SeqBio 2018

Rouen, 19-20 novembre

Monday 19th November

Tuesday 20th November

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Dichotomic Selection on words : a probabilistic analysis

Julien Clément, Brigitte Vallée

CNRS, Laboratoire GREYC, Université de Caen, ENSICAEn , CNRS
***Corresponding author**: brigitte.vallee@unicaen.fr

Abstract

The dichotomic search is one of the most basic tool for locating the position of a target value x within a sorted list of n elements. This scheme, that has a classical "divide-and-conquer" flavour, is a good algorithmic compromise in many situations, because it is straightforward to implement and nonetheless guarantees a $\Theta(\log n)$ number of comparisons between x and the n elements of the list. In this paper, we are interested in the case when the list and the target values are *words*, that are emitted by a *source*. Then, the comparison between two words is the usual (lexicographic) comparison, and the elementary operation is the comparison between symbols. The performance of the dichotomic selection on words is determined by the total *number of symbol comparisons*.

In this context, the dichotomic selection locates a word inside a list L of n words (sorted in the lexicographic order). This is the basis for an efficient implementation of searching in a suffix array. This structure is a widely used index structure in text algorithmics [see the works of Crochemore-Hancart-Lecroq (2007) and Gusfield (1997)]. In this context, the list is fixed and many target values are searched: it is then natural to consider the list sorting as a precomputation step.

We then follow the book of Crochemore-Hancart-Lecroq which explains that the sorting precomputation is not actually sufficient to build an efficient search procedure: one needs a supplementary precomputation step which determines the longest common prefixes between the words of the list L. With these two precomputation steps on hand, the technique becomes very efficient: The complexity for searching for a string of length m in a list of n strings takes $O(m + \log n)$ comparisons between symbols.

This talk will adopt a probabilistic point of view and analyses the performance of such methods on average, in terms of the total number of symbol comparisons, when the cardinality n of the list L tends to ∞. As in previous works of the authors, the probabilistic model is quite general and based on the parameterization of sources. In this context, we precisely study the mean complexity of the algorithm (in terms of the number of symbol comparisons); we first analyse (on average) the cost of the precomputation step that determines the longest common prefixes; then, we compare (on average) the complexity of the present algorithm to the complexity of the optimal algorithm (searching a word x inside the trie $T(L)$ built on the sequence L of words).

Our methods are based on the (Dirichlet) generating functions associated with the source, and deals with various analytic tools, already used in the authors in their previous works on the “realistic” analysis of classical searching and sorting algorithms.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Minimum Segmentation for Pan-genomic Founder Reconstruction in Linear Time

Tuukka Norri[1], Bastien Cazaux[1]*, Dmitry Kosolobov[1] and Veli Mäkinen[1]

[1] *Department of Computer Science, University of Helsinki, Helsinki, Finland*
***Corresponding author**: bastien.cazaux@helsinki.fi

Abstract

Given a threshold L and a set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of m haplotype sequences, each having length n, the minimum segmentation problem for founder reconstruction is to partition the sequences into disjoint segments $\mathcal{R}[i_1{+}1, i_2], \mathcal{R}[i_2{+}1, i_3], \ldots, \mathcal{R}[i_{r-1}{+}1, i_r]$, where $0 = i_1 < \cdots < i_r = n$ and $\mathcal{R}[i_{j-1}{+}1, i_j]$ is the set $\{R_1[i_{j-1}{+}1, i_j], \ldots, R_m[i_{j-1}{+}1, i_j]\}$, such that the length of each segment, $i_j - i_{j-1}$, is at least L and $K = \max_j\{|\mathcal{R}[i_{j-1}{+}1, i_j]|\}$ is minimized. The distinct substrings in the segments $\mathcal{R}[i_{j-1}{+}1, i_j]$ represent founder blocks that can be concatenated to form K founder sequences representing the original $\mathcal{R}$ such that crossovers happen only at segment boundaries. We give an optimal $O(mn)$ time algorithm to solve the problem, improving over earlier $O(mn^2)$. This improvement enables to exploit the algorithm on a pan-genomic setting of haplotypes being complete human chromosomes, with a goal of finding a representative set of references that can be indexed for read alignment and variant calling. We implemented the new algorithm and give some experimental evidence on the practicality of the approach on this pan-genomic setting.

An extended version of this abstrat can be found in [1].

References

[1] Tuukka Norri, Bastien Cazaux, Dmitry Kosolobov, and Veli Mäkinen. Minimum segmentation for pan-genomic founder reconstruction in linear time. In *18th International Workshop on Algorithms in Bioinformatics, WABI 2018, August 20-22, 2018, Helsinki, Finland*, pages 15:1–15:15, 2018.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Integrating Parallelism into Text Indexing

Johannes Fischer[1]

[1] *TU Dortmund, Germany*

Abstract

We describe some of our recent advances on the parallel construction of full-text indexes in shared and distributed memory systems. We focus in particular on wavelet trees and suffix arrays. We also describe how queries can be accelerated if the index is held in such systems, with good speed-ups and/or low communication overhead.

SeqBio 2018
Rouen, 19-20 novembre

Extended abstract

Development of indexing compressed structure for analyzing a collection of similar genomes: application to rice

Clément AGRET[1,2], Annie CHATEAU [1], Gaetan DROC [2], Alban MANCHERON [1], Manuel RUIZ [2]

[1] *LIRMM,CNRS and Université de Montpellier, 161 rue Ada, F-34095 Montpellier, France*
[2] *CIRAD, UMR AGAP, Avenue Agropolis, F-34398 Montpellier, France*
[3] *INRA, UMR AGAP, F-34060 Montpellier, France*
***Corresponding author**: agret@lirmm.fr*

Abstract

As the cost of DNA sequencing decreases, the high throughput sequencing technologies become more and more accessible to many laboratories. Consequently, new issues emerge that require new algorithms including tools for indexing and compressing thousands of genomes, as for example the 3000 rice genomes project [1], for which we are peculiarly interested in.

Indexing an unassembled genomes amounts to indexing a set of k-mers. Unassembled genomes can be considered as very large texts. We can refer them to indexable dictionary problem which consists in storing a set of words w. In our case k-mers are overlapping words based on a simple alphabet $\Sigma = \{A, C, G, T\}$. In order to make our index the most operational we propose two additionnal operations $rank_s(i)$ and $select_s(i)$. The function $rank_s(i)$ returns the number of elements (s) in the range $[0, i]$ and $select_s(i)$ returns the position of the i^{th} element.

Keywords

Index — Genomes — Rice

1. Introduction

The indexation of complete genomes is an important stage in the exploration and understanding of data from living organisms. An efficient index should provide a quick answer to the following questions:

– How many times a given pattern does appear in the genome?
– Which are the positions of a given pattern?
– What is the pattern length at the ith position in the genome?

The common way to structure index and compress one genome is to use the Burrows-Wheeler Transform (BWT) [2] with the FM-index [3] on BWT sequences for requests. If you want to index several genomes with one reference genome you may use MuGI [4]. To build MuGI index they store the reference in compact form (4 bits to encode single char), a variant database, one bit vector for each variant and an array kMA keeping information about each k-mers.

The main problem with MuGI is that it does not work with our datas. For our 3000 genomes, the VCF is too large to be handle by our clusters.

2. Methods

We hypothesized that in a similar genome collection add a new genome similar to adding only a few new k-mers. This hypothesis is validated after an experimentally defined threshold of 50 genomes.

As described in Figure 1, For all possible prefixes of length k_1, we use a 4^{k_1} array (1) of pointers to a structure that represent the suffixes of all k-mers having the same prefix.

Associated to each k-mer prefixes, we use two binary vectors of size 4^{k_2} which represent all its associated suffixes ; Let denote these two vectors associated to prefix $pref$ as $\mathcal{K}^+_{pref}$ and $\mathcal{K}^-_{pref}$.

The $\mathcal{K}^+_{pref}$ color green in the vector (2) represents set of core k-mers having prefix $pref$, where $\mathcal{K}^+_{pref}[i]$ is true if and only if the k-mer starting with the prefix $pref$ and ending with the i^{th} suffix in the lexicographic order occurs in all the indexed genomes.

Similarly, the $\mathcal{K}^-_{pref}$ color orange in vector (2) represents set of variable k-mers having prefix $pref$, where $\mathcal{K}^-_{pref}[i]$ is true if and only if the k-mer starting with the prefix $pref$ and ending with the i^{th} suffix in the lexicographic order occurs in at least one but not all of the indexed genomes.

Trivially, given a k-mer w starting with prefix $pref$ and ending with the i^{th} suffix in lexicographic, if w doesn't occur in any genome, we have $\mathcal{K}^+_{pref}[i] = \mathcal{K}^-_{pref}[i] = false$; If w occurs in at least one genome, we have either $\mathcal{K}^+_{pref}[i] = true$ or $\mathcal{K}^-_{pref}[i] = true$.

For each variable k-mer w, we define a binary vector $\mathcal{G}_w$ of length n (3) such that $\mathcal{G}_w[i]$ is true if and only if w occurs in the i^{th} indexed genome.

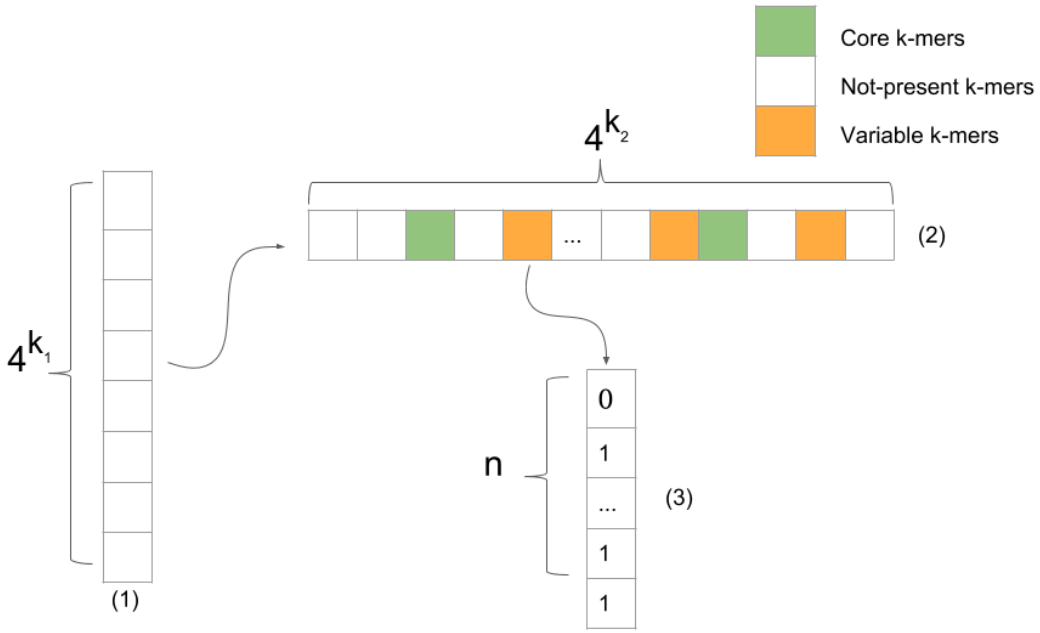


Figure 1. Representation of our structure to index the k-mers of n genomes. We computationaly define a prefix length k_1 for k-mers and denote as k_2 the k-mers suffix length such that $k_2 := k - k_1$.

In order to represent the binary vectors $\mathcal{K}^+_{pref}$ and $\mathcal{K}^-_{pref}$ we only store the indices of their true values in two arrays K^+_{pref} and K^-_{pref} in Figure 2. Since there are 4^{k_2} available k-mer suffixes, we need $2\,k_2$ bits to store each indices. For K^+_{pref}, indices are sorted by construction (the structure is initialized when indexing the first genome and core k-mers can either remain in the set of core k-mers or move to the set of variable k-mers whenever a new genome is

added to the index). Contrarily, the set of variable k-mers is not sorted by construction. Since 1/ each indices are stored using $2\,k_2$ bits (which does generally not correspond to the size of a standard integer) and 2/ for each variable k-mer w, we have an associated binary vector $\mathcal{G}_w$, sorting the K^-_{pref} is heavily time consuming or requires the use of an additional pointer for each variable k-mer, which is also heavily memory consuming. As a compromise, we choose to define an auxiliary vector O^-_{pref} using 16 bits words (instead of 32 or 64 bits for pointer) which stores the indices order of suffixes stored in K^-_{pref}. This way, the i^{th} stored variable k-mer (in increasing order) having prefix $pref$ is $K^-_{pref}[O^-_{pref}[i]]$. Sorting O^-_{pref} doesn't require anymore moving values of K^-_{pref} nor their associated $\mathcal{G}_*$ vector.

