BIOINFORMATICS APPLICATIONS NOTE Vol. 30 no. 23 2014, pages 3402–3404
doi:10.1093/bioinformatics/btu558

Sequence analysis

Advance Access publication August 20, 2014

BioBloom tools: fast, accurate and memory-efficient host species sequence screening using bloom filters

Justin Chu*, Sara Sadeghi, Anthony Raymond, Shaun D. Jackman, Ka Ming Nip, Richard Mar, Hamid Mohamadi, Yaron S. Butterfield, A. Gordon Robertson and Inanç Birol*

Canada's Michael Smith Genome Sciences Centre, British Columbia Cancer Agency, Vancouver, BC V5Z 4S6, Canada

Associate Editor: Alfonso Valencia

ABSTRACT

Large datasets can be screened for sequences from a specific organism, quickly and with low memory requirements, by a data structure that supports time- and memory-efficient set membership queries. Bloom filters offer such queries but require that false positives be controlled. We present BioBloom Tools, a Bloom filter-based sequence-screening tool that is faster than BWA, Bowtie 2 (popular alignment algorithms) and FACS (a membership query algorithm). It delivers accuracies comparable with these tools, controls false positives and has low memory requirements.

Availability and implementaion: www.bcgsc.ca/platform/bioinfo/software/biobloomtools

Contact: cjustin@bcgsc.ca or ibirol@bcgsc.ca

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

Received on April 30, 2014; revised on June 26, 2014; accepted on August 13, 2014

1 INTRODUCTION

Pipelines that detect pathogens and contamination screen for host sequences so they do not interfere with downstream analysis (Castellarin *et al.*, 2012; Kostic *et al.*, 2011; Tang *et al.*, 2013; Xu *et al.*, 2014). The alignment-based algorithms that these pipelines use provide mapping locations that are irrelevant for classification, and thus perform more computation than is needed. To address this, we have developed BioBloom Tools (BBT).

BBT uses Bloom filters—probabilistic, constant time access data structures that identify whether elements belong to a set (Bloom, 1970). Bloom filters are similar to hash tables but do not store the elements themselves; instead, they store a fixed number of bits for every element into a common bit array. Thus, they use less memory, but queries to the filter may return false membership (hits) because of hash collisions in the common bit array. The false-positive rate (FPR) resulting from these false hits can be managed by increasing the size of the filter (Supplementary Material). Using Bloom filters for sequence categorization was pioneered by the program FACS (Stranneheim *et al.*, 2010). Here, we describe a Bloom filter implementation that includes heuristics to control false positives and increase speed.

*To whom correspondence should be addressed.

2 METHODS

We first build filters from a set of reference sequences by dividing the sequences into all possible k-mers (substrings of length k). We compare the forward and reverse complement of every k-mer, and include the alphanumerically smaller sequence in the filter. We calculate the bit signature of a k-mer by mapping the sequence to a set of integer values using a fixed number of hash functions (Supplementary Materials) (Broder and Mitzenmacher, 2004). The bitwise union of the signatures of all the k-mers constitutes a Bloom filter for the corresponding reference sequences.

To test whether a query sequence of length l is present in the target reference(s), we use a sliding window of k-mers. Starting at one end of the query sequence, and shifting one base pair at a time along this sequence, we check each k-mer against each reference's Bloom filter. When a k-mer matches a filter, we incrementally calculate a score:

$$s = \sum_{i=1}^{c} \sum_{j=1}^{a_i} \frac{1 - \frac{1}{(j+1)}}{l-k}$$

where c is the number of contiguous stretches of adjacent filter-matching k-mers until the current position in the query, and a_i is the length of the i-th stretch. This heuristic penalizes likely false-positive hits. We evaluate k-mers this way until we reach either a specified score threshold (s*) or the end of the query sequence. If at any point we reach s*, we categorize the query as belonging to the reference, and terminate the process for that query. Further, we use a jumping k-mer heuristic that skips k k-mers when a miss is detected after a long series of adjacent hits. This efficiently handles cases in which the query has a single (or a few) base mismatch(es) with the target.

