ROCK: digital normalization of whole genome sequencing data

Véronique Legrand[1], **Thomas Kergrohen**[2,3], **Nicolas Joly***[1], **and Alexis Criscuolo**[4,5]¶

1 Institut Pasteur, Université Paris Cité, Plateforme HPC, F-75015 Paris, France **2** Prédicteurs moléculaires et nouvelles cibles en oncologie, INSERM, Gustave Roussy, Université Paris-Saclay, Villejuif, France **3** Département de Cancérologie de l'Enfant et de l'Adolescent, Gustave Roussy, Université Paris-Saclay, Villejuif, France **4** Institut Pasteur, Université Paris Cité, Bioinformatics and Biostatistics Hub, F-75015 Paris, France **5** Institut Pasteur, Université Paris Cité, Plateforme de Microbiologie Mutualisée (P2M), F-75015 Paris, France ¶ Corresponding author

DOI: 10.21105/joss.03790

Software

- Review
- Repository
- Archive

Editor: Luiz Irber

Reviewers:

- @hiraksarkar
- @ctb

Submitted: 01 September 2021
Published: 07 May 2022

Summary

Due to advances in high-throughput sequencing technologies, generating whole genome sequencing (WGS) data with high coverage depth (e.g. $\geq 500\times$) is now becoming common, especially when dealing with non-eukaryotic genomes. Such high coverage WGS data often fulfills the expectation that most nucleotide positions of the genome are sequenced a sufficient number of times without error. However, performing bioinformatic analyses (e.g. sequencing error correction, whole genome *de novo* assembly) on such highly redundant data requires substantial running times and memory footprint.

To reduce redundancy within a WGS dataset, randomly downsampling high-throughput sequencing reads (HTSR) is trivial. Nevertheless, this first-in-mind strategy is not efficient as it does not minimize variation in sequencing depth, thereby eroding the coverage depth of genome regions that are under-covered (if any). To cope with this problem, a simple greedy algorithm, named digital normalization, was designed to efficiently downsample HTSRs over genome regions that are over-covered (Brown et al., 2012). Given an upper-bound threshold $\kappa > 1$, it returns a subset S_κ such that the coverage depth induced by the HTSRs in S_κ is expected to be at most $\varepsilon\kappa$ across genome (where $\varepsilon > 1$ is a constant). By discarding highly redundant HTSRs while retaining sufficient and homogeneous coverage depth ($\approx \varepsilon\kappa$), this algorithm strongly decreases both running times and memory required to subsequently analyze WGS data, with often little impact on the expected results (Crusoe et al., 2015).

Interestingly, the digital normalization algorithm can be easily enhanced in several ways, so that the final subset contains fewer but more qualitative HTSRs. Unfortunately, these different improvements are scattered in distinct program tools. `ROCK` (*Reducing Over-Covering K-mers*) was therefore developed with the key purpose of implementing a fast, accurate and easy-to-use digital normalization procedure. The C++ source code is available under GNU Affero General Public License v3.0 at https://gitlab.pasteur.fr/vlegrand/ROCK.

Statement of need

The digital normalization algorithm is based on a count-min sketch (CMS), a probabilistic data stream structure able to store the number of occurrences of the canonical *k*-mers (i.e. short oligonucleotides of fixed length k, invariant to reverse-complement) that derive from the selected HTSRs in S_κ (Cormode & Muthukrishnan, 2005; Zhang et al., 2014). By means

*deceased

of this CMS, the *k*-mer coverage $c_k(r)$ of any new HTSR r (i.e. the *k*-mer coverage depth of the corresponding genome region induced by the selected HTSRs already in S_κ) can be approximated by the median number of occurences of the *k*-mers derived from r. The algorithm iterates over all HTSRs r to progressively fill S_κ: when $c_k(r) \leq \kappa$, r is added into S_κ and the CMS is updated with every *k*-mers of r. At the end, the coverage depth induced by the selected HTRs in S_κ is therefore expected to be upper-bounded by $\varepsilon\kappa$, where $\varepsilon = \ell/(\ell - k + 1)$ and ℓ is the average HTSR length. This initial version of the algorithm can be run using either the Python program `normalize-by-median` from the package *khmer* (Crusoe et al., 2015), the Java program `BBnorm` from the package *BBTools* (Bushnell, 2014) or the C++ program `BigNorm` (Wedemeyer et al., 2017).