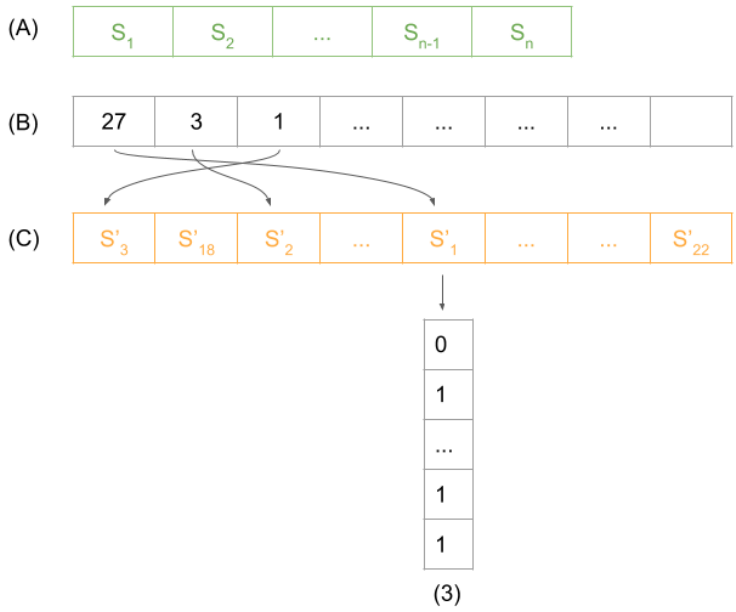


Figure 2. Representation of our vectors. (A) Vector of core suffixes $\mathcal{K}^+_{pref}$. This vector contain all position of the suffixes shared by all genomes (all green boxes in Figure 1). This vector is sorted by lexicographic order. (C) Vector of variable suffixes $\mathcal{K}^-_{pref}$. This vector contains all position of the suffixes not shared by all genomes (all orange boxes in Figure 1). This vector is unsorted. (B) Vector call O^-_{pref} allowing to get the Order of position of $\mathcal{K}^-_{pref}$ we use this table to quickly answer the question where is the i th suffix in the variable (which is not shared by all genomes). (3) is a binary vector of length n $\mathcal{G}_w$. In this example $\mathcal{G}_w$ (w=pref.s'_1)

$$\text{Let's take } \begin{cases} n = \text{Genome Number} \\ \mathcal{N} = \text{Number total of k-mers} \\ \mathcal{N}^* = \text{Number of distincts K-mers} \\ |\mathcal{K}^-_{pref}| = \mathcal{N}^- (\text{Number of k-mers in variable genome}) \\ |\mathcal{K}^+_{pref}| = \mathcal{N}^+ (\text{Number of k-mers in the core genome}) \\ \mathcal{N}^* = \mathcal{N}^+ + \mathcal{N}^- \\ f(\mathcal{G}) = o(|\mathcal{G}|) \\ g(\mathcal{G}) = o(|\mathcal{G}|) \end{cases}$$

The size of the RedOak index.

$$\begin{aligned} |index| &= \lambda\, 4^{k_1}\, 64 + 2\, k_2\, \mathcal{N}^* + (|\mathcal{G}| + 16)\, \mathcal{N}^- \\ &\overset{?}{=} \lambda\, 4^{k_1}\, 64 + 2\, k_2\, \frac{\mathcal{N}}{|\mathcal{G}|} \left(f(\mathcal{G}) + g(\mathcal{G})\right) + (|\mathcal{G}| + 16)\, \frac{\mathcal{N}}{|\mathcal{G}|}\, g(\mathcal{G}) \\ &\overset{?}{=} \lambda\, 4^{k_1}\, 64 + 2\, k_2\, \mathcal{N}\, \frac{f(\mathcal{G}) + g(\mathcal{G})}{|\mathcal{G}|} + \mathcal{N}\, g(\mathcal{G}) + 16\, \mathcal{N}\, \frac{g(\mathcal{G})}{|\mathcal{G}|} \\ &\overset{?}{=} \lambda\, 4^{k_1}\, 64 + \mathcal{N}\, g(\mathcal{G}) + o\big(k_2\, \mathcal{N}\big) \end{aligned}$$

The number of bits by nucleotides.

$$\begin{aligned} B/\mathcal{N} &= \frac{\lambda\, 4^{k_1}\, 64 + \mathcal{N}\, g(\mathcal{G}) + o\big(k_2\, \mathcal{N}\big)}{\mathcal{N}} \\ &\overset{?}{=} \frac{\lambda\, 4^{k_1}\, 64}{\mathcal{N}} + g(\mathcal{G}) + o(k_2) \\ &\overset{?}{=} \frac{\lambda\, 64\, \mathcal{N}^*}{\mathcal{N}\, \log \mathcal{N}^*} + g(\mathcal{G}) + o(k_2) \\ &\overset{?}{=} \frac{\lambda\, 64\, \frac{f(\mathcal{G}) + g(\mathcal{G})}{|\mathcal{G}|}}{\log \mathcal{N}^*} + g(\mathcal{G}) + o(k_2) \\ &\overset{?}{=} o\left(\frac{1}{\log \mathcal{N}^*}\right) + g(\mathcal{G}) + o(k_2) \\ &\overset{?}{=} g(\mathcal{G}) + o\left(k_2 + \frac{1}{\log \mathcal{N}^*}\right) \end{aligned}$$

3. Results and Discussion

We present a structure which proposes a solution to index and compress very repetitive sequences over small alphabet in texts using k-mers. k-mers are factors of length k in the considered sequences. We built a 4^{k_1} array, where $k_1 < k$, and each entry, namely an array, is indexed by a prefix of size k_1 of existing k-mers. In each prefix array we insert a 4^{k_2} bit vector which represents all possible k-mers begining with the considered prefix.
We will use libGkArray [5] to query a large read collections and update our structure. We chose libGkArray instead of JellyFish [6] or KMC (any versions) [7]. LibGkArray library work in main memory and is parallelized with openmpi therefore they use GPLv3 licence.
Results are promising on 10 genomes and we are actually performing intensive tests on more complex datasets.

References

[1] ZK Li and Rutger et al. The 3,000 rice genomes project. GigaScience, 3(1):7, 12 2014.

[2] Giovanna Rosone and Marinella Sciortino. The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words, pages 353–364. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[3] Paolo Ferragina and Giovanni Manzini. An experimental study of a compressed index. Information Sciences, 135(1):13–28, 2001.

[4] Agnieszka Danek, Sebastian Deorowicz, and Szymon Grabowski. Indexes of large genome collections on a pc. PLOS ONE, 9(10):1–12, 10 2014.

[5] Nicolas Philippe, Mikaël Salson, Thierry Lecroq, Martine Léonard, Thérèse Commes, and Eric Rivals. Querying large read collections in main memory: a versatile data structure. BMC Bioinformatics, 12(1):242, jun 2011.

[6] G. Marcais and C. Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics, 27(6):764–770, mar 2011.

[7] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. jan 2017.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Indexing de Bruijn graph with minimizers

Antoine Limasset*[1], Fatemeh Almodaresi[2], Rayan Chikhi[1] and Rob Patro[2]

[1] *Univ. Lille, CNRS, Inria, UMR 9189 - CRIStAL - F-59000, Lille, France*
[2] *Department of Computer Science, Stony Brook University*
***Corresponding author**: antoine.limasset@gmail.com

Abstract

A simple but fundamental need when dealing with genomic sequences is to be able to index very large sets of fixed length words (k-mers) and to associate information to these sequences (origin, abundance, strand, score etc. . .). As trivial as this need may seem, computationally challenging instances are extremely common in Metagenomic, pangenomic and even for the study of single large genomes, where sets of dozens or hundreds billions k-mers need to be processed. We propose a novel data structure able to both test the membership and associate information to the k-mers of a De Bruijn graph in a very efficient and exact way. We wrote a proof of concept dubbed Blight available at https://github.com/Malfoy/Blight to assess the performances of our proposed scheme. We were able to index all the k-mers of a human genome with less than 8GB of memory ($\approx$ 24 bits per k-mer). Our proposed index is also designed to provide extremely fast and parallel operations, able to perform billions queries in minutes.

Context

The de Bruijn graph structure is increasingly used as an efficient mean to represent a set of k-mers of interest. Several previous studies focused on the representation of a set of k-mer (Gosamer [1], Minia [2], DBGFM [3]). If those structure are extremely memory efficient (A modified version of Minia [4] achieved the rate of 8.58 bit per k-mer on a human dataset) they do allow to associate information to k-mers. Recently, the usage of efficient minimal perfect hash function (MPHF) library allowed the indexation of billions of keys with moderate resources [5]. But such functions are not able to recognize alien keys that were not in the indexed set. If such a key is queried, the MPHF may return the position of an indexed key hence producing a false positive (FP) error. The trivial solution would be to associate to each k-mer, in addition to its associated value, the k-mer sequence itself. This way an alien key would be recognized. But such structure require $2 * k$ bits per k-mer which can be extremely expensive, especially for large k. In order to cope with this problem SRC [6] proposed the use of a binary fingerprint, in order to keep the FP rate as low as possible while presenting a low memory overhead. The fingerprint mechanism lead to a n bits per kmer overhead for a false positive rate around 1/2 n which can guarantee a very low amount of errors. Another proposition made by Pufferfish [7] is

to propose an exact structure also based on this MPHF in order to index specifically the k-mer of a de Bruijn graph. Their idea to handle the alien keys in a memory efficient way is to associate to each k-mer its position in the indexed De Bruijn graph. This led to a memory efficient and fast to query structure able to index a human genome with 12 GB (which represent approximatively 4 bytes per k-mer) while being two time faster than FM-index based tools as BWA [8].

Methods

In the Pufferfish scheme, the main memory usage come from the encoding of the positions of the k-mers in the graph as each position cost O(ln(Genome size)) to encode. We propose to improve this scheme by working on subgraphs in order to reduce the memory amount required to encode such positions. For this we will take advantages to the fact that overlapping k-mers tend to share minimizers [9] and that we can represent a set of n overlapping k-mers sharing a minimizer with a super-k-mer of length $n + k - 1$. This super k-mer 1representation were notably used by KMC2 [10] in order to highly reduce the disk usage of external memory k-mer counter. The idea to improve the Pufferfish scheme come in two steps. First we will split the k-mers of our de Bruijn graph according to their minimizers, and encode them as super-k-mers. This way we have to deal with order of magnitude smaller sequences sets that we will call buckets. For example with a minimizer size of 12 on a human genome graph counting 2.5 billions k-mers, the largest bucket presented only 121,452 nucleotides. Those buckets are henceforth order of magnitude smaller and the amount of bits necessary to encode a position into them will be drastically reduced: $log2(2.5 * 109) = 31$ where $log2(1.2 * 105) = 17$. In a second part we will encode the k-mers positions into their respective buckets, as we can know a k-mer's minimizer directly from its sequence. This lead to several improvements :

- The amount of bit used to encode the position is highly reduced
- The data locality of the query is greatly improved, as each minimizers use its own small structure that can fit in cache, several successive query will therefore be able to be also treated without cache miss
- The construction of the index may be done in parallel
- The membership queries can be highly optimized by using the graph structure

We implemented this method in a header-only C++ library without any dependencies in order to be easily usable for most users. The code is open-source and available at https://github.com/Malfoy/Blight.

Result

We were able to index all k-mers of a human genome with less than 8GB of memory (less than 24 bits per k-mer) and the index can be built in less than one hour on a 20 cores cluster. The query of the whole dataset against itself were done within 5 minutes on the same cluster.

References

[1] Thomas C Conway and Andrew J Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, 2011.

[2] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1):22, 2013.

[3] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T Simpson, and Paul Medvedev. On the representation of de bruijn graphs. *Journal of Computational Biology*, 22(5):336–352, 2015.

[4] Kamil Salikhov, Gustavo Sacomoto, and Gregory Kucherov. Using cascading bloom filters to improve the memory usage for de brujin graphs. *Algorithms for Molecular Biology*, 9(1):2, 2014.

[5] Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. *arXiv preprint arXiv:1702.03154*, 2017.

[6] Camille Marchet, Lolita Lecompte, Antoine Limasset, Lucie Bittner, and Pierre Peterlongo. A resource-frugal probabilistic dictionary and applications in bioinformatics. *Discrete Applied Mathematics*, 2018.

[7] Fatemeh Almodaresi, Hirak Sarkar, Avi Srivastava, and Rob Patro. A space and time-efficient index for the compacted colored de bruijn graph. *Bioinformatics*, 34(13):i169–i177, 2018.

[8] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[9] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

[10] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. Kmc 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Three Strategies for the Dead-Zone String Matching Algorithm

Jacqueline W. Daykin[1,2,3,5], Richard Groult[4,3], Yannick Guesnet[3], Thierry Lecroq[3], Arnaud Lefebvre[3], Martine Léonard[3], Laurent Mouchard[3], Élise Prieur-Gaston[3], Bruce Watson[5,6]

[1] *Department of Computer Science, Aberystwyth Univ., Wales & Mauritius*
[2] *Department of Informatics, King's College London, UK*
[3] *Normandie Univ., UNIROUEN, LITIS, 76000 Rouen, France*
[4] *Modélisation, Information et Systèmes (MIS), Univ. de Picardie Jules Verne, Amiens, France*
[5] *Department of Information Science, Stellenbosch Univ., South Africa*
[6] *CAIR, CSIR Meraka, Pretoria, South Africa*

Abstract

The string matching problem consists in finding one or, more usually, all the occurrences of a pattern in a text. It can occur for instance in information retrieval, bibliographic search and molecular biology. It has been extensively studied and numerous techniques and algorithms have been designed to solve this problem (see [1, 2]). We are interested here in the problem where the pattern is given first and can then be searched in various texts. Thus a preprocessing phase is only allowed on the pattern.

Most string matching algorithms use a mechanism known as the sliding window strategy to scan the text. As early as 1997 a new family of algorithms has been designed that do not fit in the sliding window strategy: it consists in first locating the window in the middle of the text, performing an attempt and then recursively applying the same procedure on the left part and on the right part of the text, while possibly excluding some parts of the text giving the "dead-zone" method [3]. Algorithms from this family are highly parallelizable.

This strategy has not attracted much attention. To address this, here we present three different methods for performing the symbol comparisons during the attempts and for computing the lengths of the shifts.

