Detecting Mutations by eBWT

Nicola Prezza
Dipartimento di Informatica, University of Pisa, Italy
nicola.prezza@di.unipi.it
https://orcid.org/0000-0003-3553-4953

Nadia Pisanti
Dipartimento di Informatica, University of Pisa, Italy, and
ERABLE Team, INRIA, Lyon, France
pisanti@di.unipi.it
https://orcid.org/0000-0003-3915-7665

Marinella Sciortino
Dipartimento di Matematica e Informatica, University of Palermo, Italy
marinella.sciortino@unipa.it
https://orcid.org/0000-0001-6928-0168

Giovanna Rosone[1]
Dipartimento di Informatica, University of Pisa, Italy
giovanna.rosone@unipi.it
https://orcid.org/0000-0001-5075-1214

Abstract

In this paper we develop a theory describing how the extended Burrows-Wheeler Transform (eBWT) of a collection of DNA fragments tends to cluster together the copies of nucleotides sequenced from a genome G. Our theory accurately predicts how many copies of any nucleotide are expected inside each such cluster, and how an elegant and precise LCP array based procedure can locate these clusters in the eBWT.

Our findings are very general and can be applied to a wide range of different problems. In this paper, we consider the case of alignment-free and reference-free SNPs discovery in multiple collections of reads. We note that, in accordance with our theoretical results, SNPs are clustered in the eBWT of the reads collection, and we develop a tool finding SNPs with a simple scan of the eBWT and LCP arrays. Preliminary results show that our method requires much less coverage than state-of-the-art tools while drastically improving precision and sensitivity.

2012 ACM Subject Classification Applied computing → Bioinformatics, Mathematics of computing → Combinatorial algorithms

Keywords and phrases BWT, LCP Array, SNPs, Reference-free, Assembly-free

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.3

Supplement Material `https://github.com/nicolaprezza/eBWTclust`

Funding G. Rosone, N. Pisanti, M. Sciortino are partially and N. Prezza is supported by the project MIUR-SIR CMACBioSeq ("Combinatorial methods for analysis and compression of biological sequences") grant n. RBSI146R5L.

[1] Corresponding author

Acknowledgements We want to thank the anonymous reviewers and Mikaël Salson for their interesting and useful comments.

1 Introduction

The cheap and fast Next Generation Sequencing (NGS) technologies are producing huge amounts of data that put a growing pressure on data structures designed to store raw sequencing information, as well as on efficient analysis algorithms: collections of billion of DNA fragments (reads) need to be efficiently indexed for downstream analysis. After a sequencing experiment, the most traditional analysis pipeline begins with an error-prone and lossy mapping of the collection of reads onto a reference genome. Among the most widespread tools to align reads on a reference genome we can mention BWA [20], Bowtie2 [16], SOAP2 [21]. These methods share the use of the FM-index [10], an indexing machinery based on the Burrows-Wheeler Transform (BWT) [4]. Other approaches [13, 14] combine an index of the reference genome with the BWT of the reads collection in order to boost efficiency and accuracy. In some applications, however, aligning reads on a reference genome presents limitations mainly due to the difficulty of mapping highly repetitive regions, especially in the event of a low-quality reference genome (not to mention the cases in which the reference genome is not even available).

For this reason, indices of reads collections have also been suggested as a lossless dictionary of sequencing data, where sensitive analysis methods can be directly applied without mapping the reads to a reference genome (thus without needing one), nor assembling [29, 31, 17, 12]. In [7] the BWT, or more specifically its extension to string collections (named eBWT [27, 2]), is used to index reads from the 1000 Genomes Project [37] in order to support k-mer search queries. An eBWT-based compressed index of sets of reads has also been suggested as a basis for both RNA-Seq [6] and metagenomic [1] analyses. There exist also suffix array based data structures devised for indexing reads collections: the Gk array [30, 39] and the PgSA [15]. The latter does not have a fixed k-mer size. The tool SHREC [35] also uses a suffix-sorting-based index to detect and correct errors in sets of reads. The main observation behind the tool is that sequencing errors disrupt unary paths at deep levels of the reads' suffix trie. The authors provide a statistical analysis allowing to detect such branching points. Finally, there are several tools ([29, 31, 19, 3, 18, 17, 12]) that share the idea of using the de Bruijn graph (dBG) of the reads' k-mers. The advantages of dBG-based indices include allowing therein the characterization of several biologically-interesting features of the data as suitably shaped and sized *bubbles*[2] (e.g. SNPs, INDELs, Alternative Splicing events on RNA-Seq data, sequencing errors can all be modeled as bubbles in the dBG of sequencing data [29, 31, 19, 3, 18]). The drawback of these dBG representation, as well as those of suffix array based indices such as the ones introduced in [30, 39], is the lossy aspect of getting down to k-mers rather than representing the actual whole collection of reads. Also [13, 14] have this drawback as they index k-mers. An eBWT-based indexing method for reads collections, instead, has the advantages to be easy to compress and, at the same time, lossless: (e)BWT indexes support querying k-mers without the need to build different indexes for different values of k.

Here we introduce the *Positional Clustering* framework: an eBWT-based index of reads collections where we give statistical characterizations of (i) read suffixes prefixing the same genome's suffix as *clusters* in the eBWT, and (ii) the onset of these clusters by means of the

[2] A *bubble* in a graph is a pair of disjoint paths sharing the same source node and target node.

