BIOINFORMATICS ORIGINAL PAPER Vol. 28 no. 10 2012, pages 1324–1327
doi:10.1093/bioinformatics/bts123

Sequence analysis Advance Access publication March 13, 2012

Fulcrum: condensing redundant reads from high-throughput sequencing studies

Matthew S. Burriesci, Erik M. Lehnert* and John R. Pringle
Department of Genetics, Stanford University School of Medicine, Stanford, CA 94305-5120, USA
Associate Editor: Alex Bateman

ABSTRACT
Motivation: Ultra-high-throughput sequencing produces duplicate and near-duplicate reads, which can consume computational resources in downstream applications. A tool that collapses such reads should reduce storage and assembly complications and costs.
Results: We developed Fulcrum to collapse identical and near-identical Illumina and 454 reads (such as those from PCR clones) into single error-corrected sequences; it can process paired-end as well as single-end reads. Fulcrum is customizable and can be deployed on a single machine, a local network or a commercially available MapReduce cluster, and it has been optimized to maximize ease-of-use, cross-platform compatibility and future scalability. Sequence datasets have been collapsed by up to 71%, and the reduced number and improved quality of the resulting sequences allow assemblers to produce longer contigs while using less memory.
Availability and implementation: Source code and a tutorial are available at http://pringlelab.stanford.edu/protocols.html under a BSD-like license. Fulcrum was written and tested in Python 2.6, and the single-machine and local-network modes depend on a modified version of the Parallel Python library (provided).
Contact: erik.m.lehnert@gmail.com
Supplementary information: Supplementary information is available at *Bioinformatics* online.

Received on November 20, 2011; revised on February 23, 2012; accepted on March 7, 2012

1 INTRODUCTION

Ultra-high-throughput-sequencing (UHTS) methods are being used both for resequencing projects and for *de novo* sequencing and assembly of both genomes and transcriptomes (Flicek and Birney, 2009; Li *et al.*, 2010; Mondal *et al.*, 2011; Schatz *et al.*, 2010). During resequencing, reads can generally be aligned to specific locations in the previously sequenced genome. In contrast, *de novo* assembly often requires assemblers to take the input of many gigabases of sequence and return a much smaller result, ranging from tens of megabases for transcriptome assemblies to a few gigabases for many eukaryotic genome assemblies. Some assemblers (*e.g*. Velvet) store read information in RAM during processing, limiting the amount of raw sequence that can be processed (Zerbino and Birney, 2008).

UHTS often produces reads that are exact duplicates of each other due to enzyme biases, PCR amplification, or (in transcriptomes) the presence of highly expressed sequences. In addition, genetic polymorphisms, PCR errors and sequencing errors can result in near-duplicate reads, precluding the use of naive hashing algorithms to efficiently find and collapse all duplicates. Collapsing identical and near-identical reads into single consensus reads with high quality scores can significantly reduce the physical memory and/or processing time required for assembly. Previously described methods for finding identical and near-identical reads require large amounts of physical memory, long processing times or both (Giardine *et al.*, 2005; Qu *et al.*, 2009). Indeed, if the number of reads (N) is large (as is typically the case), performing a full $N \times N$ comparison on a single computer is effectively precluded by the amount of processing time required. In addition, these methods were designed to process single-end reads and cannot process the increasingly common paired-end reads.

In order to increase the number of reads that can be collapsed by the popular algorithm FASTX-collapse (Giardine *et al.*, 2005), one could first preprocess the data with an error-correction program like SHREC (Schröder *et al.*, 2009), hybrid-SHREC (Salmela, 2010) or Quake (Kelley *et al.*, 2010). However, SHREC and hybrid-SHREC used several GB of RAM just to process relatively small microbial datasets, whereas Quake required very large memories and extended times to process human datasets. This need for computational resources is expected, because these programs are designed to solve the difficult problem of error correction in non-duplicated reads. However, substantially fewer resources are required if error correction is only performed on near-duplicate reads, and even the collapse of near-duplicates can provide a significant performance advantage for subsequent assembly.

In the past decade, $N \times N$ comparisons performed over multiple networked computers have been used to process very large datasets, such as in page-ranking websites. MapReduce has been helpful in solving these problems because of its power, low cost and ease of use. In addition, investigators without extensive computational resources of their own can rent time from one of several commercial entities. Because of its wide utility, we included support for MapReduce in Fulcrum, a pipeline for collapsing identical and near-identical reads that can be run on a single machine, a local network or a rented network of arbitrary size, depending on the demands of the project.

