix sequencing errors on a sequence read by using other overlapping reads that do not contain the errors. Many *de novo* assemblers, in particular short-read assemblers for large genomes, use error correction to reduce the complexity of the assembly graph such that the graph can be fitted to limited RAM. Error correction was first expressed as the *spectral alignment problem* (Pevzner *et al.*, 2001), whereby we take a set of trusted k-mers and attempt to find a sequence with minimal corrections such that each k-mer on the corrected sequence is trusted. The majority of error correctors are based on this idea and take a greedy approach to solving this problem approximately. They make a correction based on the local sequence context and do not revert the decision. We worried that the greedy strategy might affect the accuracy given reads from a repeat-rich diploid genome, so derived a new non-greedy algorithm that explores larger search space in attempt to achieve higher accuracy.

2 Methods

Algorithm 1 is the key component of BFC. It defines a *state* of correction as a 4-tuple $(i, W, \mathcal{C}, p)$, which consists of the position i of the preceding base, the last (k-1)–mer W ending at i, the set $\mathcal{C}$ of previous corrected positions and bases (called a *solution*) up to i, and the penalty p of solution $\mathcal{C}$. BFC keeps all possible states in a priority queue $\mathcal{Q}$. At each iteration, it retrieves the state $(i, W, \mathcal{C}, p)$ with the lowest penalty p (line 1) and adds a new state $(i+1, W[1, k-2] \circ a, \mathcal{C}', p')$ if a is the read base or $W \circ a$ is a trusted k-mer. If the first k-mer in S is error free and we disallow untrusted k-mers by removing line 3, this algorithm finds the optimal solution to the spectral alignment problem. Chaisson *et al.* (2004) have described a more general non-greedy algorithm. Its strict form is not implementable in practice. The heuristic adaptation is loosely similar to ours without line 3.

Algorithm 1: Error correction for one string in one direction

Input: K-mer size k, set $\mathcal{H}$ of trusted k-mers, and one string S
Output: Set of corrected positions and bases changed to
Function CorrectErrors$(k, \mathcal{H}, S)$ **begin**
  $\mathcal{Q} \leftarrow$HeapInit() ▷ $\mathcal{Q}$ *is a priority queue*
  HeapPush$(\mathcal{Q}, (k-2, S[0, k-2], \emptyset, 0))$ ▷ 0*-based strings*
  while $\mathcal{Q}$ *is not empty* **do**
1 $(i, W, \mathcal{C}, p) \leftarrow$HeapPopBest$(\mathcal{Q})$ ▷ *current best state*
    $i \leftarrow i + 1$
    if $i = |S|$ **then return** $\mathcal{C}$ ▷ *reaching the end of S*
2 $\mathcal{N} \leftarrow \{(i, \mathrm{A}), (i, \mathrm{C}), (i, \mathrm{G}), (i, \mathrm{T})\}$ ▷ *set of next bases*
    foreach $(j, a) \in \mathcal{N}$ **do** ▷ try all possible next bases
      $W' \leftarrow W \circ a$ ▷"$\circ$" *concatenates strings*
      if $i = j$ ***and*** $a = S[j]$ **then** ▷ *no correction*
        if $W' \in \mathcal{H}$ **then** ▷good read base; no penalty
          HeapPush$(\mathcal{Q}, (j, W'[1, k-1], \mathcal{C}, p))$
        else ▷ *bad read base; penalize*
3 HeapPush$(\mathcal{Q}, (j, W'[1, k-1], \mathcal{C}, p+1))$
      else if $W' \in \mathcal{H}$ **then** ▷ make a correction with penalty
4 HeapPush$(\mathcal{Q}, (j, W'[1, k-1], \mathcal{C} \cup \{(j, a)\}, p+1))$