LCP. This clustering allows to locate and investigate, in a lossless index of reads collections, genome positions possibly equivalent to bubbles in the dBG [19, 29] *independently* from the k-mer length (a major drawback of dBG-based strategies). We thus gain the advantages of dBG-based indices while maintaining those of (e)BWT-based ones. Besides, the eBWT index also contains abundance data (useful to distinguish errors from variants, as well as distinct variant types) and does not need the demanding read-coherency check at post processing as no micro-assembly has been performed. With these promising advantages with respect to dBG-based strategies, the positional clustering framework allows likewise reference-free and assembly-free detection of SNPs [29, 28, 12], small INDELs [22, 12], sequencing errors [32, 33, 23], alternative splicing events [31], rearrangements breakpoints [19] on raw reads collections. To our knowledge, SHREC [35] and our positional clustering framework are the only attempts to characterize the statistical behavior of suffix trees of reads sets in presence of errors. We note that, while the two solutions are completely different from the algorithmic and statistical points of view, they are also, in some sense, complementary. SHREC characterizes errors as branching points at deep levels of the suffix trie; on the contrary, our positional framework characterizes clusters of read suffixes prefixing the same genome's suffix and identifies mutations (e.g. sequencing errors or SNPs) in the characters *preceding* those suffixes (i.e. eBWT characters). We note that our cluster characterization could be used to detect the suffix trie level from where sequencing errors are detected in SHREC. Similarly, SHREC's characterization of errors as branching points could be used in our framework to detect further mutations in addition to those in the eBWT clusters.

As a proof-of-concept, we test our theoretical framework with a prototype tool named eBWTclust designed to detect positional clusters and post-process them for assembly-free and reference-free SNPs detection directly on the eBWT of reads collection. Among several reference-free SNPs finding tools in the literature [29, 38, 28, 12], the state-of-the-art is represented by the well documented and maintained KisSNP and DiscoSnp suite [29, 38, 11], where DiscoSnp++ [28] is the latest and best performing tool. In order to validate the accuracy of positional clustering for finding SNPs, we compared DiscoSnp++ sensitivity and precision to those of our prototype eBWTclust. Preliminary results on human chromosomes show that, for example, using coverage 22x our tool is able to find 91% of all SNPs (vs 70% of DiscoSnp++) with an accuracy of 98% (vs 94% of DiscoSnp++).

2 Preliminaries

Let $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$ be a finite ordered alphabet with $c_1 < c_2 < \ldots < c_\sigma$, where $<$ denotes the standard lexicographic order. For $s \in \Sigma^*$, we denote its letters by $s[1], s[2], \ldots, s[n]$, where n is the *length* of s, denoted by $|s|$. We append to $s \in \Sigma^*$ an end-marker symbol \$ that satisfies $\$ < c_1$. Note that, for $1 \leq i \leq n$, $s[i] \in \Sigma$ and $s[n+1] = \$ \notin \Sigma$. A *substring* of s is denoted as $s[i, j] = s[i] \cdots s[j]$, with $s[1, j]$ being called a *prefix* and $s[i, n+1]$ a *suffix* of s.

We denote by $\mathcal{S} = \{R_1, R_2, \ldots, R_m\}$ a collection of m strings (reads), and by $\$_i$ the end-marker appended to R_i (for $1 \leq i \leq m$), with $\$_i < \$_j$ if $i < j$. Let us denote by P the sum of the lengths of all strings in $\mathcal{S}$. The *generalized suffix array* GSA of the collection $\mathcal{S}$ (see [36, 5, 25]) is an array containing P pairs of integers (r, j), corresponding to the lexicographically sorted suffixes $R_r[j, |R_r| + 1]$, where $1 \leq j \leq |R_r| + 1$ and $1 \leq r \leq m$. In particular, $\mathsf{gsa}(\mathcal{S})[i] = (r, j)$ (for $1 \leq i \leq P$) if the suffix $R_r[j, |R_r| + 1]$ is the i-th smallest suffix of the strings in $\mathcal{S}$. Such a notion is a natural extension of the suffix array of a string (see [26]). The Burrows-Wheeler Transform (BWT) [4], a well known text transformation largely used for data compression and self-indexing compressed data structure, has also been extended to a collection $\mathcal{S}$ of strings (see [27]). Such an extension, known as *extended*

Burrows-Wheeler Transform (eBWT) or multi-string BWT, is a reversible transformation that produces a string that is a permutation of the letters of all strings in $\mathcal{S}$: $\mathsf{ebwt}(\mathcal{S})$ is obtained by concatenating the symbols cyclically preceding each suffix in the list of lexicographically sorted suffixes of all strings in $\mathcal{S}$. The eBWT applied to $\mathcal{S}$ can also be defined in terms of the generalized suffix array of $\mathcal{S}$ ([2]): if $\mathsf{gsa}(\mathcal{S})[i] = (j, t)$ then $\mathsf{ebwt}(\mathcal{S})[i] = R_j[t-1]$; when $t = 1$, then $\mathsf{ebwt}(\mathcal{S})[i] = \$_j$. For gsa, ebwt, and lcp, the LF mapping (resp. FL) is a function that associates to each (e)BWT symbol the position preceding (resp. following) it on the text.

The *longest common prefix* (LCP) array of a collection $\mathcal{S}$ of strings (see [5, 25, 8]), denoted by $\mathsf{lcp}(\mathcal{S})$, is an array storing the length of the longest common prefixes between two consecutive suffixes of $\mathcal{S}$ in lexicographic order. For each $i = 2, \ldots, P$, if $\mathsf{gsa}(\mathcal{S})[i-1] = (p_1, p_2)$ and $\mathsf{gsa}(\mathcal{S})[i] = (q_1, q_2)$, $\mathsf{lcp}(\mathcal{S})[i]$ is the length of the longest common prefix of suffixes starting at positions p_2 and q_2 of the strings R_{p_1} and R_{q_1}, respectively. We set $\mathsf{lcp}(\mathcal{S})[1] = 0$.

For gsa, ebwt, and lcp, the set $\mathcal{S}$ will be omitted when clear from the context.