References

[1] C. Charras and T. Lecroq. *Handbook of exact string matching algorithms*. King's College London Publications, 2004.

[2] S. Faro and T. Lecroq. The exact online string matching problem: A review of the most recent results. *ACM Comput. Surv.*, 45(2):13:1–13:42, 2013.

[3] B. W. Watson and R. E. Watson. A new family of string pattern matching algorithms. In *PSCW 1997, Prague, Czech Republic*, pages 12–23, 1997.

SeqBio 2018
Rouen, 19-20 novembre

Extended abstract

SALZA: Algorithmic Information Theory and Universal Classification for Sequences

François Cayre[1]*, Nicolas Le Bihan[2], Marion Revolle[2]

[1] *Grenoble-INP/ GIPSA-Lab, Grenoble, France*
[2] *CNRS / GIPSA-Lab, Grenoble, France*
***Corresponding author**: first.last@grenoble-inp.fr

Abstract
This work is an attempt at the implementation of a practical tool called SALZA to accomplish various information-theoretic tasks on sequences. Traditional information theory relies on a probabilistic model of the data. When such a probabilistic model is unavailable or hard to devise, one may want to replace it with an algorithmic model. In this paper, the algorithmic model of the data is that of the well-known Lempel-Ziv primitive: we assume new data is to be expressed in terms of references to prior data.

SALZA enables a flexible specification of prior data and extracts information quantities based on the significance of the references to these prior data. The tool readily implements the computation of an information measure based on LZ77 [1] and a universal classifier based on the Ziv-Merhav relative coder [2] for the universal clustering of sequences. SALZA can be used out of the box to replace probabilistic learning routines in algorithms that need an independence measure (most notably, causality inference in artificial intelligence). All proofs are to be found elsewhere [3].

Keywords
Algorithmic information theory — Universal classification — Artificial intelligence

1. Algorithmic information theory

The crux of information theory is the definition of a conditional mutual information quantity which respects the chain rule and the data processing inequality. When seeking such a quantity for sequences, we naturally derive a universal semi-distance suitable for blind classification of sequences.

We consider finite sequences on a finite alphabet $\mathcal{A}$ and we acknowledge the usual, lexicographical ordering of these sequences. The empty sequence and the empty set are denoted as $\emptyset$. $\mathcal{A}^+$ is the set of finite, non-empty sequences and $\mathcal{A}^\star = \mathcal{A}^+ \cup \emptyset$. The length of a sequence and the cardinal of a set or an alphabet is denoted as $|.|$. In a set of $x_1, \ldots, x_n$ sequences, $x_{\leq k}$ denotes the first k of them and $x_{\leq 0} = \emptyset$.

Specifying prior information

Our generic primitive is that of the *longest* copy-paste (reference) to prior data. The key issue is to allow any arbitrary policy $\mathcal{R}$ for specifying prior information. Given the sequences $y, x_1, \ldots, x_n$ and a priori policy $\mathcal{R}$ (a subset of $y, x_1, \ldots, x_n$), SALZA will factorize y by searching for the longest word in $\mathcal{R}$. Specifying a given prior policy

$\mathcal{R}$ depends on the target application. In this paper, we consider two policies with respect to the current position of the input look-ahead pointer during the sequential LZ-based encoding of y:

1. $y|x_1, \ldots, x_n : \mathcal{R}$ is the past of y (the part of y that is already factorized) and the entirety of $x_1, \ldots, x_n$;
2. $y|^+x_1, \ldots, x_n : \mathcal{R}$ is the entirety of $x_1, \ldots, x_n$.

When left unspecified, the factorization is denoted as $y \wr x_1, \ldots, x_n$.

A SALZA factorization slices y into m symbols of the form $(s_i, l_i, z_i)_{1 \leq i \leq m}$:

$$y \wr x_1, \ldots, x_n = (s_1, l_1, z_1) \ldots (s_m, l_m, z_m).$$

In a SALZA symbol, s is a pointer to one of the prior sequences, l is a length and z is a positive integer. A SALZA symbol (s, l, z) may be either:

1. a literal: $s = y$, $l = 1$ and z is the literal in y that should be inserted into the output buffer;
2. a reference: $l > 1$ is the length of the longest word in $\mathcal{R}$ matching new data and, although they are not used here, s would be the sequence in which the word was found and z would be the offset from the start of s at which the copy-paste should begin.

The product of two factorizations (possibly with different prior sequences) is defined as the concatenation of the SALZA symbols:

$$y_1 \wr x_{1,1}, \ldots, x_{1,n_1} \times y_2 \wr x_{2,1}, \ldots, x_{2,n_2} =$$

$$(s_{1,1}, l_{1,1}, z_{1,1}) \ldots (s_{1,m_1}, l_{1,m_1}, z_{1,m_1}) \, (s_{2,1}, l_{2,1}, z_{2,1}) \ldots (s_{2,m_2}, l_{2,m_2}, z_{2,m_2}).$$

Definition 1.1 *SALZA joint factorization and LZ77 factorization.*

The joint factorization of $x_1, \ldots, x_n \in \mathcal{A}^\star$ is defined as the following product of factorizations:

$$x_1 \ldots \cdot x_n = \prod_{i=1}^{n} x_i | x_{\leq i-1}.$$

When a factorization is expected, $x_1 = x_1|\emptyset$ may therefore be used to denote the usual LZ77 factorization of x_1.

Expressing significance of references

Central to our approach is the set of lengths in a SALZA factorization:

$$\mathcal{L}_{y \wr x_1, \ldots, x_n} = \{l_i\}_{1 \leq i \leq m}.$$

Definition 1.2 *Admissible function.*

For a sequence x, $f : \mathbb{N}^\star \to [0, 1]$ is an admissible function iff:

1. *f is monotonically increasing, and*

Name	T	$f(l<T)$	$f(1)$	$f(l^\star)$
`count` ($\mathbb{1}$)	0	0	1	1
`threshold`	$l^\star$	0	0	1
`linear`	$1+2(l^\star-1)$	$\frac{l-1}{2(l^\star-1)}$	0	0.5
`quadratic`	$\sqrt{2(l^\star)^2-1}$	$\frac{l^2-1}{2((l^\star)^2-1)}$	0	0.5
`sigmoid`	$\alpha l^\star$	$\frac{1}{1+e^{-l+\alpha l^\star}}$	$\frac{1}{1+e^{\alpha l^\star}}$	0.5
`exponential`	$\frac{\log 2+l^\star \log\epsilon}{\log 2+\log\epsilon}$	$\exp\left(\log\epsilon-(l-1)\frac{\log 2+\log\epsilon}{l^\star-1}\right)$	ϵ	0.5

Table 1. List of available admissible functions in our reference implementation (option `-a`). Of theoretical interest is the constant `count` admissible function (denoted as $\mathbb{1}$ in the text) that may be used to emulate the computation of usual LZ-based complexities. By default, we use the `exponential` function. An admissible function may be centered on a "noise level" $l^\star$. A reasonable best guess for $l^\star$ is automatically used by default, but can be overriden with option `-l` $l^\star > 1$ (and T is updated accordingly by enforcing continuity of f). This makes `SALZA` suitable to manually explore the similarity at any "noise level" $l^\star$.

2. $\exists\ 0 < T < |x|\ , \forall\ l \geq T\ ,\ f(l) = 1.$

In the definition above, the value T acts as an internal threshold, above which all reference lengths are equally considered most meaningful.

Definition 1.3 *Relative `SALZA` similarity.*

Given an admissible function f and sequences $y, x_1, \ldots, x_n \in \mathcal{A}^*$, the relative `SALZA` similarity of y given $x_1, \ldots, x_n$, denoted $S_f(y \wr x_1 \ldots, x_n)$, is defined as:

$$S_f(y \wr x_1, \ldots, x_n) = |y| - \sum_{l \in \mathcal{L}_{y \wr x_1, \ldots, x_n}} (l-1) f(l). \tag{1}$$

The `SALZA` relative similarity is designed so that it degrades to usual complexities when $f = \mathbb{1}$ and it is low when the sequences are similar. It is bounded.

Lemma 1.1 $0 \leq S_f(y \wr x_1, \ldots, x_n) \leq |y|$.

`SALZA`-based information theory

Once our relative similarity operator is defined, we can use it to devise a full implementation of LZ77-based algorithmic information theory.

Lemma 1.2 *`SALZA` joint similarity and self-similarity.*

By Def. 1.1, given an admissible function f and sequences $x_1, \ldots, x_n \in \mathcal{A}^*$, the `SALZA` joint similarity is computed as:

$$S_f(x_1 \ldots \cdot x_n) = \sum_{i=1}^{n} S_f(x_i | x_{\leq i-1}).$$

By Def. 1.3, the order of the $x_1, \ldots, x_n$ does *matter. The notation for joint similarity gracefully degrades into that of the LZ77-based computation of the self-similarity $S_f(x_1)$.*

Definition 1.4 *SALZA conditional mutual similarity, asymmetric version.*
Given an admissible function f and sequences $x, y, z \in \mathcal{A}^*$, the SALZA conditional mutual similarity of x and y given z is defined as:

$$I_f(x : y|z) = S_f(z \cdot x) + S_f(z \cdot y) - S_f(z \cdot x \cdot y) - S_f(z).$$

If $I_f(x : y|z) = 0$, x and y are said to be dissimilar given z.

A fast way of performing basic computations is as follows:

Lemma 1.3 *Fast computation of $I_f(x : y|z)$.*

$$I_f(x : y|z) = S_f(y|z) - S_f(y|z, x).$$

Lemma 1.4 *Chain rule for SALZA similarity, asymmetric version.*
Given sequences $x, y, z, t \in \mathcal{A}^\star$,

$$I_f(x : y \cdot z|t) = I_f(x : y|t) + I_f(x : z|t, y).$$

Lemma 1.5 *Data processing inequality for SALZA, asymmetric version.*
Given sequences $x, y, z \in \mathcal{A}^\star$:

$$S_f(y|z) \leq 1 \implies I_f(x : y|z) = 0 \implies I_f(x : y) \leq I_f(z : y).$$

It can be shown that $S_{\mathbb{1}}$ satisfies the same requirements as in Sec. 7.1 of [4] to qualify as an information measure. Such an information measure is defined on the lattice of sequences, using the lexicographical order as $\leq$. For the sake of tractability, [4] defines two sets of sequences $A = \{z, x\}$ and $B = \{z, y\}$, and the lattice operators are approximated as $A \wedge B = zxy$ and $A \vee B = z$.

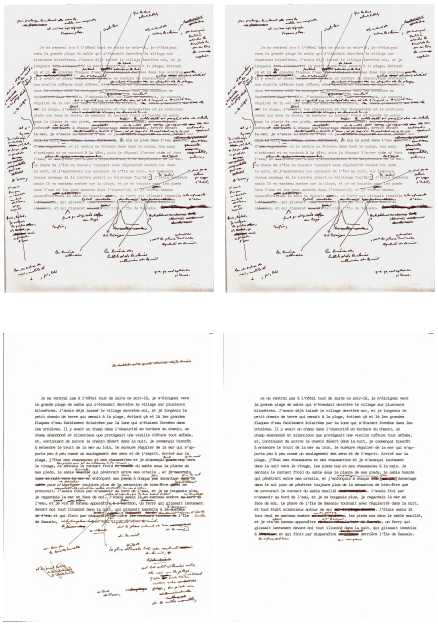
Theorem 1.1 *$S_{\mathbb{1}}$ complexity is an information measure in the sense of [4].*

1. *Normalization: $S_f(\emptyset) = 0$;*
2. *Monotonicity: $x \leq y \implies S_f(x) \leq S_f(y)$;*
3. *Approximate submodularity: $S_{\mathbb{1}}(z \cdot x) + S_{\mathbb{1}}(z \cdot y) \geq S_{\mathbb{1}}(z \cdot x \cdot y) + S_{\mathbb{1}}(z)$.*

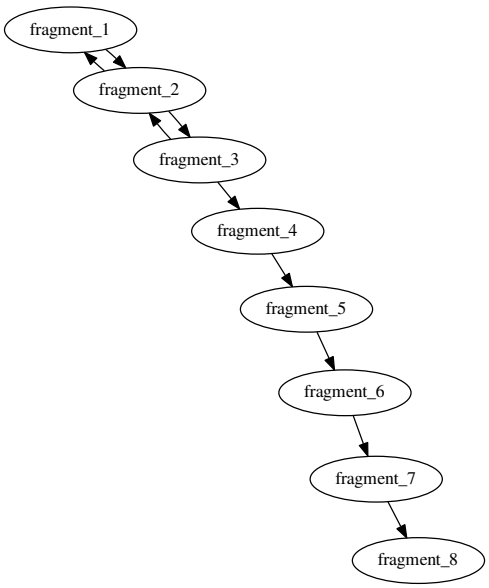
When $f \neq \mathbb{1}$, it is easy to devise dedicated counter-examples that violate the property of approximate submodularity (but normalization and monotonicity would still hold). This implies that I_f can not be guaranteed to be positive in the general case. However, we have used several datasets to assess departure from positivity and we did not witness any such departure in practice.

2. Universal causality inference

SALZA conditional mutual similarity can be plugged into the PC algorithm for causality inference [5]. In Fig. 1, we are able to (almost) recover the order of writing of 8 successive versions of a paragraph.



(a) 8 fragments of a text.



(b) Output of the PC algorithm[5].