2 ALGORITHM AND IMPLEMENTATION

Fulcrum is a read collapser that identifies and groups identical and near-identical reads and returns a single consensus sequence. It is written in Python (v. 2.6) to maximize cross-platform compatibility and allow users to easily change the comparator functions within its framework in order to accept other types of data or to call specialized

*To whom correspondence should be addressed.

modules written in C/C++. Fulcrum aims to simplify the problem of comparing N reads in a dataset to every other read in the set. In order to do this, it takes advantage of two characteristics of UHTS datasets, the generally higher quality of base-calls near the beginnings of reads and the constrained space defined by sets of sequences of length k (5^k combinations). These two characteristics allow for the definition of a hash function (the identity of the first k bases of a read) that greatly increases the likelihood that reads hashed into the same group by this function are nearly identical.

Thus, Fulcrum bins reads into files (buckets) of user-defined maximum size by their initial k-mer of user-defined length; k-mer identity is required at this step. The file system of the host server is used to store the binned data awaiting processing, reducing the requirement for physical memory or the network load when using multiple machines. The comparisons of the sequences within each file can then be processed as separate jobs.

Fulcrum incorporates a basic quality-aware collapsing 'inner-loop' algorithm that has been used for all of the trials described in this report. After the reducer function receives data from a mapper or reads data from the k-mer-sorted files on the hard drive, each sequence is placed into a list with all other sequences that share its initial k-mer, creating the 'source list'. A list of groups of strongly similar reads, called the 'similar list', is initialized using the first element of the source list. Each element of the source list is then compared to each consensus sequence of the developing similar list. If it differs by few enough high-quality bases (threshold base quality and number defined by the user via two command-line options), the new sequence is added to the group; if not, it initializes a new group. The quality threshold allows a sequence with many low-quality bases to be merged with a high-quality consensus, whereas the number threshold prevents the collapse of highly repetitive sequences that differ by several bases whose quality is above the user-set threshold. Whenever a sequence is added to a group, a new consensus sequence is calculated, with voting on discrepancies weighted by quality score. When the last sequence from the source list is checked, the reducer returns the consensus sequences and the identities of the constituent reads for each.

Fulcrum can run in any of three modes, all of which use the same comparison function, making it easy to switch to whichever mode is appropriate for the amount of data at hand. The single-machine and local-network modes utilize the Parallel Python module to distribute jobs to multiple processors, either on the same machine or on multiple machines in a network. This module has been extended so that it loads a file just-in-time, reducing the memory requirement of the master node. The MapReduce mode uses HiveQL and Hadoop, which can run atop Amazon EC2 (Elastic MapReduce Developer Guide API Version 2009-03-31) or other MapReduce clusters. In the mapping phase, mappers tag each read with its starting k-mer and transmit the data to reducers so that all sequences in the same bucket reside in the memory of a single reducer. In the reducing phase, each sequence is compared to the other sequences within its bucket, and consensus sequences are returned. Both the mapping and reducing phases can be run on multiple machines in parallel (as readily rented from commercial sources), so this technique will be able to handle datasets that are even larger than those currently produced by UHTS methods and be usable even by investigators who do not have access to high-performance clusters at their own institutions.

Importantly, Fulcrum incorporates a paired-end mode that allows collapsing of only those pairs of sequences that contain strong similarities on both ends. In addition, the mapping stage passes paired-end sequences to the reducers as a single construct, avoiding problems that might arise from partial collapsing of one end and not the other.

A detailed tutorial with a link to example data is available at http://pringlelab.stanford.edu/protocols.html. It takes a user from raw FASTQ files to collapsed data using each of the three modes and includes detailed instructions on accessing commercial MapReduce networks. It also describes common errors and illustrates trade-offs between speed and sensitivity. The flexibility of the Fulcrum framework should allow other researchers to benefit from its workload distribution features to extend the size and speed of current processing operations, for example by writing more specialized collapsers for 454, Illumina or data from other sequencing platforms.

3 RESULTS AND DISCUSSION

Table 1 shows the relative reductions in read numbers and the run-times achieved using Fulcrum in single-machine mode with different parameter settings on a publicly available Illumina dataset of *Pseudomonas* sequences. As expected, either increasing the size of the k-mer used for initial binning or decreasing the maximum bucket size decreased run-time; both approaches lead to fewer sequences' being compared with each other with the inner-loop algorithm. Although the latter approach can lead to some identical sequences' not being collapsed with each other (if they have been partitioned into separate buckets), neither approach had a significant effect on the overall level of collapse seen with this dataset. Thus, either of these approaches, or the use of increased computational resources (more machines in the network and/or longer run-times), should allow Fulcrum to be used effectively with datasets of much larger size.

We also experimented with different maximum bucket sizes during our trials of Fulcrum in its MapReduce mode. Limitations were often necessary because the memory size of the reducing instances available at an affordable price was too small to handle the number of sequences in the largest bins. It was clear that run times could decrease significantly as the maximum allowed bucket size was decreased, but more detailed analyses were restricted to the single-machine trials (see above and Table 1).