3 BENCHMARKING

We compared BBT against two widely used Burrows–Wheeler transform-based alignment tools that have low memory usage and high accuracy—BWA (Li and Durbin, 2003) and Bowtie 2 (BT2; Langmead and Salzberg, 2012)—and against the C++ implementation of FACS (https://github.com/SciLifeLab/facs). Tool versions and other details are provided in the Supplementary Materials.

3.1 Benchmarking on simulated data

We used dwgsim (https://github.com/nh13/DWGSIM) to generate simulated Illumina reads from human, mouse and *Escherichia coli* reference genomes. For each genome, we generated 1 million 2×150 bp paired-end (PE) reads and 1 million 100 bp single-end (SE) reads. We used *E.coli* because it is a common contaminant and is genetically distant from human.

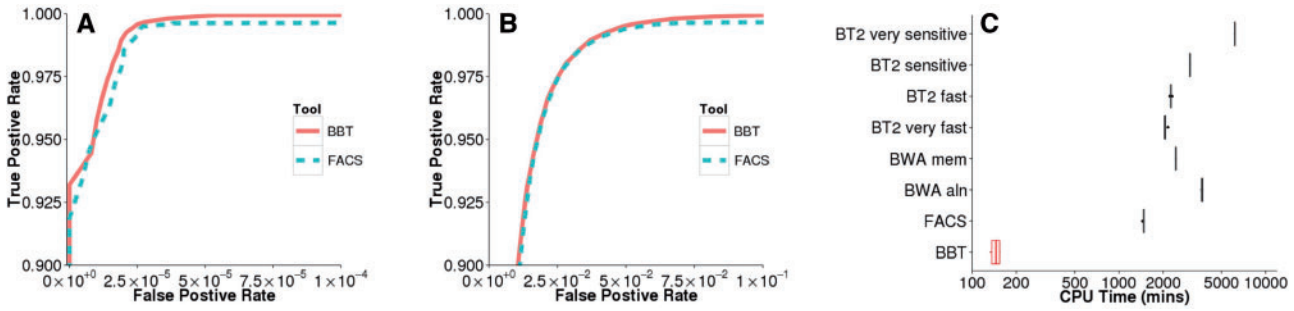

Fig. 1. Performance comparisons of BBT against FACS, BWA and BT2. Receiver operator characteristic curves of BBT and FACS using simulated 100 bp SE reads from *Homo sapiens* mixed with (**A**) *E.coli* and (**B**) *Mus musculus* filtered against an *H.sapiens* Bloom filter using a *k*-mer size of 25 bp; (**C**) CPU time benchmark comparing BT2 (for a range of built-in settings), BWA (using aln and mem settings), FACS and BBT, on one lane of human 2×150 bp PE Illumina HiSeq 2500 reads

With mouse, which is commonly used in xenograft studies, we tested categorization accuracy for species that are closely related genetically.

Because FACS does not support PE reads, we used the 100 bp SE reads to compare the false- and true-positive rates (FPR and TPR, respectively) of BBT and FACS. We tested a range of scoring thresholds for both tools. Using a *k*-mer size of 25 bp, BBT generally matched or outperformed FACS (Fig. 1A and B). We note that, for shorter *k*-mers, performance of BBT and FACS algorithms would deteriorate, especially in distinguishing sequences from closely related references. For both tools, longer *k*-mers gave lower FPR but also lower maximum TPR (Supplementary Figs S1 and S2), with BBT performing increasingly better than FACS for longer *k*-mers.

To compare BBT and FACS to BWA and BT2, we used 2×150 bp PE reads. In our tests, overall, BBT performed comparably with the aligners and outperformed 'fast' and 'very fast' settings of BT2 in both false-negative rate (FNR) and false-discovery rate (FDR; Table 1).

3.2 Benchmarking on experimental data

We used a single lane of 2×150 bp PE human DNA reads (https://basespace.illumina.com/run/716717/2x150-HiSeq-2500-demo-NA12878) generated with an Illumina HiSeq 2500 sequencer to benchmark computational performance. For a controlled comparison, we ran at least eight replicates for each tool, and we measured CPU time, with all applications using a single thread.