Figure 1. Left: 8 successive drafts of a paragraph from *La Réticence* by author Jean-Philippe Toussaint (transcripts courtesy Prof. Thomas Lebarbé, TGIR HumaNum). Right: output of:
`./salza-driver --pcalg -i ~/dataset/toussaint --skel stable --dag --alpha 0.01 | neato -Tpdf > toussaint.pdf`

3. Universal classification

SALZA can be used to devise a universal semi-distance on sequences based on the Ziv-Merhav relative coder [2].

Definition 3.1 *NSD_f.*

Given an admissible function f, and two sequences $x, y \in \mathcal{A}^+$ such that $|x|, |y| > 1$, the normalized SALZA semi-distance, denoted NSD_f, is defined as:

$$NSD_f(x,y) = max\left\{\frac{S_f(x|^+y) - 1}{|x|}, \frac{S_f(y|^+x) - 1}{|y|}\right\}.$$

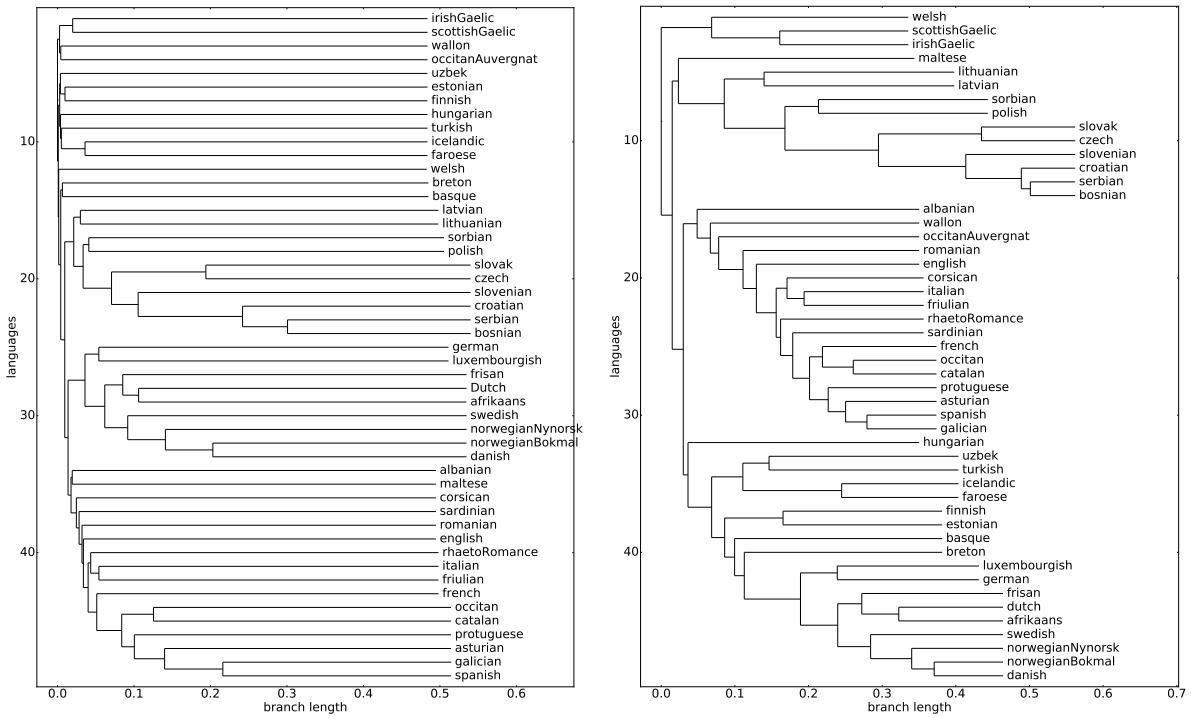
Theorem 3.1 *NSD_f is a normalized semi-distance.*

When compared with the state-of-the-art NCD [6] for blind clustering of sequences, the NSD not only performs faster but is also constantly as much discriminant as the NCD (often much more), see Fig. 2 (and [3] for more datasets).

Online resources

Preprint (with proofs), datasets, C code and binaries are located at:
https://forge.uvolante.org/stable/salza-driver/wikis/home

A future release will provide the fully multithreaded version of SALZA (parallel construction of search structures and parallel factorization) and include the PC algorithm used in Sec. 2.



(a) NCD/`xz`. **(b)** NSD, $l^\star = 2.12$.

Figure 2. `rights` dataset: Phenetic representation of various human writing systems. Texts are translations of the Universal Declaration of Human Rights.

References

[1] Jacob Ziv and Abraham Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23:337–343, May 1977.

[2] Jacob Ziv and Neri Merhav. A Measure of Relative Entropy Between Individual Sequences with Application to Universal Classification. *IEEE Transactions on Information Theory*, 39:1270–1279, Jul. 1993.

[3] Marion Revolle, François Cayre, and Nicolas Le Bihan. SALZA: Practical Algorithmic Information Theory and Sharper Universal String Similarity. *PREPRINT, Submitted*, 2018.

[4] Bastian Steudel, Dominik Janzing, and Bernhard Schölkopf. Causal Markov Condition for Submodular Information Measures. In *Proceedings of the 23rd Annual Conference on Learning Theory*, pages 464–476, Madison, WI, USA, Jun. 2010. OmniPress.

[5] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. Springer, 1993.

[6] Rudi Cilibrasi and Paul M.B. Vitányi. Clustering by Compression. *IEEE Transactions on Information Theory*, 51:1523–1545, Apr. 2005.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Practical fast exact pattern matching algorithm for highly similar sequences

Nadia Ben Nsira[1], Thierry Lecroq[1], Élise Prieur-Gaston[1]

[1] *Normandie Univ, UNIROUEN, LITIS, 76000 Rouen, France*

Abstract

Compared to traditional technologies, High-throughput sequencing technologies or *Next Generation Sequencing* (NGS) technologies have greatly increased sequencing throughput. Hence, this leads to an enormous number of highly similar DNA sequences. This collections of data are identical more than 99% to a reference sequence. Obviously, this opens the door for bioinformaticians to exhibit algorithms to handle efficiently the huge amount of available data. Naturally, storing, indexing and support for fast pattern matching have become important research topics.

Pattern matching can be carried out in two ways: off-line by using an index or on-line when indexing is not possible. Adopting the second method seems to be more suitable faced to the problem of lack of space to build an index. In our context, we chose to resolve the problem by scanning the whole set of sequences rather than index it. The main idea is to examine identical segments not repeatedly. Rather than iterating the input sequences sequentially, our main goal is to scan simultaneously the whole sequences. Thereby, classical algorithms can be extended easily, while considering the existence of eventual variations during scanning. Yet the sequences differ from another by a few number of differences such as *substitutions* or *single nucleotide variants* (SNVs), *indels*, *copy number variations* (CNVs) or *translocations* to name a few, we assume here that sequences include variations only of type *substitutions*.

We present efficient practical algorithms that solve exact pattern matching problem in a set of highly similar DNA sequences. We first present a method for exact single pattern matching when k variations are allowed in a window which size is equal to the pattern length. We then propose an algorithm for exact multiple pattern matching when only one variation is allowed in a window which size is equal to the length of the longest pattern. Experimental results show that our algorithms, though not optimal in the worst case, have good performances in practice [1].

References

[1] N Ben Nsira, T Lecroq, and É Prieur-Gaston. Practical fast exact pattern matching algorithm for highly similar sequences. In *Data mining from genomic variants and its application to genome-wide analysis 2018 jointly with IEEE BIBM 2018*. Accepted.

Abstract

Using chromosome conformation capture to assemble genomes to perfection

Nadège Guiglielmoni[1]*, Antoine Limasset[2], Romain Koszul[3], Jean-François Flot[1]

[1]*Department of Organismal Biology, Université libre de Bruxelles, Brussels, Belgium*
[2]*Bonzai team, CRIStaL lab, Université Lille 1,Villeneuve-d'Ascq, France*
[3]*Department of Genomes and Genetics, Institut Pasteur, France*
***Corresponding author**: Nadege.Guiglielmoni@ulb.be

Abstract

Despite a near-exponential increase in the number of eukaryotic genome projects over the last few years, the vast majority of them yield only "permanent drafts", i.e., genomes that remain heavily fragmented at the end of the project and are deposited as such in public databases. This is because the large and abundant repeats found in eukaryotic genomes make it extremely difficult to reach the "golden standard" of one contig per chromosome. Bleeding-edge chromosome conformation capture (3C) approaches appear as a promising path to overcome the repeat issue and may provide accurate and reliable eukaryotic genomes, thereby increasing the power of downstream analyses[1]. 3C-based scaffolders and reassemblers, such as SALSA[2] and GRAAL[3], have already demonstrated that contact genomics provides key insights into the puzzling challenge of reconstructing genomes. Improvements to existing computational pipelines are required however to enable de novo assembly of 3C data with maximal accuracy and completeness. As part of the Innovative Training Network (ITN) IGNITE, which focuses on comparative genomics of non-model, non-vertebrate organisms, we plan to create a scaffolder that uses chromosome conformation capture data to solve assembly graphs generated by genome assemblers such as Bwise[4]. We will then apply it to generate a chromosome-scale assembly of the genome of a chaetognath, a group whose taxonomic position remains highly controversial[5] and for which no genome sequence is presently available. To achieve this goal, the protocol of chromosome conformation capture will need to be fine-tuned for chaetognaths in order to obtain contact information and sequences from a single individual. We believe that chromosome conformation capture will be an efficient and economic solution to the genome assembly problem, with the additional advantage of yielding both an assembled genome sequence and a contact map as outputs.

References

[1] J.-F. Flot, H. Marie-Nelly and R. Koszul (2015) Contact genomics: scaffolding and phasing (meta)genomes using chromosome 3D physical signatures. *FEBS Letters* 589: 2966–2974

[2] J. Ghurye et al. (2018) Integrating Hi-C links with assembly graphs for chromosome-scale assembly. *bioRxiv* 261149

[3] H. Marie-Nelly et al. (2014) High-quality genome (re)assembly using chromosomal contact data. *Nature Communications* 5: 5695

[4] https://github.com/Malfoy/BWISE

[5] F. Marlétaz et al. (2006) Chaetognath phylogenomics: a protostome with deuterostome-like development. *Current Biology* 16: R577-R578

SeqBio 2018
Rouen, 19-20 novembre

Extended abstract

Détection sans alignement de recombinaisons V(D)J multi-chaînes

Mathieu Giraud[1], Mikaël Salson[1]

[1] *Univ. Lille, CNRS, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France*
Emails: contact@vidjil.org

Abstract
La grande diversité du répertoire immunitaire repose sur un mécanisme génétique appelé recombinaison V(D)J. De nombreux logiciels permettent d'analyser des données de séquençage haut-débit portant sur ces recombinaisons. Cependant, à notre connaissance, Vidjil-algo est le seul logiciel proposant une heuristique sans alignement afin d'accélérer le traitement de millions de séquences. Nous proposons ici une amélioration de notre heuristique et montrons que celle-ci permet d'aller environ 8 fois plus vite en occupant moins d'espace mémoire et sans perte de qualité de résultats.
Cette nouvelle heuristique est disponible ici : `https://gitlab.vidjil.org` (en développement)

Keywords
Spaced seeds, Aho-Corasick automaton, Alignment-free methods

1. Introduction

Les recombinaisons V(D)J sont des événements génétiques se produisant dans des cellules immunitaires immatures, des lymphoblastes. Ces recombinaisons sont à l'origine de la production d'une très grande diversité de récepteurs sur les lymphocytes B et T [1]. Une recombinaison V(D)J est le résultat d'un processus aléatoire qui a sélectionné un gène V (parmi quelques dizaines à une centaine), éventuellement un gène D, et un gène J (parmi une dizaine) à recombiner de manière contiguë sur le génome de ces lymphoblastes. À la jonction des gènes V, D et J, des nucléotides peuvent être supprimés et d'autres, alétatoires, peuvent être ajoutés (voir Figure 1), ce qui améliore encore cette diversité.

Figure 1. Exemple de recombinaison VDJ. Après suppression de quelques nucléotides à la jointure entre les gènes V, D et J, `AATA` est inséré entre le gène V et le gène D et `GCT` est ajouté entre le gène D et le gène J.

Les récépteurs des lymphocytes B et T jouent un grand rôle dans la reconnaissance des antigènes et donc dans la réponse immunitaire adaptative. En immunologie, connaître les recombinaisons V(D)J permet ainsi de qualifier voire quantifier une

telle réponse immunitaire. En outre, ces recombinaisons V(D)J étant issues d'un processus aléatoire, elles constituent un identifiant d'une population de cellules ayant le même lymphoblaste d'origine (on parle alors de *population clonale*, ou *clone*). En hémato-oncologie, recourir aux recombinaisons V(D)J comme identifiant d'un clone est particulièrement utile pour étudier l'évolution de cancers du sang, touchant ces lymphocytes.

Dans les années 2010, de nombreuses méthodes et logiciels ont été proposées pour la détection et la dénomination de recombinaisons V(D)J (voir par exemple [2]). Nous appelons *détection* le fait de déterminer si une recombinaison V(D)J est présente dans une séquence d'ADN et d'identifier le type de chaîne à laquelle cette recombinaison correspond. Nous appelons *dénomination* le fait de déterminer quels gènes ont été précisément choisis dans une recombinaison V(D)J ainsi que les suppressions et insertions au niveau de la jonction. La plupart de ces logiciels procèdent à la dénomination en même temps que la détection est faite et ceci pour chacune des séquences données en entrée.