Table 1 also shows that as expected, increasing the tolerance for single-base mismatches increased the level of collapse. With this dataset, allowing one, two or three mismatches changed the level of collapse by only a few per cent, but the optimal number of mismatches to allow is likely to depend on the particular dataset being analyzed.

The analysis in Table 1 was also informative in regard to the reduction in memory requirements provided by Fulcrum. Although in this dataset nearly 30% of the reads could be merged into the remaining set, and the number of sequences starting with the most common k-mer was several hundred times overrepresented, the maximum memory used while concurrently processing the six largest bins was ~12 GB. This was reduced to ~2.4 GB when the large bins were subdivided into smaller buckets. This compares favorably with the ~4 GB that were necessary for hybrid-SHREC to process a much smaller number of reads (~1.1 million) in 43 min

Table 1. Variation of run-time and percent sequence reduction with the parameters used for Fulcrum[a]

k-mer size[b]	Maximum bucket size (MB)[c]	Errors allowed[d]	Quality score[e]	Reduction (%)	Run-time (h)
8	20	1	3	27.1	7.5
8	20	2	3	29.0	8.5
8	20	3	3	29.9	9.7
14	20	1	3	26.7	2.2
14	20	2	3	28.4	2.1
14	20	3	3	29.2	2.3
20	20	1	3	26.3	2.1
20	20	2	3	27.8	2.1
20	20	3	3	28.5	2.2
20	20	1	7	27.5	2.0
20	20	2	7	28.4	2.1
20	20	3	7	28.9	2.1
20	1000	1	7	27.7	7.7
20	1000	2	7	28.6	6.7

[a] 61 806 537 paired-end 76-bp Illumina sequence reads from six lanes of sequence (Hiatt *et al.*, 2010) (http://www.ncbi.nlm.nih.gov/sra/SRX015074) were used for these trials. Data were processed on a workstation with a six-core Phenom 2 processor and 16 GB of DDR3 RAM (cost ~$1000).
[b] Reads were binned if they were identical over their initial *k*-mer of sequence of the indicated size.
[c] Binned reads were collected in buckets (files) of the indicated maximum size.
[d] Number of errors allowed during the comparison of the sequences within each bucket.
[e] Maximum quality score each compared base could have before it was counted against maximum allowed errors.

Table 2. Performance of Fulcrum during assembly of Illumina human-transcriptome data[a]

Assembly statistic[b]	Assembly of trimmed raw reads[c]	Assembly of trimmed, collapsed reads[c,d]
Percent collapse	N.A.[e]	21
Number of contigs	63 931	63 009
N50	1380	1391
Maximum contig length	15 055	15 025
Mean contig length	679	687
velveth runtime	18 min	13 min
Maximum *velveth* memory usage	8.5 GB	8.0 GB
velvetg runtime	31 min	19 min
Maximum *velvetg* memory usage	7.5 GB	4.9 GB

[a] 27 531 230 paired-end 76-bp Illumina sequence reads from a single lane (lane 5) (see http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeCaltechRnaSeq/wgEncodeCaltechRnaSeqGm12878R2x75Il200FastqRd1Rep1.fastq.tgz and http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeCaltechRnaSeq/wgEncodeCaltechRnaSeqGm12878R2x75Il200FastqRd2Rep1.fastq.tgz) were used for these trials.
[b] Reads were assembled using Velvet/Oases (Velvet version 1.1.04 and Oases version 0.1.21) while monitoring memory and CPU usage by recording the output of *top* every 60 s.
[c] Reads were filtered for quality and length. Briefly, reads were trimmed such that no nucleotide had a quality score <10, and no ambiguous nucleotides (Ns) remained in the read. Reads shorter than 45 bp were discarded.
[d] Read pairs were collapsed allowing one mismatch of quality score 7 or higher per read pair, and the metadata provided by Fulcrum were discarded prior to assembly.
[e] N.A., not applicable.

(Salmela, 2010), as well as with the very large memories (a 20-processor Hadoop instance and 16-processor Open MP system) that were required for Quake to process (over multiple days) the 82.9 gigabases resulting from a human-genome-sequencing project (Kelley *et al.*, 2010). This performance increase arises because Fulcrum is designed to facilitate subsequent assembly by reducing both input read number and some of the complexity resulting from PCR and sequencing errors, rather than addressing the more difficult problem of full error correction.

We also tested Fulcrum with a human transcriptome sequence dataset from the ENCODE Project (Table 2). The raw and Fulcrum-collapsed data were trimmed and then assembled using Velvet/Oases. The assembly time was reduced by 35% when we used the collapsed data, an effect that would be multiplied if a multiple-*k*-mer assembly technique were used. We also found that maximum memory usage of the Velvet processes *velveth* and *velvetg* fell when we used collapsed reads (Table 2).

