RedOak: a reference-free and alignment-free structure for indexing a collection of similar genomes

Clément Agret [1,2,5¶], **Annie Chateau** [1,4], **Gaetan Droc** [2], **Gautier Sarah** [3], **Manuel Ruiz** [2,4], and **Alban Mancheron** [1,4]

1 LIRMM, Univ Montpellier, CNRS, Montpellier, France **2** Cirad, UMR AGAP, Avenue Agropolis, Montpellier, France **3** INRA, UMR AGAP, 2 Place Pierre Viala, Montpellier, France **4** Institut de Biologie Computationnelle, Montpellier, France **5** CRIStAL, Centre de Recherche en Informatique Signal et Automatique de Lille, Lille, France ¶ Corresponding author

DOI: 10.21105/joss.04363

Software

- Review
- Repository
- Archive

Editor: Lorena Pantano

Reviewers:

- @swatimanekar
- @samhorsfield96

Submitted: 15 February 2022
Published: 28 December 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

Here we present `RedOak`, a reference-free and alignment-free software package that allows for the indexing of a large collection of similar genomes. `RedOak` can also be applied to reads from unassembled genomes, and it provides a nucleotide sequence query function. Our method is about the analysis of complete genomes from the 3000 rice genomes sequencing project, but our indexing structure is generic enough to be used in similar projects. This software is based on a *k*-mer approach and has been developed to be heavily parallelized and distributed on several nodes of a cluster. The source code of our `RedOak` algorithm is available at RedOak.

Statement of need

`RedOak` may be really useful for biologists and bioinformaticians expecting to extract information from large sequence datasets.

The indexation of complete genomes is an important stage in the exploration and understanding of data from living organisms. Complete genomes, or at least a set of sequences representing whole genomes, *i.e.*, draft genomes, are becoming increasingly easy to obtain through the intensive use of high-throughput sequencing. A new genomic era is coming, therein not only being focused on the analyses of specific genes and sequences regulating them but moving toward studies using from ten to several thousands of complete genomes per species. Such a collection is usually called a pan-genome [Computational Pan-Genomics (2016)](Golicz et al., 2016). Within pan-genomes, large portions of genomes are shared between individuals. This feature could be exploited to reduce the storage cost of the genomes.

Based on this idea, this paper introduces an efficient data structure to index a collection of similar genomes in a reference- and alignment-free manner. A reference-free and alignment-free approach avoids the loss of information about genetic variation not found in the direct mapping of short sequence reads onto a reference genome (Computational Pan-Genomics, 2016). Furthermore, the method presented in this paper can be applied to next-generation sequencing (NGS) reads of unassembled genomes. The method enables the easy and fast exploration of the presence-absence variation (PAV) of genes among individuals without needing the time-consuming step of *de novo* genome assembly nor the step of mapping to a reference sequence.

Complexity

In this part, we present the time and space complexity of the algorithm, using the notations below: Total number of distinct k-mers: $N = K$ Total number of core k-mers: $N^* = K^*$ Total number of shell k-mers: $N^+ = K^+$ Total number of cloud k-mers: $N^- = K^-$ Number of instances running in parallel: np Size in bits of a memory word: u

Theorem1. The space needed for indexing n genomes is equal to $2k_2N + N^+(n+u) + O(4^{k_1}n)$ bits.

If k_1 is defined as $k_1 = \frac{\log(N)-\log(\log(N))+O(1)}{2}$,then the memory space required by RedOak to index the k-mers of n genomes is increased by $N(2, k_2 + n) + o(n, N)$ bits.

Theorem 2. The time needed to index the N distinct k-mers of n genomes is $O(nNk)$.

Theorem 3 Assuming that the number of genomes per indexed k-mer follows a Poisson distribution of parameter λ (where λ is the average number of genome sharing a k-mer), the size of N is $O(\frac{nm}{\lambda})$.

Proof Since the run time clearly depends on the number of indexed k-mers, let us use a simple model to approximate the time complexity. Suppose that each genome has m distinct k-mers and that each k-mer has a fixed probability p_i to be shared exactly by i genomes out of n. The total number of indexed k-mers is then

$$N = n\sum_{i=1}^{n}\frac{p_i m}{i} = nm\sum_{i=1}^{n}\frac{p_i}{i}$$

.