À l'inverse, lorsque nous avons initialement publié notre logiciel Vidjil [3] nous avons choisi une approche différente : d'abord détecter des recombinaisons V(D)J, les regrouper par clone et enfin dénommer, pour les clones les plus abondants, les recombinaisons V(D)J. En effet, pour de nombreuses applications, il est inutile de connaître la dénomination exhaustive de chacun des clones (il peut y en avoir des dizaines ou des centaines de milliers dans un jeu de données de quelques millions de séquences).

Chez l'humain, les récepteurs des lymphocytes B ont une chaîne lourde (IGH) et une chaîne légère (IGλ ou IGκ), tandis que ceux des lymphocytes T ont soit deux chaînes TRα et TRγ, soit deux chaînes TRβ et TRδ. Une recombinaison V(D)J peut donc provenir de n'importe laquelle de ces chaînes. Notre algorithme précédent recherchait ainsi 16 types de recombinaisons différentes, incluant ces différents types de chaines ainsi que des recombinaisons *incomplètes* – dans une telle recombinaison, on cherche, d'une manière plus générale, un gène en 5' et un autre gène en 3' de la séquence, comme c'est le cas par exemple pour les recombinaisons DJ. De manière plus générale, si ℓ est le nombre de type de recombinaisons recherchées, et n la longueur de la séquence, notre algorithme de détection était en $O(\ell n)$. Nous présentons maintenant un algorithme en $O(n)$.

2. Détection linéaire de recombinaison V(D)J multi-chaînes

Pour *détecter* des recombinaison V(D)J dans une séquence donnée, des k-mers espacés sont utilisés afin de déterminer le nombre de hits parmi des gènes V et des gènes J. S'il existe dans la séquence un point à partir duquel le nombre de hits avec des gènes V d'un côté et avec des gènes J de l'autre côté est statistiquement significatif, une recombinaison V(D)J est alors *détectée*. Cette heuristique, déjà utilisée dans la version d'origine de notre algorithme, est conservée.

Initialement l'heuristique était relancée pour chaque type de recombinaison possible. Nous utilisons ici un automate d'Aho-Corasick adapté afin de détecter en une fois les recombinaison V(D)J de tous les types (y compris les recombinaisons incomplètes) en temps $O(n)$, avec n la taille de la séquence.

L'automate d'Aho-Corasick est particulièrement utile pour rechercher une banque de motifs dans une séquence. La complexité en temps de la requête est linéaire dans la taille de la séquence et indépendante de la taille de la banque de motifs [4]. Pour plus de détails sur cet automate, voir [5, Chap. 2.2]. Ici, la banque de motifs indexés par l'automate d'Aho-Corasick ne sont pas les gènes V, D, J eux-mêmes mais les graines espacées extraites de ces gènes. Nous stockons dans les états acceptants de l'automate un couple comportant le type de gène (V, D ou J) ainsi que la chaîne de laquelle il provient.

Plus formellement, une graine espacée u est une suite de caractères matchs `#` et jokers `-`. Notons $seed(w, u)$ l'instance de la graine u pour le mot w telle que $seed(w, u) = v$ avec, pour $0 \leq i < |w|$, $v_i = w_i$ si $u_i =$ `#` et $v_i =$ `-` sinon. On a ainsi $seed($`ATCG`$,$ `##-#`$) =$ `AT-G`. On définit $P(g, u)$ l'ensemble des mots qui sont un facteur d'un gène g avec la graine u: $P(g, u) = \{w \mid \exists i, seed(g_{i\ldots i+|u|-1}, u) = seed(w, u)\}$. Notons qu'il y en a au plus $O(|g|4^z)$, où z est le nombre de jokers dans w (en pratique nos graines ont au plus un joker, au milieu). Tous les mots de $P(g, u)$, pour l'ensemble des gènes g, sont ajoutés à l'automate d'Aho-Corasick. Cette première étape constitue l'indexation des données. Il est possible de moduler les graines espacées utilisées en fonction de la chaîne et du type de gène, ce qui n'était pas possible avec une simple table. Lorsque plusieurs types de gènes ou de chaînes correspondent à un état de l'automate, celui-ci est marqué comme ambigu.

Lorsqu'une séquence est requêtée, il suffit de la lire en suivant les transitions dans l'automate. Les états acceptant rencontrés indiquent le type de gène et la chaîne avec laquelle (ou lesquels) la séquence possède une similarité. Nous identifions alors les deux couples (type de gène, chaîne) les plus rencontrés. Un calcul de significativité détermine s'il y a un nombre suffisant d'un type de gène d'un côté et de l'autre type de gène de l'autre.

Pour chaque séquence ainsi traitée, Vidjil-algo attribue un identifiant constitué d'un facteur de longueur fixe dont la position centrale correspond à la position maximisant le nombre de graines d'un type de gène d'un côté et d'un autre type de gène de l'autre côté.

3. Résultats

Afin d'évaluer la capacité de notre heuristique à détecter des recombinaisons V(D)J, nous avons généré des données synthétiques pour tous les types de chaînes (évaluation de la sensibilité) ainsi que des données complètement aléatoires dans lesquelles nous ne devrions pas trouver de recombinaison V(D)J (évaluation de la spécificité). Nous n'avons considéré que des recombinaisons complètes, les autres logiciels étudiés ne détectant pas les recombinaisons incomplètes.

Les recombinaisons aléatoires ont été générées en effectuant toutes les recombinaisons possibles entre chaque combinaison de gène, V, D ou J. Pour chaque combinaison, 10 séquences ont été générées, chacune avec un nombre aléatoire d'insertions et de délétions au niveau des jonctions entre les gènes (moyenne à 5 et écart-type à 5) ainsi que 2 % de substitutions dans les données (afin de prendre en compte des artefacts de séquençage mais aussi des différences entre individus).

Nous avons comparé les résultats de notre heuristique avec MiXCR [6] et IgReC [7]. Ces logiciels effectuent dès le départ une étape de dénomination. Nous avons limité

l'analyse des performances à cette étape-là, bien qu'ils puissent faire des analyses plus approfondies. Ces logiciels produisent des résultats plus complets que ceux de Vidjil-algo puisqu'ils fournissent une dénomination pour chaque séquence donnée en entrée. À l'inverse Vidjil-algo commence par essayer de détecter une recombinaison V(D)J et, si c'est le cas, extrait un identifiant qui lui servira à clusteriser les séquences en clones. Seuls les 100 clones les plus abondants sont ensuite dénommés (cette étape-là n'a pas été discutée ici).

Tous les logiciels (IgReC commit 2ec3b7d51, MiXCR version 2.1.12, Vidjil-algo (old) version 2018.02 et Vidjil-algo commit dcc7643) ont été lancés sur un seul thread sur une machine équipée de 4 processeurs i7-8650U à 1.90GHz et de 32 Go RAM.

Les résultats pour les recombinaisons V(D)J générées aléatoirement sont présentés dans la Figure 2. Bien qu'elle construise l'automate de Aho-Corasick, cette version est plus économe en mémoire que MiXCR mais moins que IgReC qui a une consommation mémoire semblant linéaire dans le nombre de séquences analysées. À partir de fichiers d'environ 100 000 séquences, Vidjil-algo est bien plus rapide que ses concurrents, en particulier la version de l'heuristique que nous présentons ici. Même avec cette heuristique très rapide, ce nouvel algorithme est très sensible pour détecter des recombinaisons V(D)J dans des recombinaisons générées aléatoirement, souvent plus que MiXCR ou IgReC.

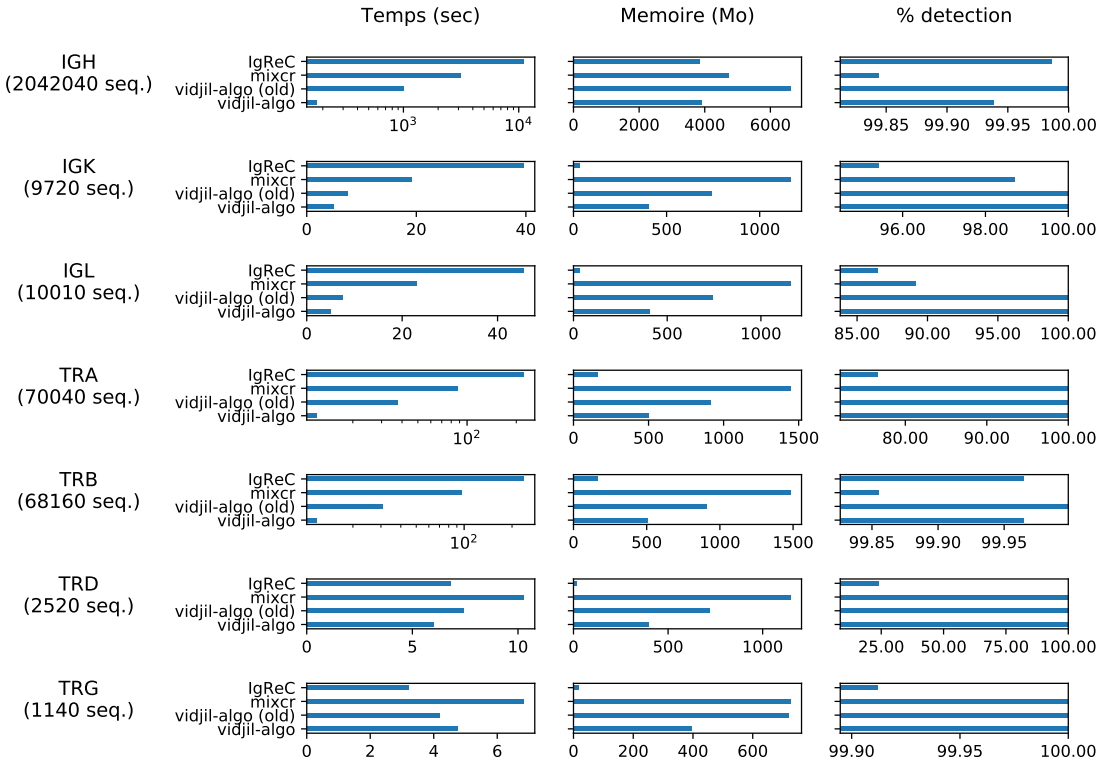


Figure 2. Évaluation de IgReC, MiXCR et Vidjil-algo sur les recombinaisons V(D)J générées aléatoirement. **Attention** dans certains cas l'axe des abscisses est logarithmique pour les graphiques de temps.

Enfin nous lançons les logiciels sur des séquences de longueur 350 à 450, tirées aléatoirement sur l'alphabet {`A`, `C`, `G`, `T`} (Figure 3). Le nouvel algorithme confirme

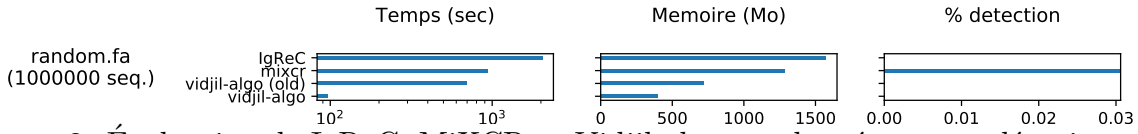


Figure 3. Évaluation de IgReC, MiXCR et Vidjil-algo sur des séquences aléatoires.

ses bonnes performances et ne reconnaît aucune recombinaison V(D)J parmi ces séquences aléatoires, tout comme IgReC et la précédente version de Vidjil-algo.

4. Conclusions

L'étude des recombinaisons V(D)J dans le cadre immunologique ou onco-hématologique ne nécessite pas toujours d'avoir une dénomination de chaque séquence. Il est utile de proposer des approches précises et rapides capables de détecter toutes les recombinaisons, puis de se concentrer sur la dénomination des clones les plus abondants.

Nous avons proposé une nouvelle heuristique sans alignement, plus rapide que la précédente d'un facteur près de 10, pour de gros jeux de données, tout en diminuant légèrement l'espace mémoire requis. Les performances de détection (sensibilité et spécificité) restent excellentes. Vidjil-algo est ainsi un logiciel extrêmement rapide pour la détection de millions de recombinaisons V(D)J et la dénomination des clones les plus abondants. Ce nouvel algorithme sera prochainement intégré à la version de production de Vidjil-algo et déployé dans la plateforme Vidjil utilisée par des laboratoires en immunologie et en onco-hématologie [8].

References

[1] S. Tonegawa. Somatic generation of antibody diversity. *Nature*, 302(5909):575–581, 1983.

[2] S. Afzal, I. Gil-Farina, R. Gabriel, et al. Systematic comparative study of computational methods for T-cell receptor sequencing data analysis. *Briefings in Bioinformatics*.

[3] M. Giraud, M. Salson, M. Duez, et al. Fast multiclonal clusterization of V(D)J recombinations from high-throughput sequencing. *BMC Genomics*, 15(1):409, 2014.

[4] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[5] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithmique du texte*. Vuibert, 2001.

[6] D. A. Bolotin, S. Poslavsky, I. Mitrophanov, et al. MiXCR: software for comprehensive adaptive immunity profiling. *Nature Methods*, 12(5):380–381, 2015.

[7] A. Shlemov, S. Bankevich, A. Bzikadze, and Y. Safonova. New algorithmic challenges of adaptive immune repertoire construction. In *RECOMBSeq*, 2016.