Table 1. Performance of error correction

Prog.	*k*	Time	RAM	Perfect	Chim.	Better	Worse
raw data	–	–	–	2.40 M	12.4 k	–	–
BFC-bf	31	7h32m	23.3 G	3.01 M	13.1 k	783 k	9.2 k
BFC-bf	55	**4h41m**	23.3 G	**3.05 M**	11.8 k	819 k	11.4 k
BFC-ht	31	7h15m	83.5 G	3.03 M	13.6 k	816 k	10.8 k
BFC-ht	55	5h51m	67.9 G	**3.05 M**	11.7 k	830 k	**9.0 k**
BLESS	31	6h31m	22.3 G	2.91 M	13.1 k	674 k	20.8 k
BLESS	55	5h09m	22.3 G	3.01 M	**11.5 k**	775 k	10.3 k
Bloocoo	31	5h52m	**4.0 G**	2.88 M	14.1 k	764 k	31.5 k
Fermi2	29	17h14m	64.7 G	3.00 M	17.7 k	**849 k**	42.8 k
Lighter	31	5h12m	13.4 G	2.98 M	13.0 k	756 k	30.1 k
Musket	27	21h33m	77.5 G	2.94 M	22.5 k	790 k	36.3 k
SGA	55	48h40m	35.6 G	3.01 M	12.1 k	755 k	12.8 k

In total, 445 million pairs of ~150 bp reads were downloaded from BaseSpace, under the sample 'NA12878-L7' of project 'HiSeq X Ten: TruSeq Nano (4 replicates of NA12878)', and were corrected together. On a subset of 2 million randomly sampled read pairs, the original and the corrected sequences were mapped to hs37d5 (http://bit.ly/GRCh37d5) with BWA-MEM (Li, 2013). A read is said to become *better* (or *worse*) if the best alignment of the corrected sequence has more (or fewer) identical bases to the reference genome than the best alignment of the original sequence. The table gives *k*-mer size (maximal size used for Bloocoo, fermi2, Lighter and Musket), the wall-clock *time* when 16 threads are specified if possible, the peak *RAM* measured by GNU time, number of corrected reads mapped *perfectly*, number of *chimeric* reads (i.e. reads with parts mapped to different places), number of corrected reads becoming *better* and the number of reads becoming *worse* than the original reads. For each metric, the best tool is highlighted in boldface.

It is possible to modify Algorithm 1 to correct insertion and deletion errors (INDELs) by augmenting the set of the 'next bases' at line 2 to:

$$\mathcal{N} = \{(j,a)|j \in \{i-1,i\}, a \in \{\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\}\} \cup \{(i,\ \epsilon\ )\}$$

In this set, $(i,\ a)$ substitutes a base at position i, $(i,\ \epsilon\ )$ deletes the base and $(i-1, a)$ inserts a base a before i. We have not implemented this INDEL-aware algorithm because such errors are rare in Illumina data.

The worst-case time complexity of Algorithm 1 is exponential in the length of the read. In implementation, we use a heuristic to reduce the search space by skipping line 4 if the base quality is 20 or higher (Q20) and the *k*-mer ending at it is trusted, or if five bases or two Q20 bases have been corrected in the last 10 bp window. If function CorrectErrors still takes too many iterations before returning, it stops the search and claims the read uncorrectable.

Given a read, BFC finds the longest substring on which each *k*-mer is trusted. It then extends the substring to both ends of the read with Algorithm 1. If a read does not contain any trusted *k*-mers, BFC exhaustively enumerates all *k*-mers one-mismatch away from the first *k*-mer on the read to find a trusted *k*-mer. It marks the read uncorrectable if none or multiple trusted *k*-mers are found this way.

We provided two related implementations of Algorithm 1, BFC-bf and BFC-ht. BFC-bf uses KMC2 (Deorowicz *et al.*, 2015) to get exact *k*-mers counts and then keeps *k*-mers occurring three times or more in a blocked bloom filter (Putze *et al.*, 2007). BFC-ht uses a combination of bloom filter and in-memory hash table to derive approximate *k*-mer counts (Melsted and Pritchard, 2011) and counts of *k*-mers consisting of Q20 bases. We modified Algorithm 1 such that missing trusted high-quality *k*-mers incurs an extra penalty. This supposedly helps to correct systematic sequencing errors which are recurrent but have lower base quality.

