ROCK: digital normalization of whole genome sequencing data

Véronique Legrand¹, Thomas Kergrohen², Thomas Joly², and Alexis Criscuolo⁽¹⁵⁾

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

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Due to advances in high-throughput sequencing technologies, generating whole genome sequencing (WGS) data with high coverage depth (e.g. ≥ 500×) 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 \( S_\kappa \) such that the coverage depth induced by the HTSRs in \( S_\kappa \) 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 (≈ \( \varepsilon \kappa \)), 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 \( S_\kappa \) (Cormode & Muthukrishnan, 2005; Zhang et al., 2014). By means

---

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 $S_k$) can be approximated by the median number of occurrences of the $k$-mers derived from $r$. The algorithm iterates over all HTSRs $r$ to progressively fill $S_k$: when $c_k(r) \leq \kappa$, $r$ is added into $S_k$ and the CMS is updated with every $k$-mers of $r$. At the end, the coverage depth induced by the selected HTSRs in $S_k$ is therefore expected to be upper-bounded by $\varepsilon \kappa$, where $\varepsilon = \ell/(\ell - k + 1)$ and $\ell$ 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 BlgNorm (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 $S_k$ 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 $S_k$ 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 $\kappa'$ ($<\kappa$), the digital normalization approach can be easily extended to return a subset $S_{\kappa',k} \subset S_k$ that does not contain the HTSRs with $k$-mer coverage lesser than $\kappa'$. By means of the CMS that stores the number of occurrences of all canonical $k$-mers deriving from $S_k$, every HTSR $r$ such that $c_k(r) \leq \kappa'$ is simply discarded, therefore yielding a final subset $S_{\kappa',k}$ 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 R0CK 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. R0CK 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 $\kappa'$ and $\kappa$ 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), R0CK is also able to automatically calculate it, provided that the number of distinct canonical $k$-mers is specified.

Implementation

R0CK 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 $y$th 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 $\lambda$ rows is associated to a mod prime hashing function $h_i$ on $[0,m]$, with $m=\text{INT\_MAX}$ (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 occurrence into the CMS, each of its $\lambda$ 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 $4\lambda$ Gb; when $\kappa > 255$, the unsigned short type is used, with twice RAM usage.
By definition, the number $\eta$ of occurrences of any given canonical k-mer in the CMS is expected to be the minimum value $\tilde{\eta}$ among the $\lambda$ associated entries. However, $\tilde{\eta}$ can be strictly higher than the true number $\eta$ of occurrences with a probability $p_{1-n}$ (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 $\eta$ after inserting $n$ distinct k-mers is $\pi_{\eta,n} = 1 - \sum_{z \leq \theta} b_{n,1/m}(z)$, where $b_{n,p}$ is the probability mass function of the binomial distribution $B(n, p)$. Therefore, one gets $p_{1-n} = \pi_{\eta,n}$ with $\theta = \kappa$ (or $\theta = \kappa'$ if $\kappa' \neq 0$).

By default, ROIK 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 $p_{1-n} < \tau$ with any $\kappa > \kappa' > 1$. However, when the number $n$ of distinct canonical k-mers is provided by the user, ROIK automatically computes the smallest integer $\lambda$ such that $\pi_{\eta,n} < \tau$, which often results in $\lambda = 1$ with WGS data from small genomes (see Example).

After instantiating a CMS with all entries set to 0, ROIK 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 $\alpha$ of canonical k-mer occurrences $\tilde{\eta}$ of $\kappa$ is obtained by querying the CMS, and next compared to the total number $\beta$ of canonical k-mers. If $2\alpha > \beta$, then the median of the k-mer occurrence values is strictly higher than $\kappa$, 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. ROIK 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 ROIK 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 (Crisculo & 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 $l = 277$ bps), corresponding to a coverage depth of $\sim 722 \times$.

To reduce this high coverage depth to $c = 50 \times$, ROIK (v1.9.6) was run with $k = 25$ and $\kappa = c/e \approx 45$. To assess the optimal CMS size $\lambda$, the total number $n = 105,584,331$ of distinct canonical k-mers was estimated using nCard (Mohamadi et al., 2017), and next specified to ROIK, leading to $\lambda = 1$. In theory, using $\kappa = 45$ is expected to yield a subset of $cl/(2\kappa) \approx 344,000$ PE HTSRs totaling $c \approx 191$ 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 ROIK (on 1 thread) with $\kappa = 45$ and varying $\kappa'$ values.

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

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 R0CK, 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 R0CK. 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 R0CK. All authors provided critical feedback to optimize the program. AC wrote the manuscript in consultation with VL and TK.

References