[8] M. Duez, M. Giraud, R. Herbert, et al. Vidjil: A web platform for analysis of high-throughput repertoire sequencing. *PLOS One*, 11(11):e0166126, 2016.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Utilisation de k-mers avec erreurs pour l'analyse de données Nanopore

Quentin Bonenfant, Laurent Noé, Hélène Touzet [1]*

[1] *Équipe BONSAI, CRIStAL, UMR 9189, Université de Lille, Villeneuve d'Ascq, France*
***Corresponding author**: quentin.bonenfant@univ-lille.fr*

Abstract

Beaucoup d'algorithmes pour l'analyse des reads courts sont basés sur des approches heuristiques à base de k-mers. Toutefois, l'utilisation de k-mers exacts peut entraîner une perte de sensibilité lorsque les séquences considérées présentent des taux d'erreurs élevés, comme c'est le cas avec les reads longs. Ce problème est d'autant plus patent quand on veut comparer des reads entre eux.

Dans l'équipe BONSAI, ont été développées des *graines avec erreurs*, qui permettent de retrouver toutes les sous-séquences communes avec un nombre borné d d'erreurs. Il s'agit des graines 01*0 [1]. L'idée derrière ces graines est de diviser la séquence à chercher en blocs de sorte que la distribution des erreurs ne soit plus aléatoire. Ces graines n'ont jamais été utilisées dans le contexte d'analyse des reads longs. Nous proposons ici leur utilisation pour la comparaison des reads dans l'objectif d'identifier des motifs communs entre reads longs.

Notre cas d'application est celui de la détection des séquences des adaptateurs de séquençage Nanopore. Nous avons montré que l'utilisation de ces graines à la place de k-mers exacts permettait une reconstruction plus précise des séquences des adaptateurs.

La méthode que nous proposons repose sur deux étapes: *l'identification* des k-mers composant potentiellement l'adaptateur, à l'aide d'une approche par comptage en tenant compte des erreurs, et *la reconstruction* de la séquence intégrale de l'adaptateur en utilisant une méthode d'assemblage gloutonne des k-mers identifiés. Nos résultats tendent à montrer que l'utilisation de graines avec erreurs permet d'obtenir des séquences consensus stable pour 80% des échantillons étudiés contre 40% avec les approches exactes, et ce pour un coût en temps très faible: de l'ordre de 10 à 20 secondes pour un échantillon de 10k reads.

References

[1] Christophe Vroland, Mikaël Salson, Sébastien Bini, and Hélène Touzet. Approximate search of short patterns with high error rates using the 01 0 lossless seeds. *Journal of Discrete Algorithms*, 37:3–16, March 2016.

Abstract

La co-évolution des protéines et le monde des virus

Alessandra Carbone[1]

[1] *Sorbonne Université - Laboratoire de Biologie Computationnelle et Quantitative*

Abstract

Une question fondamentale en biologie computationnelle est l'extraction de l'information évolutive à partir de séquences d'ADN. Cet information concerne les sites de liaison protéine-protéine et les propriétés mécaniques et allostériques des protéines. Nous présenterons une approche computationnelle pour l'analyse de co-évolution de paires de résidus dans les séquences protéiques. Elle a été appliquée à l'étude des interactions protéine-protéine dans le cadre d'un projet sur la reconstruction du réseau protéique des génomes viraux. Nous montrerons comment le réseau d'interaction des 10 protéines du génome du virus de l'hépatite C peut être reconstruit à une résolution de résidus / domaine et nous présenterons brièvement des travaux expérimentaux récents démontrant la fusion du VHC comme un mécanisme unique. Ce travail fournit une preuve de concept pour une exploration plus large des processus médiés par les protéines virales et souligne la coévolution comme un outil précieux pour guider la conception des inhibiteurs viraux. Dans un deuxième exemple, nous montrerons comme une généralisation de la méthode, appliquée au virus de l'hépatite B, nous donne des informations importantes sur les mutations primaires et secondaires de réponse à des drogues antivirales.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Optimizing early steps of long-read genome assembly

Pierre MARIJON[1]*, Maël KERBIRIOU[1], Jean Stéphane VARRÉ[2] and Rayan CHIKHI[2]

[1] *Inria, Université de Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France*
[2] *Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France*
***Corresponding author**: pierre.marijon@inria.fr

Abstract

As long-read genome assembly is becoming more and more prevalent, there is an increasing need to scrutinize the initial data processing steps of long-read genome assemblers. We found that the quality and efficiency of two key steps, read trimming and overlap computation, still have room for improvement. We propose two lightweight tools to improve these steps in order to increase results quality, save disk space, memory and computation time.

Long reads have a high error rate (10-20%) and many tools have been developed to correct them either with short-reads (hybrid correction) or only with long-reads (self correction). Long reads may have chimeras, i.e. when a read consists of two distant regions of the genome that have been incorrectly joined. Chimeras are typically detected and either split or removed by correction tools. We propose yacrd (yet another chimeric read detector) to optimize the process of finding chimeric regions and remove them prior to read correction. Yacrd is based on all-against-all read mapping and pile-up coverage, and shows significantly improved precision and running time compared to a state of the art tool (Table 1).

In the overlap-layout-consensus model of genome assembly, an overlap graph is built, where nodes are reads and edges correspond to overlaps between reads. Computing the edges of the graph is typically done via an all-versus-all mapping of the reads, which recovers similar regions (*matches*), e.g. using the minimap2 [1] tool. Many similar regions end up being found between reads, but not all of them are overlaps. Matches may be between inner regions of reads, or be to short to be usable as overlaps. We developed the fpa tool (Filter Pairwise Alignement) to perform fast and configurable filtering of all-versus-all read similarities, in order to keep only matches of interest (e.g. overlaps) see blog post for more details [2]. On a real Oxford Nanopore dataset, removing matches shorter than 2000 bp save 20% of disk space without lost of contiguity in miniasm[1] assembly.

	minimap2 + yacrd	DAScrubber
wallclock time (seconds)	48.13	365.79
precision	100.00%	87.70%
sensitivity	70.34%	71.16%

Table 1. On synthetic dataset minimap2 + yacrd is 7 times faster than DAScrubber[3] with same sensitivity and better precision.

References

[1] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, mar 2016.

[2] Pierre Marijon. How to reduce the impact of your paf file on your disk by 95%. https://blog.pierre.marijon.fr/binary-mapping-format/.

[3] Gene Myers. DAScrubber. https://dazzlerblog.wordpress.com/2017/04/22/1344/.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

BCOOL-Trans: accurate and variant-preserving correction for RNA-seq

Camille Marchet[1] and Antoine Limasset[1]

[1] *Univ. Lille, CNRS, Inria, UMR 9189 - CRIStAL - F-59000, Lille, France*

Abstract

Motivation

At first sight, sequencing data using Next Generation Sequencing does not involves high amounts of sequencing errors. However, looking more closely, those errors have drastic impacts on assembly tasks. They also bring noise to variant analysis. This is why error correction is a well-broached subject in genomics. The most efficient tools are currently based on k-mer spectrums. Surprisingly, few tools are dedicated to correction of transcriptomics data. Yet, such a data type requires specific developments. Firstly, contrary to genomics, coverage in transcriptomics is far from being uniform since genes have different expression levels. This is why this data cannot be handled with k-mer spectrum techniques. Secondly, eukaryotic transcriptomes undergo alternative splicing, thus usually convey different mRNA variants for a given gene. Such alternative sequences create a multiplicity of possible correct k-mers in a single context, which can be an extremely difficult situation for correctors.

Previous work

There is a plethora of methods ([1, 2, 3]), but at the exception of RCorrector [4], no tool is dedicated to accurately correcting RNA-seq data. In their publication, RCorrector's authors demonstrated the need of specific developments for RNA-seq by comparing their tool to state-of-the-art, genomics-oriented tools. In addition to focusing on RNA-seq, RCorrector brings interesting features by considering several correct k-mers at a position in order to take into account and preserve alternative splicing events. The idea is to set a local threshold on the k-mer counts in nodes of a De Bruijn Graph, in order to prune the less covered branches. Such a local k-mer threshold strategy was also automatically inserted for spurious k-mers removal in tools such as KisSplice [5]. Finally, more recent methods relying on graph such as LoRMA [6] are out of the scope of this work since they aim at correcting another type of sequencing data.

Contribution

We propose a new RNA-seq correcion method inspired by BCOOL [7], that proposes several enhancements of the graph usage for read correction. In particular, the BCOOL publication identifies several k-mer spectrum techniques limitations and tackles them by using De Bruijn graph's simple paths (unitigs) coverage and topological configuration instead of solely k-mers abundances. More into detail, BCOOL first builds a "reference" De Bruijn Graph with the solid (i.e. which count is higher than

a threshold) k-mers of the data set. The De Bruijn Graph is then compacted to a unitig graph [8], and unitigs with a low average k-mer abundance are removed. Then it uses topological patterns, such as dead ends, to identify the remaining putative errors and to remove them. Finally, reads are mapped on the cleaned graph and each read is corrected according to the most parsimonious path. In this work, we propose a new algorithm that is more suitable for transcriptomic data and a new tool following the BCOOL paradigm to correct transcriptomics data: BCOOL-Trans. This method takes into account in the correction process the possible occurrence of variants in the data set, such as alternative splicing, alternative start or end of the transcription, genomic variants. We propose the following new features, first on the graph construction and cleaning:

- Work with all k-mers of the original data set instead of applying a k-mer based threshold before constructing the De Bruin Graph. This seems realistic on a transcriptome (graphs can be built from dozens of billions k-mers [9]), and allows not to lose any k-mer *a priori* because of local shallow coverage. Then correction is performed only by relying on the topology and relative abundance.

- Let be n a node of the graph of unitigs. n is a dead-end if there is no outgoing node n' from n. Use large (ideally larger than half the size of reads) k-mers so that most errors create dead-ends in the graph while SNPs or alternative splicing create bubbles [5]. Correction then focuses on dead-ends.

- In BCOOL's algorithm, a dead-end is removed if it is shorter than a threshold. In BCOOL-Trans, we propose to use the relative coverage for dead-ends instead of length.

Contrary to Rcorrector, the coverage of a dead-end is computed using the average k-mer coverage of a unitig. It means that we use a less local information to conserve or remove parts of the graph. This way, we can keep short but biologically sound events if they are supported enough in the data-set. This can be the case for alternative start or end of the transcription, or variants that are not very well covered. Then, the modification of the mapping algorithm to fit the transcriptomics data can also bring improved results:

- In BCOOL, the mapping strategy is greedy. However, intricated bubbles and repeats are common in a transcriptome graph. A greedy choice could frequently lead to suboptimal or erroneous correction in such complex hubs. We propose to explore more deeply the different possible paths before to make a decision.

- Handle paired-end reads mapping so that if two reads can be linked by a single unitig, we can output paired-end read merging such as proposed for genomics (for instance [10]). Those precise, longer reads can be useful to improve various analysis steps or compensate for tools that do not handle paired-end data.

The expected outcome is that BCOOL-Trans introduces less errors during the correction phase and better reckognizes alternative variants from sequencing errors. Moreover BCOOL-Trans is designed to preserve genomic SNPs and indels whithin

transcripts that are biologically meaningful.

A second contribution, still a work in progress, is to demonstrate whether extremely accurate correction of short reads benefits to long read sequencing correction (TGS). Those long reads are reknown for their high and difficult error profile. Many tools from the litterature propose to use short reads for their correction. One reason is that reads such as those provided Oxford Nanopore Technologies contain non-systematic errors. Thus they require a thirdparty for their correction, this is why some tools use short read mapping onto long reads to perform correction [11]. Another way is to first assemble short reads into contigs, to be used as templates for correction [12, 13]. Other works propose to work directly on an assembly graph made of the short reads [6, 14, 15]. It is likely that errors impact the assembly phases. However, the accuracy of the short reads used for correction was never questioned in any of these works. We would like to assess the impact of correction with BCOOL-Trans as a prior to a good long read hybrid correction.

References

[1] Heng Li. Bfc: correcting illumina sequencing errors. *Bioinformatics*, 31(17):2885–2887, 2015.

[2] Li Song, Liliana Florea, and Ben Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome biology*, 15(11):509, 2014.

[3] Yongchao Liu, Jan Schröder, and Bertil Schmidt. Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data. *Bioinformatics*, 29(3):308–315, 2012.

[4] Li Song and Liliana Florea. Rcorrector: efficient and accurate error correction for illumina rna-seq reads. *GigaScience*, 4(1):48, 2015.

[5] Gustavo AT Sacomoto, Janice Kielbassa, Rayan Chikhi, Raluca Uricaru, Pavlos Antoniou, Marie-France Sagot, Pierre Peterlongo, and Vincent Lacroix. K is s plice: de-novo calling alternative splicing events from rna-seq data. In *BMC bioinformatics*, volume 13, page S5. BioMed Central, 2012.

[6] Leena Salmela, Riku Walve, Eric Rivals, and Esko Ukkonen. Accurate self-correction of errors in long reads using de bruijn graphs. *Bioinformatics*, 33(6):799–806, 2016.

[7] Antoine Limasset, Jean-Francois Flot, and Pierre Peterlongo. Toward perfect reads. *arXiv preprint arXiv:1711.03336*, 2017.

[8] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T Simpson, and Paul Medvedev. On the representation of de bruijn graphs. In *International conference on Research in computational molecular biology*, pages 35–55. Springer, 2014.

[9] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

[10] Brian Bushnell, Jonathan Rood, and Esther Singer. Bbmerge–accurate paired shotgun read merging via overlap. *PloS one*, 12(10):e0185056, 2017.

[11] Thomas Hackl, Rainer Hedrich, Jörg Schultz, and Frank Förster. proovread: large-scale high-accuracy pacbio correction through iterative short read consensus. *Bioinformatics*, 30(21):3004–3011, 2014.