Of note, since the HTSRs are selected stepwise, the digital normalization algorithm may be improved by first considering the longest HTSRs, therefore yielding a subset S_κ containing quite less HTSRs. In a similar fashion, the input HTSRs may also be first sorted according to their decreasing quality, estimated as the sum of Phred scores. Such a preliminary sorting results in an optimized subset S_κ containing long and high-quality HTSRs. This quality-based sorting step can be set when performing the particular normalization procedure implemented in the program `ORNA` (Durai & Schulz, 2019).

Finally, given a small lower-bound threshold κ' ($< \kappa$), the digital normalization approach can be easily extended to return a subset $S_{\kappa',\kappa} \subset S_\kappa$ that does not contain the HTSRs with *k*-mer coverage lesser than κ'. By means of the CMS that stores the number of occurences of all canonical *k*-mers deriving from S_κ, every HTSR r such that $c_k(r) \leq \kappa'$ is simply discarded, therefore yielding a final subset $S_{\kappa',\kappa}$ whose overall coverage depth lies between $\varepsilon\kappa'$ and $\varepsilon\kappa$. This second-pass strategy thus discards HTSRs containing many infrequent *k*-mers, e.g. incorrectly sequenced, artefactual or contaminating HTSRs. Unfortunately, this useful approach is implemented by a few program tools only, e.g. `BBnorm`.

The main purpose of `ROCK` is to pool these complementary strategies to obtain a complete and efficient digital normalization procedure. It was designed to be used as a preprocessing step prior to performing fast genome *de novo* assembly. `ROCK` is a command-line program that may consider up to 15 input FASTQ files containing HTSRs from different single-end (SE) or paired-end (PE) sequencing. Both lower and upper thresholds κ' and κ can be set by the user, as well as the length $k \leq 31$. As the CMS size is a key parameter to obtain fast running times while keeping accurate results (see Implementation), `ROCK` is also able to automatically calculate it, provided that the number of distinct canonical *k*-mers is specified.

Implementation

`ROCK` first reads the input FASTQ file(s) to store the location of each HTSR into a container $\mathbf{M}[x][y]$ based on `std::map`, where $\mathbf{M}[x][y]$ corresponds to the yth SE/PE HTSR(s) with the quality score x (i.e. the sum of Phred scores). Each entry $\mathbf{M}[x][y]$ contains the minimum necessary information to quickly get the associated SE/PE HTSR(s), i.e. the index of the corresponding file(s) and the index (i.e. offset from the start of the file) of the SE/PE HTSR(s). Using such a structure enables to quickly traverse every HTSR in their decreasing quality order. However, as each traversed HTSR is read using its offset-related values stored in $\mathbf{M}[x][y]$, input FASTQ files should be uncompressed.

The CMS is a two-dimensional $\lambda \times m$ array $\mathbf{C}[i][j]$, for which each of the λ rows is associated to a *mod prime* hashing function h_i on $[0, m]$, with $m =$ `UINT_MAX` (i.e. $2^{32} - 1$). Every *k*-mer is associated with one entry $\mathbf{C}[i][j]$ for each row i, where j is the hashing value returned by h_i for that *k*-mer. To add a canonical *k*-mer occurence into the CMS, each of its λ associated entries is incremented by one. As a consequence, when $\kappa \leq 255$, the CMS is a two-dimensional array of `unsigned char` with a memory footprint of 4λ Gb; when $\kappa > 255$, the `unsigned short` type is used, with twice RAM usage.