Implementation

RedOak is implemented in C++ and its construction relies on parallelized data processing. A preliminary step, before indexing genomes, is performing an analysis of the composition in k-mers of the different genomes. During this step, k-mer counting tools could be involved and their performance is crucial in the whole process(Manekar & Sathe, 2018). We looked for a library allowing us to handle a large collection of genomes or reads, zipped or not, working in RAM memory, and providing a sorted output. Indeed, RedOak uses libGkArrays-MPI from private communication, Mancheron et al. which is based on the Gk Arrays library (Nicolas Philippe et al., 2011). The Gk array library and libGkArrays-MPI are available under CeCILL license (GPL compliant). The libGkArrays-MPI library is highly parallelized with both Open~MPI and OpenMP.

To manipulate k-mers, the closest method is Jellyfish (Marcais & Kingsford, 2011). This approach is not based on disk but uses memory and allows the addition of genomes to an existing index. However, we did not use it because in the output, k-mers are in "fairly pseudo-random" order and "no guarantee is made about the actual randomness of this order" (see the Jellyfish documentation).

Value of k and k_1, in most of the k-mer based studies, the k-mer size varies between 25 (with reference genome) and 40 (without reference genome). The value of this parameter can be statistically estimated as stated in (N. Philippe et al., 2009).

The k_1 prefix length in our experiments has been defined on the basis of analytic considerations presented in (Park et al., 2009) but can be arbitrarily fixed to some value between 10 and 16, which respectively leads to an initial memory allocation from 8MiB to 32GiB, equally split across the running instances of RedOak. Setting a higher value is not necessary; otherwise, it may allocate unused memory.

To efficiently store and query the k-mers, each k-mer is split into two parts: its prefix of size k_1 and its suffix of size k_2, with $k_1 + k_2 = k$. Actually, the k-mers are clustered by their common prefix, and for each cluster, only the suffixes are stored. The choice of the value of k_1 minimizing memory consumption is guided by both analytic considerations (Park et al., 2009) and empirical estimation. Each k-mer can be associated a bit vector of size n to denote the presence (1) or absence (0) in each genome. Structure1. Representation of the data structure used to index the k-mers from n genomes. The (A) The green array represent core k-mers, (B) Blue cells the shell k-mers, and (C) in orange the cloud k-mers Structure2.

The concrete representation of the data structure used to store the k-mers having the same prefix is shown in (Structure 2). The k-mers absent from all genomes are obviously deduced from present k-mers and thus are not physically represented (and they all share the same 0-filled bit vector). The k-mers present in all genomes (core k-mers) are simply represented by a sorted vector where each suffix is encoded by its lexicographic rank (array (A)). These k-mers share the same 1-filled bit vector. The other k-mers (shell k-mers) are represented by an unsorted vector where each suffix is encoded by its lexicographic rank (array (C)). To each suffix is associated its presence/absence bit vector (array (3)). The order relationship between the suffixes is stored in a separate vector (array (B)).

In the `RedOak` implementation, both the core and shell k-mer suffixes are stored using $\frac{\lceil 2\,k_2 \rceil}{8}$ bytes each. The remaining unused bits are set to 0. This choice greatly improves the comparison time between k-mers suffixes. Moreover, because the presence/absence bit vectors are all of size n (the number of genomes), `RedOak` provides its own implementation for that structure, which removes the need to store the size of each vector. This implementation also emulates the 0-filled and 1-filled bit vector (arrays (4) and (5) of the Structure 1).

The choice of this data structure was guided by the desire to allow genome addition without having to rebuild the whole structure from scratch. Indeed, indexing a new genome can be represented by some basic operations on sets. First, it is obvious that the only case where the set of core k-mers expands is when the first genome is added. The other updates of the core k-mers occur and only lead to the removal of some k-mers from this set.