[12] Ehsan Haghshenas, Faraz Hach, S Cenk Sahinalp, and Cedric Chauve. Colormap: Correcting long reads by mapping short reads. *Bioinformatics*, 32(17):i545–i551, 2016.

[13] Ergude Bao and Lingxiao Lan. Halc: High throughput algorithm for long read error correction. *BMC bioinformatics*, 18(1):204, 2017.

[14] Pierre Morisse, Thierry Lecroq, Arnaud Lefebvre, and Bonnie Berger. Hybrid correction of highly noisy long reads using a variable-order de bruijn graph. *Bioinformatics*, 1:10, 2018.

[15] Giles Miclotte, Mahdi Heydari, Piet Demeester, Stephane Rombauts, Yves Van de Peer, Pieter Audenaert, and Jan Fostier. Jabba: hybrid error correction for long sequencing reads. *Algorithms for Molecular Biology*, 11(1):10, 2016.

SeqBio 2018
Rouen, 19-20 novembre

Abstract

Utilisation de k-mers avec erreurs pour l'analyse de données Nanopore

Quentin Bonenfant, Laurent Noé, Hélène Touzet [1]*

[1] *Équipe BONSAI, CRIStAL, UMR 9189, Université de Lille, Villeneuve d'Ascq, France*
***Corresponding author**: quentin.bonenfant@univ-lille.fr*

Abstract

Beaucoup d'algorithmes pour l'analyse des reads courts sont basés sur des approches heuristiques à base de k-mers. Toutefois, l'utilisation de k-mers exacts peut entraîner une perte de sensibilité lorsque les séquences considérées présentent des taux d'erreurs élevés, comme c'est le cas avec les reads longs. Ce problème est d'autant plus patent quand on veut comparer des reads entre eux.

Dans l'équipe BONSAI, ont été développées des *graines avec erreurs*, qui permettent de retrouver toutes les sous-séquences communes avec un nombre borné d d'erreurs. Il s'agit des graines 01*0 [1]. L'idée derrière ces graines est de diviser la séquence à chercher en blocs de sorte que la distribution des erreurs ne soit plus aléatoire. Ces graines n'ont jamais été utilisées dans le contexte d'analyse des reads longs. Nous proposons ici leur utilisation pour la comparaison des reads dans l'objectif d'identifier des motifs communs entre reads longs.

Notre cas d'application est celui de la détection des séquences des adaptateurs de séquençage Nanopore. Nous avons montré que l'utilisation de ces graines à la place de k-mers exacts permettait une reconstruction plus précise des séquences des adaptateurs.

La méthode que nous proposons repose sur deux étapes: *l'identification* des k-mers composant potentiellement l'adaptateur, à l'aide d'une approche par comptage en tenant compte des erreurs, et *la reconstruction* de la séquence intégrale de l'adaptateur en utilisant une méthode d'assemblage gloutonne des k-mers identifiés. Nos résultats tendent à montrer que l'utilisation de graines avec erreurs permet d'obtenir des séquences consensus stable pour 80% des échantillons étudiés contre 40% avec les approches exactes, et ce pour un coût en temps très faible: de l'ordre de 10 à 20 secondes pour un échantillon de 10k reads.

References

[1] Christophe Vroland, Mikaël Salson, Sébastien Bini, and Hélène Touzet. Approximate search of short patterns with high error rates using the 01 0 lossless seeds. *Journal of Discrete Algorithms*, 37:3–16, March 2016.

SeqBio 2018
Rouen, 19-20 novembre

Extended abstract

LoRSCo: Long Reads Self-Correction

Pierre Morisse[1]*, Antoine Limasset[2], Camille Marchet[2], Arnaud Lefebvre[1], Pierre Peterlongo[3], Thierry Lecroq[1]

[1] *Normandie Univ, UNIROUEN, LITIS, Rouen 76000, France.*
[2] *Univ. Lille, CNRS, Inria, UMR 9189 - CRIStAL - F-59000, Lille, France.*
[3] *Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France.*
***Corresponding author**: pierre.morisse2@univ-rouen.fr

Abstract

Third generation sequencing technologies such as Pacific Biosciences and Oxford Nanopore allow the sequencing of long reads of tens of kbs, that are expected to solve various problems, especially in the genome assembly field. However, they also reach high error rates of 10 to 30%, and thus require efficient error correction. As first long reads sequencing experiments produced reads displaying error rates higher than 15% on average, most methods relied on the complementary use of short reads data to perform correction, in a hybrid approach. However, these sequencing technologies evolve fast, and now manage to produce long reads displaying error rates around 10-12% on average. As a result, correcting the long reads solely based on the information they contain, in a self-correction approach, is now an efficient alternative. Various self-correction methods were thus recently developed, but most of them either face scalability issues or require deep long reads coverage. We introduce LoRSCo, a new method for the self-correction of long reads that combines different efficient approaches from the state-of-the-art. Our experiments show that LoRSCo compares well to the state-of-the-art in terms of quality of the results, while achieving globally higher throughput, and low memory consumption. LoRSCo is available on GitHub: https://github.com/morispi/LoRSCo.

Keywords

long reads — correction — self-correction

1. Introduction

Third generation sequencing technologies Pacific Biosciences and Oxford Nanopore became widely used since their inception in 2010. In contrast to what second generation sequencing technologies offer, they allow the sequencing of much longer reads (tens of kb on average, and up to 882 kb) that are expected to solve various problems, most specifically in the genome assembly field. These long reads are however very noisy, reaching error rates of 10 to 30%, whereas short reads usually display error rates around 1%. The error profiles of these long reads are also much more complex than those of the short reads, as they are mainly composed of insertions and deletions, while short reads' are mostly composed of substitutions. As a result, efficient error correction is a mandatory step before making use of these reads in any kind of application.

As first long reads sequencing experiments resulted in highly erroneous long reads

(15-30% error rates on average), most of the efficient error correction methods relied on a hybrid strategy, using additional short reads data. Third generation sequencing technologies however evolve fast, and now manage to produce long reads reaching error rates of 10-12% on average. However, these error rates do not seem to significantly decrease anymore. As a result, error correction is still needed, but a self-correction approach can now efficiently be adopted, thus totally getting rid of the short reads data.

1.1 Related works

Due to the fast evolution of third generation sequencing technologies, and to the lower error rates they now reach, various efficient self-correction methods were recently developed. Most of them share the common first step of computing overlaps between the long reads, whether it is via mapping (Canu [1], MECAT [2], HALS [unpublished]) or via alignment (PBDAGCon [3], daccord [4]). Most method then build a directed acyclic graph (DAG) from the alignments, in order to compute consensus (PDAGCon, Canu, MECAT, HALS), after recomputing actual alignments of mapped regions, if necessary. Other methods rely on de Bruijn graphs, either built from small windows of the alignments (daccord), or directly from the long reads sequences with no alignment or mapping step at all (LoRMA [5]). However, methods relying on direct alignment of the long reads consume large amounts of time and memory, and therefore cannot scale to large genomes. Moreover, methods solely relying on de Bruijn graphs and avoiding the alignment step altogether usually require very deep long reads coverage, as the graphs are built for large values of k. Therefore, methods relying on overlap computing via a mapping approach seem to provide the most interesting results at the present times.

1.2 Contribution

We present LoRSCo, a new self-correction method that combines different approaches from the state-of-the-art into an efficient strategy. Like most efficient methods, LoRSCo starts by computing overlaps between the long reads using a mapping approach. This way, only matched regions further need to be aligned in order to compute consensus for a given read. Like in existing methods, the consensus is computed with the help of a DAG. However, the alignment of matched regions is performed via a multiple sequences alignment strategy based on partial order graphs (POA) [6]. This allows LoRSCo to directly build the DAG during the multiple alignment of matched regions, instead of performing alignment and graph construction separately. This multiple sequences alignment strategy also benefits from an efficient heuristic, based on k-mers chaining, allowing to reduce the time and memory footprints of the method, with little to no effect on the quality of the results. In a second step, once the consensus of a given read has been computed, LoRSCo makes use of local de Bruijn graphs to polish the corrected long read. This allows to refine the correction by removing remaining errors in weakly supported regions, that are, regions containing weak k-mers, and thus, to further reduce the error rate. Our experiments show that LoRSCo achieves comparable throughput and quality compared to state-of-the-art long read self-correction methods.

2. Methods

In order to initiate correction, LoRSCo starts by computing overlaps between the long reads via a mapping approach, using Minimap2 [7]. This allows LoRSCo to save both time and

memory, as resource-consuming alignments then only have to be computed between matched regions. After the overlapping step, all the reads are processed independently. For the sake of simplicity, we thus present the correction process for a single read.

Given a read A to correct, we define an alignment pile for A as a set of reads overlapping with A. This concept was originally introduced in daccord, but we slightly alter it for our purpose. Formally, we thus define an alignment pile as a set of tuples (R, Ab, Ae, Rb, Re, C) where R is a long read id, Ab and Ae represent respectively the start and the end positions of the alignment on A, Rb and Re represent respectively the start and the end positions of the alignment on R, and C indicates whether R aligns forward (0) or reverse complement (1) to A. In its alignment pile, the read A is called the *template read*. The alignment pile of a given template read A thus contains all the necessary information for its correction, and is retrieved by parsing the Minimap2 output file.

As processing whole alignment piles at once can still be resource consuming, daccord also underlined the interest of dividing alignment piles into windows. A window from an alignment pile is defined as follows. Given an alignment pile for a template read A, a window of this pile is a couple (Wb, We), where Wb and We represent respectively the start and the end positions of the window relatively to A, and such as:

- $0 \leq Wb \leq We < |A|$, *i.e.* the beginning and end positions of the window define a factor of the template read A. We refer to this factor as the *window's template*.
- $We - Wb + 1 = L$, *i.e.* windows have a fixed size.
- $\forall i$, $Wb \leq i \leq We$, $A[i]$ is supported by at least C reads of the pile (including A), *i.e.* windows have a minimum coverage threshold.

In the case of daccord, this window strategy allows to build local de Bruijn graphs for small regions, and thus use small values of k, in order to overcome the high error rates of the long reads, which causes issues when using large values of k. In our case, as we seek to correct long reads by computing multiple alignment of sequences, working with windows allows to save both time and memory, since the sequences that need to be aligned are significantly shorter. Each window is then processed independently during the next steps.

Given a window from an alignment pile for a read A, LoRSCo seeks to compute its consensus in order to correct the window's template. The processing of a window is performed in two distinct steps. First, the window is cleaned in order to remove sequences that deviate significantly from the window's template, in order to avoid the introduction of errors in the correction. Second, the sequences from the cleaned window are aligned using POA, a multiple sequences alignment strategy based on partial order graphs, in order to compute consensus, and correct the window's template. Unlike other methods that compute 1-versus-1 alignments between the read to be corrected and other reads mapping to it, and then build a DAG to compute consensus, this strategy allows LoRSCo to directly build the DAG, during the multiple alignment. Indeed, the DAG is first initialized with the sequence of the window's template, and is then iteratively enriched by aligning the other sequences from the window, until it becomes the final graph. Classically, like in other DAG based methods,

the consensus of the window is then computed by following the highest weight path of the graph. This multiple sequences alignment strategy also benefits from an efficient heuristic, based on k-mers chaining, allowing to decompose the global problem into smaller instances, thus reducing both time and memory consumption.

Once the consensus of a window has been computed, the window's template needs to be replaced by its correction on the template read which is being processed. To this aim, the window's template is first locally aligned to the consensus of the window, with dynamic programming. This allows to retrieve the factor of the consensus which actually represent the corrected window. Indeed, as long reads mainly contain insertions and deletions errors, it is more than likely that the consensus computed from a window actually spans outside of the window's template. In this case, not trimming the consensus on the parts that do not align to the window's template would cause over-correction of the read. The original window's template is then replaced by its correction on the template read.

After processing all the windows of the alignment pile of a given template read A, a few erroneous bases might remain on the corrected long read. This might be the case especially on windows' extremities, as only factors of the windows' templates are sometimes corrected, due to the local alignment between windows' templates and windows' consensuses. Moreover, a few weakly supported k-mers can also be present in the correction of a window's template, despite the consensus computation. This might happen in cases where the coverage depth of the window is particularly low. In both cases, in order to get read of the remaining sequencing errors, these regions are further polished with the use of local de Bruijn graphs. To this aim, LoRSCo searches for sketches of n (usually, $n = 3$) solid k-mers flanking these regions. A de Bruijn graph is then built from the window of the alignment pile starting on the leftmost solid k-mer and ending on the rightmost solid k-mer flanking the region. The de Bruijn graphs are thus local, and a small k can be used, allowing to overcome the issues encountered when using large k values, due to the high error rate of the long reads. For a given region to polish, the associated graph is then traversed in order to find a path between the left and the right solid k-mers, dictating a correction.

3. Results

We compare LoRSCo against state-of-the-art error correction methods Canu, daccord, LoRMA, and MECAT. We voluntarily excluded hybrid error correction tools from the comparison, as we believe it makes more sense to only compare self-correction tools against each other. Results on a 50x simulated *E.coli* dataset are given in Table 1, and results on a 50x simulated *S. cerevisiae* dataset are given in Table 2. These results show that LoRSCo compares well to the state-of-the-art, achieving comparable quality, and higher throughput than most methods, while consuming reasonable amounts of memory. Runtime, however, remains the biggest issue in the current implementation, further optimization of the method is thus required.