3 eBWT positional clustering

Let R be a read sequenced from a genome $G[1, n]$. We say that $R[j]$ is a *read-copy* of $G[i]$ iff $R[j]$ is copied from $G[i]$ during the sequencing process (and then possibly changed due to sequencing errors). Let us consider the eBWT of a set of reads $\{R_1, \ldots, R_m\}$ of length[3] r, sequenced from a genome G. Assuming that c is the *coverage* of $G[i]$, let us denote with $R_{i_1}[j_1], \ldots, R_{i_c}[j_c]$ the c read-copies of $G[i]$. Should not there be any sequencing error, if we consider k such that the genome fragment $G[i+1, i+k]$ occurs only once in G (that is, nowhere else than right after $G[i]$) and if r is large enough so that with high probability each $R_{i_t}[j_t]$ is followed by at least k nucleotides, then we observe that the c read copies of $G[i]$ would appear contiguously in the eBWT of the reads. We call this phenomenon eBWT *positional clustering*. Due to sequencing errors, and to the presence of repetitions with mutations in real genomes, a *clean* eBWT positional clustering is not realistic. However, in this section we show that, even in the event of sequencing errors, in the eBWT of a collection of reads sequenced from a genome G, the read-copies of $G[i]$ still *tend to be clustered* together according to a suitable Poisson distribution.

We make the following assumptions: (i) the sequencing process is uniform, i.e. the positions from where each read is sequenced are uniform and independent random variables, (ii) the probability ϵ that a base is subject to a sequencing error is a constant[4], (iii) a sequencing error changes a base to a different one uniformly (i.e. with probability 1/3 for each of the three possible variants), and (iv) the number m of reads is large (hence, in our theoretical analysis we can assume $m \to \infty$).

▶ **Definition 1** (eBWT cluster)**.** The eBWT *cluster of* i, with $1 \leq i \leq n$ being a position on G, is the substring $\mathsf{ebwt}[a, b]$ such that $\mathsf{gsa}[a, b]$ is the range of read suffixes prefixed by $G[i+1, i+k]$, where $k < r$ is the smallest value for which $G[i+1, i+k]$ appears only once in G. If no such value of k exists, we take $k = r - 1$ and say that the cluster is *ambiguous*.

If no value $k < r$ guarantees that $G[i+1, i+k]$ appears only once in G, then the eBWT cluster of i does not contain only read-copies of $G[i]$ but also those of other $t-1$ characters $G[i_2], \ldots, G[i_t]$. We call t the *multiplicity* of the eBWT cluster. Note that $t = 1$ for non-ambiguous clusters.

[3] For simplicity of exposition, here we assume that all the reads have the same length r. With little more effort, it can be shown that our results hold also when r is the *average* read length.

[4] We assume this to simplify the theoretical framework. In Section 4 we will see that our framework works even on data with sequencing simulated using realistic position-dependent errors distribution.

▶ **Theorem 2** (eBWT positional clustering). *Let $R_{i_1}[j_1], \ldots, R_{i_c}[j_c]$ be the c read-copies of $G[i]$. An expected number $X \leq c$ of these read copies will appear in the* eBWT *cluster* $\mathsf{ebwt}[a, b]$ *of i, where $X \sim Poi(\lambda)$ is a Poisson random variable with mean*

$$\lambda = m \cdot \frac{r-k}{n} (1-\epsilon)^k$$

and where k is defined as in Definition 1.

Proof. The probability that a read covers $G[i]$ is r/n. However, we are interested only in those reads such that, if $R[j]$ is a read-copy of $G[i]$, then the suffix $R[j+1, r+1]$ contains at least k nucleotides, i.e. $j \leq r-k$. In this way, the suffix $R[j+1, r+1]$ will appear in the GSA range $\mathsf{gsa}[a, b]$ of suffixes prefixed by $G[i+1, i+k]$ or, equivalently, $R[j]$ will appear in $\mathsf{ebwt}[a, b]$. The probability that a random read from the set is uniformly sampled from such a position is $(r-k)/n$. If the read contains a sequencing error inside $R[j+1, j+k]$, however, the suffix $R[j+1, r+1]$ will not appear in the GSA range $\mathsf{gsa}[a, b]$. The probability that this event does not happen is $(1-\epsilon)^k$. Since we assume that these events are independent, the probability of their intersection is therefore

$$Pr(R[j] \in \mathsf{ebwt}[a, b]) = \frac{r-k}{n} (1-\epsilon)^k$$

This is a Bernoullian event, and the number X of read-copies of $G[i]$ falling in $\mathsf{ebwt}[a, b]$ is the sum of m independent events of this kind. Then, X follows a Poisson distribution with mean $\lambda = m \cdot \frac{r-k}{n} (1-\epsilon)^k$. ◀

Theorem 2 states that, if there exists a value $k < r$ such that $G[i+1, i+k]$ appears only once in G (i.e. if the cluster of i is not ambiguous), then X of the $b-a+1$ letters in $\mathsf{ebwt}[a, b]$ are read-copies of $G[i]$. The remaining $(b-a+1)-X$ letters are noise introduced by suffixes that mistakenly end up inside $\mathsf{gsa}[a, b]$ due to sequencing errors. It is not hard to show that this noise is extremely small under the assumption that G is a uniform text; we are aware that this assumption is not realistic, but we will experimentally show in Section 4 that also on real genomes and simulated reads our approach produces extremely accurate results (as predicted by our simplified theoretical framework).

Note that the expected coverage of position $G[i]$ is also a Poisson random variable, with mean $\lambda' = \frac{mr}{n}$ equal to the average coverage. On expectation, the size of non-ambiguous eBWT clusters is thus $\lambda/\lambda' = \frac{(r-k)(1-\epsilon)^k}{r} < 1$ times the average coverage. E.g., with $k = 14$, $\epsilon = 0.0033$ (see [34, Table 1, HiSeq, R2]), and $r = 100$ the expected cluster size is $100 \cdot \lambda/\lambda' \approx 80\%$ the average coverage.

Finally, it is not hard to prove, following the proof of Theorem 2, that in the general case with multiplicity $t \geq 1$ the expected cluster size follows a Poisson distribution with mean $t \cdot \lambda$ (because the read-copies of t positions are clustered together). This observation will allow us to detect and discard ambiguous clusters using a significance test.

So far, we have demonstrated the eBWT positional clustering property but we don't have a way for identifying the eBWT clusters. A naive strategy could be to fix a value of k and define clusters to be ranges of k-mers in the GSA. This solution, however, fails to separate read suffixes differing after k positions (this is, indeed, a drawback of all k-mer-based strategies). The aim of Theorem 3 is precisely to fill this gap, allowing us to move from theory to practice. Intuitively, we show that clusters lie between local minima in the LCP array. This strategy automatically detects the value k satisfying Definition 1 in a data-driven approach.