Example

Analysis of presence–absence variation (PAV) of genes among different genomes is a classical output of pan-genomic approaches. `RedOak` has a nucleotide sequence query function (including reverse complements) that can be used to quickly analyze the PAV of a specific gene among a large collection of genomes. Indeed, we can query, using all k-mers contained in a given gene sequence, the index of genomes. For each genome, if the k-mer is present in any direction we increment the score by 1. If the k-mer is absent but the preceding k-mer (overlapping on the first $k - 1$ nucleotides) is present, we note that there is an overlap, but `RedOak` does not increase the score. If the score divided by the size of the query sequence is greater than some given threshold, then we admit that the query is present in the genome. As an example, we indexed the 67 rice genomes with `RedOak` using $k = 30$, and we accessed the PAV of all the genes from Nipponbare and one gene from A. *Thaliana* using a threshold of 0.9. The gene Pstol, which controls phosphorus-deficiency tolerance. For a specific genome (GP104), we were able to detect the gene presence of the gene Pstol.

We need to keep in mind that this score under-estimates the percentage of identity. Indeed, let us suppose that the query sequence (of length ℓ) can be aligned with some indexed genome with only one mismatch, then all the k-mers (of the query) overlapping this mismatch may not be indexed for this genome. This implies that only one mismatch can reduce the final score by $\frac{\ell-k}{\ell}$, whereas the percentage of identity is $\frac{\ell-1}{\ell}$. In this experiment, a query having a score 0.9 can potentially be aligned with a percentage of identity greater than 97%.

Results

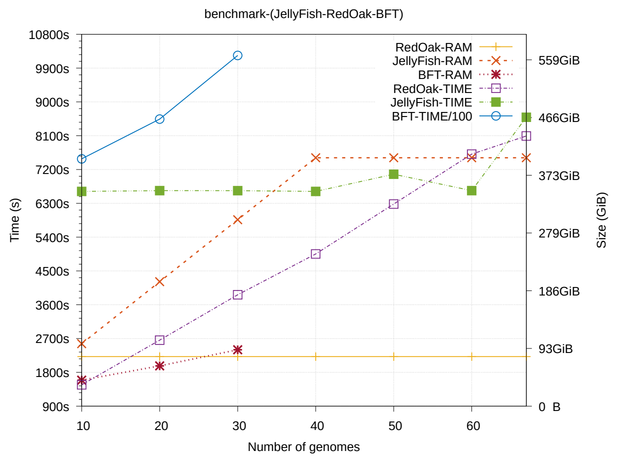

Figure 1: Performance comparison between `RedOak`, `Jellyfish` and `BFT` for the index build step.

The size of the data set was successively set to 10, 20, 30, 40, 50, 60 and 67 genomes out of the original data set (x-axis). A dot represents the wall-clock run time (y-axis) or the RAM usage (y2-axis) required to build the index. The colors represent the software used: `RedOak`, `Jellyfish` or `BFT`. For `BFT`, we divided the construction time by 100 to fit our figure Figure 1.

References

Computational Pan-Genomics, C. (2016). Computational pan-genomics: Status, promises and challenges [Journal Article]. *Brief Bioinform*. https://doi.org/10.1093/bib/bbw089

Golicz, A. A., Batley, J., & Edwards, D. (2016). Towards plant pangenomics [Journal Article]. *Plant Biotechnol J*, *14*(4), 1099–1105. https://doi.org/10.1111/pbi.12499

Manekar, S. C., & Sathe, S. R. (2018). A benchmark study of k-mer counting methods for high-throughput sequencing [Journal Article]. *Gigascience*, *7*(12). https://doi.org/10.1093/gigascience/giy125

Marcais, G., & Kingsford, C. (2011). A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, *27*(6), 764–770. https://doi.org/10.1093/bioinformatics/btr011

Park, G., Hwang, H.-K., Nicodème, P., & Szpankowski, W. (2009). Profiles of Tries. *Journal on Computing*, *38*(5), 1821–1880. https://doi.org/10.1137/070685531

Philippe, N., Boureux, A., Brehelin, L., Tarhio, J., Commes, T., & Rivals, E. (2009). Using reads to annotate the genome: influence of length, background distribution, and sequence errors on prediction capacity. *Nucleic Acids Research*, *37*(15), e104. https://doi.org/10.1093/nar/gkp492

Philippe, Nicolas, Salson, M., Lecroq, T., Léonard, M., Commes, T., & Rivals, E. (2011). Querying large read collections in main memory: a versatile data structure. *BMC Bioinformatics*, *12*(1), 242. https://doi.org/10.1186/1471-2105-12-242