The decrease in memory usage and improvement in assembly statistics were somewhat modest with the ENCODE dataset, presumably because of its relatively small size and the high quality of the reads. As expected, the advantages provided by Fulcrum are more pronounced in working with sequence datasets from libraries that were highly amplified and/or have lower average read quality. An example is provided by the project that originally stimulated the writing of Fulcrum, namely the *de novo* assembly of a transcriptome for the sea anemone *Aiptasia pallida* (Sunagawa *et al.*, 2009) using paired end 76- and 101-bp Illumina sequence reads (E. Lehnert *et al.*, submitted for publication). Initial attempts to assemble the entire dataset using Velvet/Oases on a 64-GB instance failed due to insufficient memory; the failure occurred during the running of *velvetg*. After collapsing reads with Fulcrum, assembly proceeded smoothly. The seemingly large reduction in memory requirement is understandable in that Fulcrum produced read-number collapses ranging from 28.8% to 71.7% for the datasets from the individual libraries that were sequenced. In addition, average nucleotide quality scores were significantly improved, particularly at the ends of reads, by the collapse of independent sequence reads of the same PCR clones into a consensus sequence, leading to longer reads post-trimming and significantly improved assembly.

Fulcrum was designed to speed *de novo* sequencing and assembly efforts in which an $N \times N$ comparison of reads is necessary, and we anticipate that it will be most useful in such projects. However, Fulcrum can also be used as a first step in read mapping for polymorphism detection. A rapid estimate of how many duplicate reads are present can be obtained from the first step of Fulcrum on a single machine, which clusters sequences by their initial *k*-mer and writes them into files. Opening the directory and allowing the operating system to sort the files by size will give an idea of the extent of read duplication. As a rule of thumb, if the 10 largest files comprise >10 times their expected value, as given by the original FASTQ file size divided by 5^k (where 5 is the number of possible nucleotides, including N, assuming that all are represented in equal frequency), the use of Fulcrum is likely to be helpful.

In sequencing/assembly projects, problems might arise in repetitive-element or paralogous-gene assembly if reads with many mismatches are allowed to collapse; thus, we recommend limiting the number of bases whose quality is above the user-set threshold that are allowed to differ to a maximum of three. Users may want to reduce this number further if recent gene or genome duplication is believed to have occurred. In addition, a potential pitfall of using Fulcrum-collapsed sequences for read-mapping applications after an assembled transcriptome or genome is available is that the collapse of sequences with mismatches could compromise the identification of low-frequency polymorphisms in a deeply sequenced, genetically heterogeneous population. Thus, the standard practice of using re-mapped original genome sequences is likely to be preferable for this purpose.

In summary, for appropriate applications, the possibility of running Fulcrum in single-machine, local-network or MapReduce modes, all using the same comparison function, should make it a flexible and valuable tool for dealing with datasets of varying size, including very large datasets.

ACKNOWLEDGEMENTS

We thank Dario Valenzano and Michelle Davison for providing additional Illumina and 454 data that were used to evaluate applicability to real-life datasets, Joe Foley for directing us to other useful test sets, and Prof. Jure Leskovec for introducing MB to several parallel-processing algorithms and methodologies. We also thank the reviewers of an earlier version for comments that helped to improve this article.

Funding: Gordon and Betty Moore Foundation (#2629); National Institutes of Health (5 T32 HG000044); National Science Foundation (Graduate Research Fellowship).

Conflict of Interest: Amazon.com donated computer time to a class taken by MB; he used time worth ∼$70.

REFERENCES

Flicek,P. and Birney,E. (2009) Sense from sequence reads: methods for alignment and assembly. *Nat. Methods*, **6**, S6–S12.

Giardine,B. *et al.* (2005) Galaxy: a platform for interactive large-scale genome analysis. *Genome Res.,* **15**, 1451–1455.

Hiatt,J.B. *et al.* (2010) Parallel, tag-directed assembly of locally derived short sequence reads. *Nat. Methods*, **7**, 119–122.

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

Li,R. *et al.* (2010) The sequence and de novo assembly of the giant panda genome. *Nature*, **463**, 311–317.

Mondal,K. *et al.* (2011) Targeted sequencing of the human X chromosome exome. *Genomics*, **98**, 260–265.

Qu,W. *et al.* (2009) Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Res.*, **19**, 1309–1315.

Salmela,L. (2010) Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, **26**, 1284–1290.

Schatz,M.C. *et al.* (2010) Assembly of large genomes using second-generation sequencing. *Genome Res.*, **20**, 1165–1173.

Schröder,J. *et al.* (2009) SHREC: a short-read error correction method. *Bioinformatics*, **25**, 2157–2163.

Sunagawa,S. *et al.* (2009) Generation and analysis of transcriptomic resources for a model system on the rise: the sea anemone *Aiptasia pallida* and its dinoflagellate endosymbiont. *BMC Genomics*, **10**, 258.

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