Table 2. Effect on *de novo* assembly

Program	Scaffold NG50 (kb)	Contig aligned-N50 (kb)	Alignment break points	Potential FP SNP
raw data	31.6/29.9	14.8/20.2/4.9	352/157/29	1568/334
BFC-bf	33.7/33.7	17.3/22.8/7.6	341/173/33	3334/668
BFC-ht	34.8/34.2	17.3/22.9/9.1	314/176/31	1397/374
BLESS	33.6/31.7	15.4/20.7/8.0	351/165/28	1744/414
Bloocoo	34.5/33.7	17.1/22.6/6.2	340/177/32	3128/480
Fermi2	33.8/33.7	16.8/22.6/8.9	333/174/32	1444/396
Lighter	34.6/33.6	16.6/22.4/7.9	329/180/34	3011/651
Musket	33.4/32.2	16.0/21.2/8.2	338/181/30	1940/617
SGA	34.4/33.4	16.7/22.4/6.6	360/163/32	3107/495

In total, 33.8 million pairs of 100 bp *C. elegans* reads were downloaded from SRA under accession SRR065390, corrected using *k*-mer length 23 and then assembled with Velvet-1.2.10 (Zerbino and Birney, 2008; velvetg option '-exp_cov auto -cov_cutoff auto -ins_length 250' with *k*-mer length 61), ABySS-1.5.2 [Simpson *et al.*, 2009; abyss-pe option 'k = 67 q = 0 s = 500 n = 5' taken from Simpson and Durbin (2012)] and fermikit-0.9 (fermi2.pl option '-Es100m'). For Velvet and ABySS, contigs were derived by splitting scaffolds at contiguous 'N' bases longer than 5 bp. In the table, each row gives the NG50 of Velvet/ABySS scaffolds longer than 200 bp, the aligned N50 of Velvet/ABySS/fermikit contigs longer than 200 bp, the number of contig alignment break points with > 200 bp flanking sequences and the number of false positive unfiltered/filtered fermikit SNPs as compared to freebayes-0.9.20 calls (Garrison and Marth, 2012; option '–experimental-gls'). BFC and fermi2 were tuned to work with fermikit for human variant calling before this evaluation while others were not.

3 Results and discussions

We evaluated BFC along with BLESS-v0p23 (Heo *et al.*, 2014), Bloocoo-1.0.4 (Drezen *et al.*, 2014), fermi2-r175 (Li, 2012), Lighter-20150123 (Song *et al.*, 2014), Musket-1.1 (Liu *et al.*, 2013) and SGA-0.9.13 (Simpson and Durbin, 2012). We ran the tools on a Linux server with 20 cores of Intel E5-2660 CPUs and 128 GB RAM. Precompiled binaries are available through http://bit.ly/biobin and the command lines were shown in Supplementary Table S2. Notably, BLESS only works with uncompressed files. Other tools were provided with gzip'd FASTQ.

On human data (Table 1), BFC has similar performance to other error correctors that use greedy algorithms. It tends to correct more errors without introducing new errors, potentially due to its non-greedy and quality-aware algorithm.

BFC also works well with assemblers (Table 2). It should be noted that the short reads were sequenced from the Bristol N2 strain, the same as the *Caenorhabditis elegans* reference genome. We expect to see few alignment break points and base-level differences. Extended discussions on the results can be found in Supplementary Notes.

Funding

NHGRI U54HG003037; NIH GM100233

Conflict of Interest: none declared.

References

Chaisson,M. *et al.* (2004) Fragment assembly with short reads. *Bioinformatics*, **20**, 2067–2074.

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

Drezen,E. *et al*. (2014) GATB: Genome assembly & analysis tool box. *Bioinformatics*, **30**, 2959–2961.

Garrison,E. and Marth,G. (2012) Haplotype-based variant detection from short-read sequencing. *arXiv:1207.3907*.

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

Li,H. (2012) Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, **28**, 1838–1844.

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv:1303.3997*.

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

Melsted,P. and Pritchard,J.K. (2011) Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics*, **12**, 333.

Pevzner,P.A. *et al*. (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.*, **98**, 9748–953.

Putze,F. *et al*. (2007) Cache-, hash- and space-efficient bloom filters. In: *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6–8, 2007, Proceedings*, pp. 108–121.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, **22**, 549–56.

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

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

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