Our result holds only if two conditions on the eBWT cluster under investigation are satisfied (see Proposition 4 for an analysis of the success probability). More specifically, we require that the eBWT cluster $\mathsf{ebwt}[a,b]$ of position i satisfies:

(1) The cluster does not have noise, i.e. $X = b - a + 1$, and

(2) Let $(p_1, j_1), (p_2, j_2) \in \mathsf{gsa}[a,b]$. For any x such that $k \leq x < r$, if both $R_{p_1}[j_1, r]$ and $R_{p_2}[j_2, r]$ contain their leftmost sequencing errors in $R_{p_1}[j_1 + x]$ and $R_{p_2}[j_2 + x]$, then $R_{p_1}[j_1 + x] \neq R_{p_2}[j_2 + x]$.

Proposition 4 will show that with probability high enough Condition (2) is satisfied in practice. E.g., with $r = 100$, $\epsilon = 0.0033$ (see [34, Table 1, HiSeq, R2]), mean coverage $\lambda' = 20$, and for any $k \geq 14$, Proposition 4 shows that the condition holds (in expectation) on at least 85% of the non-ambiguous clusters. Decreasing ϵ to 0.002 [34, Table 1, HiSeq, R1] this percentage increases to 93%. While the rough lower bound of Proposition 4 is quite sensitive with respect to the substitution rate ϵ, the sensitivity of $\approx 90\%$ we obtain in our experiments (see Section 4.4) suggests that on our test-cases at least 90% of the clusters satisfy Condition (2).

▶ **Theorem 3.** *Let* $\mathsf{ebwt}[a,b]$ *be the* eBWT *cluster of a position* i *meeting Conditions (1) and (2). Then, there exists a value* $a < p \leq b$ *such that* $\mathsf{lcp}[a+1, p]$ *is a non-decreasing sequence and* $\mathsf{lcp}[p+1, b]$ *is a non-increasing sequence.*

Proof. Let us denote by p_M the largest index in $(a,b]$ such that $\mathsf{lcp}[p_M] = M$, where M is the maximum value of LCP in $(a,b]$ (if M occurs multiple times, take the rightmost occurrence). We claim the theorem holds for $p = p_M$. Let us denote by q and j, $1 \leq q \leq m$, $1 \leq j \leq r$, the positive integers such that $\mathsf{gsa}[p_M] = (q, j)$. This means, by using Condition (2), that the read R_q contains the longest prefix $R_q[j, j + M - 1]$ without sequencing errors, $j + M \leq r + 1$. Consider any other suffix $R_u[j_u, r+1]$ in the range and let $R_u[j_u + x]$ be the leftmost mismatch letter in $R_u[j_u, r + 1]$, with $x \geq k$. Note that if $R_u[j_u, r + 1]$ does not contain sequencing errors, then $j_u + x - 1 = r$, otherwise $R_u[j_u + x]$ has been mutated with an error. We suppose that the mutation at position $j_u + x$ generated a letter lexicographically smaller than that of the genome (the other case is symmetric). By Condition (2), $x \leq M + 1$ and no other suffix $R_v[j_v, r + 1]$ in the range satisfies $R_v[j_v + x] = R_u[j_u + x]$. Then, $R_u[j_u, r + 1]$ falls right after a suffix $R_{u'}[j_{u'}, r + 1]$ such that either $R_{u'}[j_{u'}, r]$ properly prefixes $R_u[j_u, r + 1]$ or the leftmost mismatch occurs at position $x' \leq x$. Similarly, $R_u[j_u, r + 1]$ falls right before a suffix $R_{u''}[j_{u''}, r + 1]$ such that $R_u[j_u + x] < R_{u''}[j_{u''} + x]$ and whose leftmost mismatch position x'' satisfies $x \leq x'' \leq M + 1$. This shows that, before suffix $R_q[j, r + 1]$, other suffixes are ordered by increasing position of their leftmost mismatch letter (since $x' \leq x \leq x''$) and, then, by the lexicographic order among mismatch letters, which in particular implies that before suffix $R_q[j, r + 1]$ the lcp values are non-decreasing. Symmetrically, with a similar reasoning one can easily prove that after suffix $R_q[j, r + 1]$ the lcp values are non-increasing. ◀

▶ **Proposition 4.** *Given a* eBWT *cluster* $\mathsf{ebwt}[a,b]$ *with multiplicity* $t \geq 1$*, Condition (2) holds with probability at least*

$$CDF(t\lambda, \lceil t\lambda \rceil + \delta) \cdot \left(\sum_{e=0}^{3} \binom{\lceil t\lambda \rceil + \delta}{e} \cdot (1-\epsilon)^{\lceil t\lambda \rceil + \delta - e} \cdot \epsilon^e \cdot c_e \right)^{r-k}$$

for any integer $\delta \geq 0$ *and where:* k *is the value defined in Definition 1,* λ *is the mean of the Poisson distribution of Theorem 2,* $CDF(\mu, z)$ *is the cumulative distribution function of a Poisson random variable with mean* μ *evaluated in* z*, and* $c_0 = c_1 = 1$*,* $c_2 = 2/3$*,* $c_3 = 2/9$*.*

Proof. First note that, by Theorem 2, the number of suffixes in the cluster sharing their first k bases is at most $\lceil t\lambda \rceil + \delta$ with probability $CDF(t\lambda, \lceil t\lambda \rceil + \delta)$. We first analyze the probability that the condition holds for a fixed offset x (i.e. looking at the $(x+1)$-th base of all suffixes in the range sharing their first x characters), and then compute the joint probability for all $k \leq x < r$.