By definition, the number η of occurences of any given canonical *k*-mer in the CMS is expected to be the minimum value $\tilde{\eta}$ among the λ associated entries. However, $\tilde{\eta}$ can be strictly higher than the true number η of occurences with a probability p_{FP} (Cormode & Muthukrishnan, 2005), henceforth a false positive (FP) probability. Following Kim et al. (2019), when $\lambda = 1$, the probability that an arbitrary entry of **C** is strictly higher than a threshold θ after inserting n distinct *k*-mers is $\pi_{\theta,n} = 1 - \sum_{z \leq \theta} b_{n,1/m}(z)$, where $b_{n,p}$ is the probability mass function of the binomial distribution $B(n,p)$. Therefore, one gets $p_{\text{FP}} = \pi^{\lambda}_{\theta,n}$ with $\theta = \kappa$ (or $\theta = \kappa'$ if $\kappa' \neq 0$). By default, ROCK uses $\lambda = 4$ and a FP probability cutoff $\tau = 5\%$, which can cope with up to $n = 10$ billions distinct canonical *k*-mers while verifying $p_{\text{FP}} < \tau$ with any $\kappa > \kappa' > 1$. However, when the number n of distinct canonical *k*-mers is provided by the user, ROCK automatically computes the smallest integer λ such that $\pi^{\lambda}_{\theta,n} < \tau$, which often results in $\lambda = 1$ with WGS data from small genomes (see Example).

After instantiating a CMS with all entries set to 0, ROCK performs the digital normalization procedure by traversing **M**$[x][y]$ from the highest to the lowest quality x. For every SE/PE HTSR(s) r, the number α of canonical *k*-mer occurences $\tilde{\eta} \leq \kappa$ is obtained by querying the CMS, and next compared to the total number β of canonical *k*-mers. If $2\alpha > \beta$, then the median of the *k*-mer occurence values is strictly higher than κ, and r is/are not selected; otherwise, r is/are selected and the corresponding *k*-mers are added in the CMS. When $\kappa' \neq 0$, the second pass is next performed on the selected HTSRs following the same approach. ROCK runs the entire procedure using only a unique thread. At the end, all selected HTSRs are written into FASTQ-formatted output file(s). Of note, to avoid mixing up HTSRs that derive from different WGS strategies (e.g. different insert sizes), each input file corresponds to its own output file.

Example

To illustrate the performances of ROCK on a real-case high coverage WGS dataset, we considered the Illumina PE sequencing (run accession SRR2079909) of the *Clostridium botulinum* ATCC 9564 genome (assembly accession GCA_001273225), of length l = 3,813,606 base pairs (bps). The two FASTQ files were first processed using AlienTrimmer (Criscuolo & Brisse, 2013) to trim off 5'/3' regions containing many sequencing errors (i.e. Phred score cutoff $Q = 20$) or stretches of identical nucleotides. After discarding too short (i.e. < 100 bps) and unpaired HTSRs, a total of 4,973,401 PE HTSRs remained (2,756,008,939 bps, average length $\ell = 277$ bps), corresponding to a coverage depth of ~722×.

To reduce this high coverage depth to c = 50×, ROCK (v1.9.6) was run with $k = 25$ and $\kappa = c/\varepsilon \approx 45$. To assess the optimal CMS size λ, the total number n = 105,584,331 of distinct canonical *k*-mers was estimated using ntCard (Mohamadi et al., 2017), and next specified to ROCK, leading to $\lambda = 1$. In theory, using $\kappa = 45$ is expected to yield a subset of $cl/(2\ell) \approx 344{,}000$ PE HTSRs totaling $cl \approx 191$ Mbps. Obtained results are summarized in Table 1.