4. Conclusion

We introduced LoRSCo, a new method for the self-correction of long reads, combining efficient approaches from the state-of-the-art, such as overlap computation via a mapping

Corrector	Throughput	Error rate	Deletions	Insertions	Substitutions	Runtime	Memory peak (MB)
Original	232,013,329	12.2674	2.6403	8.7841	0.8430	N/A	N/A
Canu	173,134,629	0.5841	0.1563	0.4524	0.0272	19 min 20	3,623
daccord	217,669,319	0.0166	0.0023	0.0061	0.0097	38 min	13,559
LoRMA	125,825,166	9.4315	0.4074	7.3781	2.3844	37 min	31,902
MECAT	192,580,040	0.1118	0.0970	0.0243	0.0010	4 min	2,130
LoRSCo	203,018,368	0.1728	0.0746	0.1060	0.0084	3 h 19 min	3,927

Table 1. Results on the simulated *E. coli* dataset.

Corrector	Throughput	Error rate	Deletions	Insertions	Substitutions	Runtime	Memory peak (MB)
Original	617,531,933	12.2835	2.6459	8.7943	0.8433	N/A	N/A
Canu	477,048,097	0.6294	0.1715	0.4811	0.0330	55 min	3,702
daccord	579,339,282	0.0451	0.0080	0.0203	0.0209	1 h 51 min	31,774
LoRMA	339,182,738	9.6010	0.4124	7.4299	2.5086	2 h 41 min	31,480
MECAT	510,446,777	0.1493	0.1279	0.0334	0.0022	11 min	4,275
LoRSCo	537,232,951	0.3077	0.1502	0.1596	0.0269	10 h 48 min	8,487

Table 2. Results on the simulated *S. cerevisiae* dataset.

strategy, alignment piles, multiple sequences alignment, and local de Bruin graphs. While current results are promising in terms of throughput and quality, the overall runtime of the method remains an issue. Future works should therefore mainly focus on this aspect. In particular, the multiple sequences alignment, which is the bottleneck of the current implementation, could be performed with other algorithms, to achieve lower time consumption. The method should also be tested on real long reads, both from Pacific Biosciences and Oxford Nanopore, to ensure it also performs well on such data.

References

[1] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k -mer weighting and repeat separation. *Genome Research*, 27:722–736, 2017.

[2] Chuan Le Xiao, Ying Chen, Shang Qian Xie, Kai Ning Chen, Yan Wang, Yue Han, Feng Luo, and Zhi Xie. MECAT: Fast mapping, error correction, and de novo assembly for single-molecule sequencing reads. *Nature Methods*, 14(11):1072–1074, 2017.

[3] Chen-Shan Chin, David H Alexander, Patrick Marks, Aaron A Klammer, James Drake, Cheryl Heiner, Alicia Clum, Alex Copeland, John Huddleston, Evan E Eichler, Stephen W Turner, and Jonas Korlach. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature Methods*, 10:563–569, 2013.

[4] German Tischler and Eugene W Myers. Non Hybrid Long Read Consensus Using Local De Bruijn Graph Assembly. *bioRxiv*, 2017.

[5] Leena Salmela, Riku Walve, Eric Rivals, and Esko Ukkonen. Accurate selfcorrection of errors in long reads using de Bruijn graphs. *Bioinformatics*, 33:799–806, 2017.

[6] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[7] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

SeqBio 2018
Rouen, 19-20 novembre

Extended abstract

ELECTOR: EvaLuator of Error Correction Tools for lOng Reads

Camille Marchet[1†*] Pierre Morisse[2†], Lolita Lecompte[3], Antoine Limasset[1], Arnaud Lefebvre[2], Thierry Lecroq[2], Pierre Peterlongo[3]

[1] *Univ. Lille, CNRS, Inria, UMR 9189 - CRIStAL - F-59000, Lille, France.*
[2] *Normandie Univ, UNIROUEN, LITIS, Rouen 76000, France.*
[3] *Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France.*
[†] These authors contributed equally to this work.
[*] **Corresponding author**: camille.marchet@irisa.fr

Abstract

Pacific Biosciences and Oxford Nanopore long reads were rapidly adopted in a broad specter of applications. However, due to their high error rates, error correction is a mandatory step before being able to process these reads efficiently. As a result, various error correction methods were developed for long reads, whether they make use of additional short reads (hybrid correction) or not (self-correction). As the quality of the error correction directly impacts downstream applications, developing methods allowing to evaluate error correction tools is a crucial need. To date, only one tool proposes such an assessment of error correction methods. However, it suffers from scalability issues when assessing correction for large datasets. We present ELECTOR, a new tool that allows the evaluation of long read error correction methods, provides relevant metrics, and overcomes the scalability issues of the previous tool. Our experiments show that ELECTOR is several orders of magnitude faster than the previous tool, while providing comparable results, and also additional metrics. ELECTOR is an open-source software available on GitHub: https://github.com/kamimrcht/ELECTOR.

Keywords

long reads — correction — evaluation

1. Introduction

Pacific Biosciences (PB) and Oxford Nanopore (ONT) long reads, despite their high error rates and complex error profiles, were rapidly adopted for various applications. In particular, they are expected to help solving problems faced with short reads in the genomic assembly field. To overcome these high error rates, a plethora of error correction methods directly targeted at long reads were developed. These methods either aim at correcting the long reads solely based on the information contained on their sequences (self-correction), or use complementary short reads, relying on their important coverage depth and their low error rate (hybrid correction).

As the quality of the error correction has huge impacts on downstream processes, developing methods allowing to evaluate error correction tools with precise and reliable statistics

is therefore a crucial need. However, works introducing new error correction methods usually evaluate the quality of their tools based on how the corrected long reads can be realigned to the reference. Despite being interesting, this information remains incomplete, and is likely not to mention poor quality reads, or regions to which it is difficult to align. In this work we propose ELECTOR, a novel tool that enables the evaluation of long read hybrid and self-correction methods, that provides relevant metrics and that scales to large datasets.

To date, LRCstats [1] was the only method able to realize long read correctors evaluation. LRCstats proposes a three-way alignment strategy that relies on pairwise alignments of both corrected and original versions of each read to the reference. LRCstats provides reads error rate before and after correction, as well as the detailed counts of every type of error. However, only studying the error rate of the reads is not a satisfying indication of the corrector's behaviour, as it does not report information about the putative insertions of new errors by the corrector. LRCstats is well-tailored for experiments with long reads of a few kilobases and for medium throughputs of less than 200 Mb. However we show that it can be more time and/or memory consuming than the actual error correction methods on larger experiments or reads longer than 10kb.

2. Contribution

In order to cope with these limits, we designed ELECTOR to 1/ compute more relevant metrics on long read correction; and 2/ scale to very long reads and large sequencing experiments. ELECTOR is directly compatible with a wide range of state-of-the-art error correction tools, without needing the user to perform any pre-processing. Therefore, it simply takes as input a set of reads, their corresponding corrected versions, and the corresponding reference genome. Contrary to LRCstats, it also includes additional steps performing and assessing corrected reads remapping and assembly, respectively using BWA-MEM [2] and Miniasm [3]. Output statistics include average identity of the alignments and genome coverage for the remapping part, and number of contigs, number of breakpoints, NGA50 and NGA75 for the assembly part. It is also meant to be a user-friendly tool, that delivers its results through different output formats, such as graphics than can be directly integrated to the users' projects.

First of all, the three-way alignment paradigm used in LRCstats is replaced by a multiple alignment of triplets of sequences in ELECTOR. Such an approach allows to efficiently compare the three different versions of each read: the uncorrected version, as provided by the sequencing experiment or by the reads simulator, the corrected version, as provided by the error correction method, and the reference version, that represents a perfect version of the original read, on which no error would have been introduced.

This choice of using a multiple alignment strategy allows ELECTOR to provide a wide range of metrics that assess the actual quality of the correction. In particular, ELECTOR is able to compute the recall, which is the rate of erroneous bases correctly modified (corrected) by the corrector, the precision, which measures the ability of the corrector not to add new erroneous bases, and the overall correct bases rate for each read. In addition to these classical metrics, ELECTOR also displays other results including GC content before and after correction, number of trimmed and/or split corrected reads, and mean missing size in those reads.

Secondly, we propose solutions to tackle scaling issues and thus to offer a faster and more scalable evaluation pipeline, which benefits large genomes processing. In particular, we coupled an implementation of multiple sequence alignment (MSA) using partial order graphs [4] to a seed strategy comparable to MUMmer [5] or Minimap [3]. This so-called seed-MSA strategy allows to divide the multiple sequence alignment problem, known to be time and memory consuming, into smaller instances. In addition to bringing an interesting methodological contribution, this allows to achieve a significant gain in resources footprint.

ELECTOR can be used on simulated data generated from state-of-the-art long reads simulation tools, such as NanoSim [6] or SimLoRD [7], on which introduced errors are precisely known, but also on real data. In the case of simulated data, the reference version of a given read is easily retrieved by parsing the files describing the introduced errors, generated by the simulator. In the case of real data, the reference sequences are retrieved by aligning the uncorrected reads to the reference genome, using Minimap2 [unpublished]. Only the best hit for each read is kept, and used to determine the corresponding reference sequence. In the case a read cannot align to the reference genome and thus cannot produce a reference sequence, this read is simply excluded from the analysis. In both cases, ELECTOR retrieves the reference versions of the reads by itself.

3. Results

Using bacterial and eukaryotic read sets, we validate our approach and demonstrate that 1/ our heuristic for multiple alignment of long reads provides results that are extremely similar to the original partial order graph alignment, while being several orders of magnitude faster, 2/ ELECTOR provides sound metrics in comparison to the state-of-the-art.

In order to validate our speedup strategy for multiple sequence alignment, we simulated two datasets from the E. coli genome, with SimLoRD. The first was composed of reads with a 1kb mean length, a 10% error rate and a coverage of 100X and the second was composed of reads with a 10kb mean length, a 15% error rate and a coverage of 100X. The reads from the two datasets were corrected with MECAT [8] with default parameters. The correction was then assessed both with MSA and seed-MSA strategies. Results of our experiments show that classic MSA and seed-MSA approaches only differ by a few digits in the presented metrics (recall, precision and correct bases rate of corrections). However, using seed-MSA, a substantial gain in time is achieved: while the classic MSA strategy has a subcubic runtime with respect to the read length and the average number of predecessors of nodes in the partial order graph, seed-MSA limits this drawback by working on small instances. As an example, for the second dataset, MSA and seed-MSA compute respectively a recall of 84.505% and 84.587%, a precision of 88.347% and 88.278%, and a correct bases rate of 95.290% and 95.250%, in 107 hours for the classical MSA approach, and in 42 minutes for the seed-MSA approach.

In order to validate the accuracy of ELECTOR's metrics, we then used SimLoRD to simulate three other datasets, respectively from A. bayli, E. coli and S. cerevisiae. Each of these datasets was composed of reads with a 8kb mean length, a 18% error rate, and a

coverage of 20X. We corrected these datasets with various long read correctors, and assessed the correction results using both LRCstats and ELECTOR. Results of these experiments show that the metrics computed by ELECTOR are comparable to LRCstats outputs, but also allow us to highlight several novelties. For instance, on the long reads of the A. baylyi dataset corrected with Nanocorr [9], LRCstats reports an error rate of 0.005777 and ELECTOR reports a correct bases rate of 0.99534, which are in accordance, but ELECTOR only reports a recall of 0.97992, meaning that Nanocorr failed to correct 2% of the erroneous bases. Computation of these results is also more time-saving than LRCstats. In particular, on the E. coli dataset, LRCstats took an average of 3h50min to evaluate the quality of the correction of the different tools, while ELECTOR only took an average of 25 minutes.

Both hybrid and self correctors are included in this benchmark. Although our goal is not to debate on the comparison of correction methods efficiency, this is the first time several self-correctors appear assessed independently of a new error correction tool presentation.

4. Conclusion

We propose a novel and open-source method for fast long read correction assessment. Both hybrid and self correctors are compatible. Our software ELECTOR outputs a wide range of metrics to finely understand the behavior of correction tools, even on large experiments. We compare ELECTOR to a previous work for correctors evaluation and show that it allows a faster and more extensive assessment of long read correction on several species.

Conclusions about pros and cons of hybrid vs self-correction and comparisons of correction paradigms vary a lot according to publications. At the moment, no coherent vision is proposed for long read correction and the field lacks a global study that goes over the sum of individual publications. ELECTOR could thus be a perfectly fitted basis for such benchmark study.

References

[1] Sean La, Ehsan Haghshenas, and Cedric Chauve. LRCstats, a tool for evaluating long reads correction methods. *Bioinformatics*, 33:3652–3654, 2017.

[2] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26:589–595, 2010.

[3] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *arXiv*, 25:1–7, 2015.

[4] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[5] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5:R12, jan 2004.

[6] Chen Yang, Justin Chu, René L. Warren, and Inanç Birol. NanoSim: Nanopore sequence read simulator based on statistical characterization, 2017.

[7] Bianca K. Stöcker, Johannes Köster, and Sven Rahmann. SimLoRD: Simulation of Long Read Data. In *Bioinformatics*, volume 32, pages 2704–2706, 2016.

[8] Chuan Le Xiao, Ying Chen, Shang Qian Xie, Kai Ning Chen, Yan Wang, Yue Han, Feng Luo, and Zhi Xie. MECAT: Fast mapping, error correction, and de novo assembly for single-molecule sequencing reads. *Nature Methods*, 14(11):1072–1074, 2017.

[9] Sara Goodwin, James Gurtowski, Scott Ethe-Sayers, Panchajanya Deshpande, Michael C Schatz, and W Richard Mccombie. Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome Research*, 25:1750–1756, 2015.