Note that the condition cannot fail if the number e of bases $R_{p_h}[j_h + x]$ that have been subject to errors is either $e = 0$ or $e = 1$. The condition does not fail also if we have $e = 2$ or $e = 3$ *distinct* errors, while it always fails for $e \geq 4$ (since at least two errors will produce the same base). On a generic number $Y \leq \lceil t\lambda \rceil + \delta$ of suffixes (those sharing their first x bases), one can verify that the probability that the condition does not fail for a fixed number $e < 4$ of errors is $\binom{Y}{e} \cdot (1-\epsilon)^{Y-e} \cdot \epsilon^e \cdot c_e$, where $c_0 = c_1 = 1$, $c_2 = 2/3$, $c_3 = 2/9$. Since these events are disjoint, we can sum these probabilities to get a lower bound to the probability that condition (2) holds on a specific offset x on Y suffixes. Since this probability decreases as Y increases, we get a lower bound by taking the maximum $Y = \lceil t\lambda \rceil + \delta$, and obtain probability $\sum_{e=0}^{3} \binom{\lceil t\lambda \rceil + \delta}{e} \cdot (1-\epsilon)^{\lceil t\lambda \rceil + \delta - e} \cdot \epsilon^e \cdot c_e$.

To get a lower bound to the probability that the condition holds *simultaneously* on all $k \leq x < r$, we take the product of these probabilities (note that errors at different offsets are independent events). This reasoning holds only if there are at most $\lceil t\lambda \rceil + \delta$ suffixes sharing their first k bases – which happens with probability $CDF(t\lambda, \lceil t\lambda \rceil + \delta)$ – so we must include this multiplicative correction factor to get our final lower bound. ◀

Note: to get a lower bound that is independent from δ in Proposition 4, use the δ that maximizes the expression (since the lower bound holds for any δ).

According to Theorem 3, clusters are delimited by local minima in the LCP array of the read set. This gives us a strategy for finding clusters that is *independent* from k. Importantly, the proof of Theorem 3 also gives us the suffix in the range (the p-th suffix) whose longest prefix without sequencing errors is maximized. This will be useful in the next section to efficiently compute a consensus of the reads in the cluster.

Observe that by applying Theorem 3 we also find ambiguous clusters. However, the expected length of these clusters is a multiple of λ, so they can be reliably discarded with a significance test based on the Poisson distribution of Theorem 2.

To conclude, we note that our analysis automatically adapts to the case where also indels are present in the reads (i.e. not just substitutions, but possibly also insertions and deletions). To see why this holds true, note that our analysis only looks at the *first* base that changes w.r.t. the reference in any read suffix. Since we do not look at the following bases, indels behave exactly like SNPs in Theorems 2 and 3. For the same reason, indels between the two individuals will produce clusters containing two distinct letters (i.e. we capture the last letter of the indel, which by definition differs from the corresponding letter in the reference). On the other hand, this shows that we will mistakenly classify indels as SNPs. The straightforward solution (i.e. performing a multiple alignment on the left-contexts in each cluster) will be discussed in a forthcoming extension of this paper.

4 Experimental Validation: Reference-Free SNPs Discovery

When the reads dataset contains variations (e.g. two allele of the same individual, or two or more distinct individuals, or different isoforms of the same gene in RNA-Seq data, or different reads covering the same genome fragment in a sequencing process, etc.), the eBWT positional clustering described in the previous section can be used to detect, directly from the raw reads (hence, without assembly and without the need of a reference genome), positions

$G[i]$ exhibiting possibly different values, but followed by the same context: they will be in a cluster delimited by LCP minima and containing possibly different letters (corresponding to the read copies of the variants of $G[i]$ in the read set). This general idea can be used in several applications: error correction, assembly (the strategy finds overlaps between reads, so it can be used for assembly purposes), haplotype discovery (if fed with reads from just one diploid sample, this strategy can find heterozygous sites), and so on.

In this section, with the purpose of experimentally validating the theoretical framework of Section 3, we describe a new alignment-free and reference-free method that, with a simple scan of the eBWT and LCP arrays, detects SNPs in read collections.

Since (averagely) half of the reads comes from the forward (F) strand, and half from the reverse-complement (RC) strand, we denote with the term *right* (resp. *left*) *breakpoint* those variants found in a cluster formed by reads coming from the F (resp. RC) strand, and therefore sharing the right (resp. left) context adjacent to the variant. A *non-isolated SNP* [38] is a variant at position i such that the closest variant is within k bases from i, for some fixed k (we use $k = 31$ in our validation procedure, see below). The SNP is *isolated* otherwise. Note that, while isolated SNPs are found twice with our method (one as a right breakpoint and one as a left breakpoint), this is not true for non-isolated SNPs: variants at the sides of a group of non-isolated SNPs are found as either left or right breakpoint, while SNPs *inside* the group will be found with positional clustering plus a partial local assembly of the reads in the cluster. In the next two subsections we give all the details of our strategy.

4.1 Pre-processing (eBWT computation)

Since we do not aim at finding matches between corresponding pairs of clusters on the forward and reverse strands, we augment the input adding the reverse-complement of the reads: for a reads set $\mathcal{S}$, we add $\mathcal{S}^{RC}$ as well. Hence, given two reads sets $\mathcal{S}$ and $\mathcal{T}$, in the pre-processing phase we compute $\mathsf{ebwt}(\mathcal{R})$, $\mathsf{lcp}(\mathcal{R})$, and $\mathsf{gsa}(\mathcal{R})$, for $\mathcal{R} = \{\mathcal{S} \cup \mathcal{S}^{RC} \cup \mathcal{T} \cup \mathcal{T}^{RC}\}$. This task can be achieved using, for example, BCR[5] [5], Egsa[6] [25] or gsacak[7] [24]. We also compute $\mathsf{gsa}(\mathcal{R})$ because we will need it (see Subsection 4.2) to extract left and right contexts of the SNP. Though this could be achieved by performing (in external memory) multiple steps of LF- and FL-mappings on the eBWT, this would significantly slow-down our tool. Note that our approach can also be generalized to more than two reads collections.