Table 1: Running times (min:sec), and numbers of PE HTSRs and corresponding base pairs (bps) returned by ROCK (on 1 thread) with $\kappa = 45$ and varying κ' values.

	run. time	**no. PE HTSRs**	**no. bps**
$\kappa' = 0$	3:16	424,040	244,188,971
$\kappa' = 2$	3:22	386,665	227,069,995
$\kappa' = 4$	3:23	371,834	220,241,468
$\kappa' = 6$	3:24	370,336	219,612,591
$\kappa' = 8$	3:24	369,994	219,471,025

For comparison sake, comparable standard digital normalizations ($\kappa = 45$ and $k = 25$) were also carried out on the same computer (AMD Epyc 2.2 GHz processor, 128 Gb RAM) using other dedicated tools (see Statement of need). They returned slightly larger subsets, containing from 498,899 (275 Mbps; `BBnorm`) to 693,978 (367 Mbps; `normalize-by-median`) PE HTSRs, with slower running times as compared to `ROCK`, varying from 5:17 (`BBnorm` on 12 threads) to 21:29 (`normalize-by-median`).

Authors' contribution

AC devised the project, the main conceptual ideas and algorithmic improvements. VL coded most parts of `ROCK`. NJ implemented the data structures with VL, and participated to the coding. TK and VL ran benchmarks to assess and improve the overall performance of `ROCK`. All authors provided critical feedback to optimize the program. AC wrote the manuscript in consultation with VL and TK.

References

Brown, C. T., Howe, A., Zhang, Q., Pyrkosz, A. B., & Brom, Y. H. (2012). A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data. *arXiv*, 1203.4802v2. https://arxiv.org/abs/1203.4802v2

Bushnell, B. (2014). BBnorm: Kmer-based error-correction and normalization tool (from the BBTools package). In *SourceForge repository*. https://sourceforge.net/projects/bbmap/

Cormode, G., & Muthukrishnan, S. (2005). An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, *55*, 29–38. https://doi.org/10.1016/j.jalgor.2003.12.001

Criscuolo, A., & Brisse, S. (2013). AlienTrimmer: a tool to quickly and accurately trim off multiple short contaminant sequences from high-throughput sequencing reads. *Genomics*, *102*(5–6), 500–506. https://doi.org/10.1016/j.ygeno.2013.07.011

Crusoe, M. R., Alameldin, H. F., Awad, S., Boucher, E., Caldwell, A., Cartwright, R., Charbonneau, A., Constantinides, B., Edvenson, G., Fay, S., Fenton, J., Fenzl, T., Fish, J., Garcia-Gutierrez, L., Garland, P., Gluck, J., González, I., Guermond, S., Guo, J., … Brown, C. T. (2015). The khmer software package: enabling efficient nucleotide sequence analysis [version 1; peer review: 2 approved, 1 approved with reservations]. *F1000Research*, *4*, 900. https://doi.org/10.12688/f1000research.6924.1

Durai, D. A., & Schulz, M. H. (2019). Improving in-silico normalization using read weights. *Scientific Reports*, *9*, 5133. https://doi.org/10.1038/s41598-019-41502-9

Kim, K., Jeong, Y., Lee, Y., & Lee, S. (2019). Analysis of Counting Bloom Filters Used for Count Thresholding. *Electronics*, *8*(7), 779. https://doi.org/10.3390/electronics8070779

Mohamadi, H., Khan, H., & Birol, I. (2017). ntCard: a streaming algorithm for cardinality estimation in genomics data. *Bioinformatics*, *33*(9), 1324–1330. https://doi.org/10.1093/bioinformatics/btw832

Wedemeyer, A., Kliemann, L., Srivastav, A., Schielke, C., Reusch, T. B., & Rosenstiel, P. (2017). An improved filtering algorithm for big read datasets and its application to single-cell assembly. *BMC Bioinformatics*, *18*, 324. https://doi.org/10.1186/s12859-017-1724-7

Zhang, Q., Pell, J., Canino-Koning, R., Howe, A. C., & Brown, C. T. (2014). These Are Not the K-mers You Are Looking For: Efficient Online K-mer Counting Using a Probabilistic Data Structure. *PLoS ONE*, *9*(7), e101271. https://doi.org/10.1371/journal.pone.0101271