We ran BBT with $s^* = 0.1$ and compared it with FACS, BWA and BT2, using a range of run modes for the latter two tools. BBT was faster than the fastest aligner/settings combination (BT2 very fast) by at least an order of magnitude (Fig. 1C). The mapping rates (categorization rates for BBT and FACS) of each tool were comparable, at 96.69 (BT2 very sensitive), 96.57 (BT2 sensitive), 96.18 (BT2 fast), 95.97 (BT2 very fast), 99.76 (BWA mem), 95.12 (BWA aln), 95.81 (FACS) and 97.27% (BBT).

3.3 Memory usage

For categorization, using the human reference and simulated reads, the peak memory usage (GB) for each tool was 3.8 (BBT), 4.8 (FACS), 3.1 (BWA aln), 5.2 (BWA mem) and 3.4 (BT2). These figures are for categorization only and do not include the memory usage for creating the FM-indexes or Bloom filters. Unless slower disk-based methods are used, creating an FM-index takes at least $O(n\log(n))$ bits of memory, where n is the size of the reference sequence (Ferragina *et al.*, 2012). In contrast, Bloom filter memory usage is the same for the creation and categorization stages, and takes $O(-n\log(f))$ bits of memory, where f is the FPR and n is the number of input sequences.

We created filters using 3.2 GB of memory for both FACS and BBT. Assuming optimal numbers of hash functions are used, filters with the same size should have similar FPRs. However, in practice, we had to use different FPR settings in creating these filters (FPR of 0.5% for FACS and 0.75% for BBT). We note that the tools would differ from theoretical estimates because of implementation-specific calculation differences.

Finally, to demonstrate the scalability of BBT, we built a filter for 5182 bacterial sequences (representing 6×10^{10} unique 25-mers), using 6.8 GB of memory, corresponding to an FPR of 0.75%.

Table 1. Benchmarking results using simulated paired end 2×150 bp reads

Tool and Settings	FNR (*H.sapiens*)	FDR (*M.musculus*)	FDR (*E.coli*)
BT2 very sensitive	1.40×10^{-5}	2.03×10^{-2}	0
BT2 sensitive	7.52×10^{-4}	9.08×10^{-3}	0
BT2 fast	1.26×10^{-2}	5.90×10^{-3}	0
BT2 very fast	1.34×10^{-2}	5.65×10^{-3}	0
BWA aln	3.26×10^{-3}	8.14×10^{-4}	0
BWA mem	0	1.92×10^{-1}	1.00×10^{-4}
FACS	1.22×10^{-1}	9.88×10^{-3}	0
BBT ($s^* = 0.1$)	8.42×10^{-3}	3.78×10^{-3}	0

Note: All reads were treated as SE reads for FACS.

ACKNOWLEDGEMENTS

The authors thank Genome Canada, British Columbia Cancer Foundation, and Genome British Columbia for their support, and Irene Li, Rene Warren and Karen Mungall for useful discussions.

Funding: The work was funded by Genome Canada, British Columbia Cancer Foundation and Genome British Columbia.

Conflict of interest: none declared.

REFERENCES

Bloom,B.H. (1970) Space/time trade-offs in hash coding with allowable errors. *Commun ACM*, **13**, 422–426.

Broder,A. and Mitzenmacher,M. (2004) Network applications of bloom filters: a survey. *Int. Math.*, **1**, 485–509.

Castellarin,M. *et al.* (2012) Fusobacterium nucleatum infection is prevalent in human colorectal carcinoma. *Genome Res.*, **22**, 299–306.

Ferragina,P. *et al.* (2012) Lightweight data indexing and compression in external memory. *Algorithmica*, **63**, 707–730.

Kostic,A.D. *et al.* (2011) PathSeq: software to identify or discover microbes by deep sequencing of human tissue. *Nat. Biotechnol.*, **29**, 393–396.

Li,H. and Durbin,R. (2003) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Stranneheim,H. *et al.* (2010) Classification of DNA sequences using bloom filters. *Bioinformatics*, **26**, 1595–1600.

Tang,K.-W. *et al.* (2013) The landscape of viral expression and host gene fusion and adaptation in human cancer. *Nat. Commun.*, **4**, 2513.

Xu,G. *et al.* (2014) RNA CoMPASS: a dual approach for pathogen and host transcriptome analysis of RNA-Seq datasets. *PLoS One*, **9**, e89445.