4.2 SNP calling

Our SNPs calling approach takes as input $\mathsf{ebwt}(\mathcal{R})$, $\mathsf{lcp}(\mathcal{R})$, and $\mathsf{gsa}(\mathcal{R})$ and outputs SNPs in KisSNP2 format [11]: a fasta file containing a pair of sequences per SNP (one per sample, containing the SNP and its context). The SNP calling is divided in two main steps.

Build clusters. First, we scan $\mathsf{ebwt}(\mathcal{R})$ and $\mathsf{lcp}(\mathcal{R})$, find clusters using Theorem 3, and store them to file as a sequence of ranges on the eBWT. In addition, while computing clusters we also apply a threshold of minimum LCP (by default, 16): we cut clusters' tails containing LCP values smaller than the threshold. This additional filtering drastically reduces the number of clusters saved to file (and hence memory usage and running time), since the original strategy would otherwise output many short clusters containing small LCP values corresponding to noise.

[5] `https://github.com/giovannarosone/BCR_LCP_GSA`
[6] `https://github.com/felipelouza/egsa`
[7] `https://github.com/felipelouza/sacak-lcp`

Call SNPs. The second step takes as input the clusters file, $\mathsf{ebwt}(\mathcal{R})$, $\mathsf{lcp}(\mathcal{R})$, $\mathsf{gsa}(\mathcal{R})$, and $\mathcal{R}$, and processes clusters from first to last as follows:

1. We test the cluster's length using the Poisson distribution predicted by Theorem 2; if the cluster's length falls in one of the two tails at the sides of the distribution (by default, the two tails summing up to 5% of the distribution), then the cluster is discarded; Moreover, due to k-mers that are not present in the genome but appear in the reads because of sequencing errors (which introduce noise around cluster length equal to 1), we also fix a minimum value of length for the clusters (by default, 4 letters per sample).
2. In the remaining clusters, we find the most frequent nucleotides b_1 and b_2 of samples 1 and 2, respectively, and check whether $b_1 \neq b_2$; if so, then we have a candidate SNP: for each sample, we use the GSA to retrieve the coordinate of the read containing the longest right-context without errors (see explanation after Proposition 4); moreover, we retrieve, and temporarily store in a buffer, the coordinates of the remaining reads in the cluster.
3. After processing all events, we scan the fasta file storing $\mathcal{R}$ to retrieve the reads of interest (those whose coordinates are in the buffer); for each one of them, we compute a partial assembly of the read prefixes preceding the SNP, for each of the two samples. This allows us to compute a left-context for each SNP (by default, of length 20), and it also represents a further validation step: if the assembly cannot be built because a consensus cannot be found, then the cluster is discarded. Note that these left-contexts preceding SNPs (which are actually right-contexts if the cluster is formed by reads from the RC strand) allow us to capture non-isolated SNPs.

Complexity. In the clustering step, we process the eBWT and LCP and on-the-fly output clusters to disk. The SNP-calling step performs one scan of the eBWT, GSA, and clusters file to detect interesting clusters, plus one additional scan of the read set to retrieve contexts surrounding SNPs. Both these phases take linear time in the size of the input and do not use disk space in addition to the input and output. Due to the fact that we store in a buffer the coordinates of reads inside interesting clusters, this step uses an amount of RAM proportional to the number of SNPs times the average cluster size λ times the read length r (e.g. a few hundred MB in our case study of Section 4.4). Notice that our method is very easy to parallelize, as the analysis of each cluster is independent from the others.

4.3 Validation

Here we describe the validation tool we designed to measure the sensitivity and precision of any tool outputting SNPs in KisSNP2 format. Note that we output SNPs as pairs of reads containing the actual SNPs plus their contexts (one sequence per sample). This can be formalized as follows: the output is a series of pairs of triples (we call them *calls*) (L', s', R'), (L'', s'', R'') where L', R', L'', R'' are the left/right contexts of the SNP in the two samples, and letters s', s'' are the actual variant. Given a .vcf file (Variant Call Format) containing the ground truth, the most precise way to validate this kind of output is to check that the triples actually match contexts surrounding true SNPs on the reference genome (used here just for accuracy validation purposes). That is, for each pair in the output calls:

1. If there is a SNP $s' \to s''$ in the .vcf that is surrounded in the first sample by contexts L', R' (or their RC), then (L', s', R'), (L'', s'', R'') is a true positive (TP).
2. Any pair (L', s', R'), (L'', s'', R'') that is not matched with any SNP in the ground truth (as described above) is a false positive (FP).
3. Any SNP in the ground truth that is not matched with any call is a false negative (FN).

We implemented the above validation strategy with a (quite standard) reduction of the problem to the 2D range reporting problem: we insert in a two-dimensional grid two points per SNP (from the .vcf) using as coordinates the ranks of its right and (reversed) left contexts among the sorted right and (reversed) left contexts of all SNPs (contexts from the first sample) on the F and RC strands. Given a pair (L', s', R'), (L'', s'', R''), we find the two-dimensional range corresponding to all SNPs in the ground truth whose right and (reversed) left contexts are prefixed by R' and (the reversed) L', respectively. If there is at least one point in the range matching the variation $s' \to s''$, then the call is a TP[8] (case 1 above; note: to be a TP, a SNP can be found either on the F or on the RC strand, or both); else, it is a FP (case 2 above). Finally, pairs of points (same SNP on the F/RC strands) that have not been found by any call are marked as FN (case 3 above). We repeat the procedure for any other SNP found between the two strings $L's'R'$ and $L''s''R''$ to find non-isolated SNPs.

4.4 Preliminary Experiments

In order to evaluate our method, we compare eBWTclust with DiscoSnp++, that is a revisiting of the DiscoSnp algorithm: while DiscoSnp detects (both heterozygous and homozygous) *isolated* SNPs from any number of read datasets without a reference genome, DiscoSnp++ detects and ranks all kinds of SNPs as well as small indels. As shown in [28], DiscoSnp++ performs better than state-of-the-art methods in terms of both computational resources and quality of the results.

DiscoSnp++ is composed of several independent tools. As a preprocessing step, the dBG of the input datasets is built, by also removing erroneous k-mers. Then, DiscoSnp++ detects bubbles generated by the presence of SNPs (isolated or not) and indels, and it outputs a fasta file containing the variant sequences (KisSnp2 module). A final step (kissreads2) maps back the reads from all input read sets on the variant sequences, mainly in order to determine the read coverage per allele and per read set of each variant. This module also computes a rank per variant, indicating whether it exhibits discriminant allele frequencies in the datasets. The last module generates a .vcf of the predicted variants. If no reference genome is provided this step is a change of format from fasta to .vcf (VCFcreator module).

We propose two experiments simulating two human chromosomes haploid read sets obtained mutating (with real .vcf files) real reference chromosomes[9]. The final goal of the experiments is to reconstruct the variations contained in the original (ground truth) .vcf files. We generated the mutated chromosomes using the 1000 genome project (phase 3) .vcf files[10] related to chromosomes 16 and 22, suitably filtered to keep only SNPs of individuals HG00100 (ch.16) and HG00096 (ch.22). From these files, we simulated Illumina sequencing with SimSeq [9], both for reference and mutated chromosomes: individual HG00096 (ch.22) at a 29x getting $15,000,000$ of 100-bp reads, and individual HG00100 (ch.16) a 22x getting $20,000,000$ of 100-bp reads. To simulate the reads, we used the HiSeq error profile[11] publicly available in the SimSeq's repository. Note that our experiments are easily reproducible given the links of the datasets, simulator, and error profile we have provided.

---

[8] Since other tools such as DiscoSnp++ do not preserve the order of samples in the output, we actually check also the variant $s'' \to s'$ and also search the range corresponding to L'' and R''

[9] `ftp.1000genomes.ebi.ac.uk//vol1/ftp/technical/reference/phase2_reference_assembly_sequence/hs37d5.fa.gz`

[10] `ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/`

[11] `https://github.com/jstjohn/SimSeq/blob/master/examples/hiseq_mito_default_bwa_mapping_mq10_1.txt`

Table 1 Pre-processing comparative results of eBWTclust (i.e. building the eBWT using either Egsa or BCR) and DiscoSnp++ (i.e. building the de Bruijn graph). Wall clock is the elapsed time from start to completion of the instance, while RAM is the peak Resident Set Size (RSS). Both values were taken with `/usr/bin/time` command.

Dataset	Coverage per Sample	#reads	Preprocessing	Wall Clock (h:mm:ss)	RAM (MB)
HG00096 (ch. 22)	29x	15,000,000	gsacak	0:53:34	100607
			Egsa	1:41:37	30720
			BCR	4:18:00	1970
			DiscoSnp++	00:01:09	5170
HG00100 (ch. 16)	22x	20,000,000	gsacak	1:13:04	112641
			Egsa	3:39:04	30720
			BCR	6:10:28	3262
			DiscoSnp++	00:02:01	6111

Our framework has been implemented in C++ and is available at `https://github.com/nicolaprezza/eBWTclust`. All tests were done on a DELL PowerEdge R630 machine, used in non exclusive mode. Our platform is a 24-core machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40 GHz, with 128 GB of shared memory. The system is Ubuntu 14.04.2 LTS. Note that, unlike DiscoSnp++, our tool is currently able to use one core only.

We experimentally observed that the pre-processing step (see Table 1) is more computationally expensive than the actual SNP calling step. The problem of computing the eBWT is being intensively studied, and improving its efficiency is out of the aim of this paper. However, a recent work [7] suggests that direct storing of read data with a compressed eBWT leads to considerable space savings, and could therefore become the standard in the future. Our strategy can be easily adapted to directly take as input these compressed formats. Building the dBG requires a few minutes and, in order to keep the RAM usage very low, no other information other than k-mer presence is stored in the dBG used by DiscoSnp++. On the other hand, the construction of the eBWT, LCP and GSA arrays can take a few hours. So, overall DiscoSnp++ is faster than eBWTclust when considering the whole processing.

We run DiscoSnp++ with default parameters (includes k-mers size 31) except for $P = 3$ (it searches up to P SNPs per bubble) and b ($b = 0$ forbids variants for which any of the two paths is branching; $b = 2$ imposes no limitation on branching; $b = 1$ is inbetween).

eBWTclust takes as input few main parameters, among which the most important are the lengths of right and left contexts surrounding SNPs in the output (`-L` and `-R`), the minimum cluster size (`-m`), and (`-v`) the maximum number of non-isolated SNPs to seek in the left contexts (like parameter P of DiscoSnp++). In order to make a fair comparison between DiscoSnp++ and eBWTclust, with eBWTclust we decided to output (exactly as for DiscoSnp++) 30 nucleotides following the SNP (`-R 30`), 31 nucleotides preceding and including the SNP (`-L 31`) (i.e. the output reads are of length 61, with the SNP in the middle position), and `-v 3` (as we used $P = 3$ with DiscoSnp++). For the minimum cluster size, we tested both combinations `-m 6` and `-m 4`. We also show experiments with `-L 20` in order to study how extracting shorter left-contexts impacts running times and precision.

In Table 2, we show the number of TP, FP and FN as well as sensitivity (SEN), precision (PREC), and the number of non-isolated SNPs found by the tools. The outcome is that eBWTclust is always more precise and sensitive than DiscoSnp++. Moreover, while in our case precision is stable and always quite high (always between 94% – 99%), for

Table 2 Post-processing comparative results of eBWTclust (i.e. building clusters from the eBWT and performing SNP calling) and DiscoSnp++ (i.e. running KisSnp2 and kissreads2 using the pre-computed de Bruijn graph). Wall clock (mm:ss) is the elapsed time from start to completion of the instance, while RAM is the peak Resident Set Size (RSS). Both values were taken with `/usr/bin/time` command.

Individual HG00096 vs reference (chromosome 22, 50818468bp), coverage 29× per sample

Tool	Param.	Wall Clock	RAM (MB)	TP	FP	FN	SEN (%)	PREC (%)	Non-isol. SNP
DiscoSnp++	b=0	5:07	101	32773	3719	13274	71.17	89.81	4707/8658
	b=1	16:39	124	37155	10599	8892	80.69	77.80	5770/8658
	b=2	20:42	551	40177	58227	5870	87.25	40.83	6325/8658
eBWTclust	m=4 L=20	19:43	415	42973	2639	3074	93.32	94.21	7268/8658
	m=6 L=20	24:58	411	41972	630	4075	91.15	98.52	6940/8658
	m=4 L=31	35:56	314	42309	1487	3738	91.88	96.60	7233/8658
	m=6 L=31	22:19	300	40741	357	5306	88.47	99.13	6884/8658

Individual HG00100 vs reference (chromosome 16, 90338345bp), coverage 22× per sample

Tool	Param.	Wall Clock	RAM (MB)	TP	FP	FN	SEN (%)	PREC (%)	Non-isol. SNP
DiscoSnp++	b=0	6:20	200	48119	10226	18001	72.78	82.47	6625/11055
	b=1	31:57	208	53456	24696	12664	80.85	68.40	7637/11055
	b=2	51:45	1256	57767	124429	8353	87.37	31.71	8307/11055
eBWTclust	m=4 L=20	41:12	423	61264	943	4856	92.66	98.48	9314/11055
	m=6 L=20	43:51	419	58085	391	8035	87.85	99.33	8637/11055
	m=4 L=31	33:24	418	59668	898	6452	90.24	98.51	9287/11055
	m=6 L=31	44:53	337	53749	190	12371	81.29	99.64	8169/11055

DiscoSnp++ precision is much lower in general, and even drops with $b = 2$, especially with lower coverage, when inversely sensitivity grows. Sensitivity of DiscoSnp++ gets close to that of eBWTclust only in case $b = 2$, when its precision drops and memory and time get worse than ours.

Note that precision and sensitivity of DiscoSnp++ are consistent with those reported in [28]. In their paper (Table 2), the authors report a sensitivity of 79.31% and a precision of 72.11% for DiscoSnp++ evaluated on a Human chromosome with simulated reads (i.e. using an experimental setting similar to ours). In our experiments, using parameter $b = 1$, DiscoSnp++'s sensitivity and precision are, on average between the two datasets, 80.67% and 73.1%, respectively. Therefore, such results almost perfectly match those obtained by the authors of [28]. The same Table 2 of [28] shows that DiscoSnp++ can considerably increase precision at the expense of sensitivity by filtering low-ranking calls. By requiring $rank > 0.2$, the authors show that their tool achieves a sensitivity of 65.17% and a precision of 98.73%. While we have not performed this kind of filtering in our experiments, we note

that also in this case eBWTclust's sensitivity would be higher than that of DiscoSnp++. Precision of the two tools, on the other hand, would be comparable.

Finally, we note that also DiscoSnp++ has been evaluated by the authors of [28] using the SimSeq simulator (in addition to other simulators which, however, yield similar results). We remark that SimSeq simulates position-dependent sequencing errors, while our theoretical assumptions are more strict and require position-independent errors. Similarly, we assume a uniform random genome, while in our experiments we used real Human chromosomes. Since in both cases our theoretical assumptions are more stringent than those holding on the datasets, the high accuracy we obtain is a strong evidence that our theoretical analysis is robust to changes towards less-restrictive assumptions. We plan to test our (extended) prototype on real reads in a forthcoming extension of this paper.

5 Conclusions

We introduced a positional clustering framework for the characterization of breakpoints in the eBWT, paving the way to several possible applications in assembly-free and reference-free analysis of NGS data. The experiments proved the feasibility and potential of our approach. Further work will focus on improving the prediction in highly repeated genomic regions and using our framework to predict SNPs, predict INDELs, haplotyping, correcting sequencing errors, detecting Alternative Splicing events in RNA-Seq data, and sequence assembly.

References

1 C. Ander, O.B. Schulz-Trieglaff, J. Stoye, and A.J. Cox. metaBEETL: high-throughput analysis of heterogeneous microbial populations from shotgun DNA sequences. *BMC Bioinf.*, 14(5):S2, 2013.

2 M.J. Bauer, A.J. Cox, and G. Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theoret. Comput. Sci.*, 483(0):134–148, 2013.

3 E. Birmelé, P. Crescenzi, R.A. Ferreira, R. Grossi, V. Lacroix, A. Marino, N. Pisanti, G.A.T. Sacomoto, and M.-F. Sagot. Efficient Bubble Enumeration in Directed Graphs. In *SPIRE*, LNCS 7608, pages 118–129, 2012.

4 M. Burrows and D.J. Wheeler. A Block Sorting data Compression Algorithm. Technical report, DIGITAL System Research Center, 1994.

5 A.J. Cox, F. Garofalo, G. Rosone, and M. Sciortino. Lightweight LCP construction for very large collections of strings. *J. Discrete Algorithms*, 37:17–33, 2016.

6 A.J. Cox, T. Jakobi, G. Rosone, and O.B. Schulz-Trieglaff. Comparing DNA sequence collections by direct comparison of compressed text indexes. In *WABI*, LNBI 7534, pages 214–224, 2012.

7 D.D. Dolle, Z. Liu, M. Cotten, J.T. Simpson, Z. Iqbal, R. Durbin, S.A. McCarthy, and T.M. Keane. Using reference-free compressed data structures to analyze sequencing reads from thousands of human genomes. *Gen. Res.*, 27(2):300–309, 2017.

8 L. Egidi and G. Manzini. Lightweight BWT and LCP merging via the Gap algorithm. In *SPIRE*, LNCS 10508, pages 176–190, 2017.

9 D. Earl et al. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Gen. Res.*, 21(12):2224–2241, 2011.

10 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *FOCS*, pages 390–398, 2000.

11 S.N. Gardner and B.G. Hall. When Whole-Genome Alignments Just Won't Work: kSNP v2 Software for Alignment-Free SNP Discovery and Phylogenetics of Hundreds of Microbial Genomes. *PLoS ONE*, 8(12):e81760, 2013.

12 Z. Iqbal, I. Turner, G. McVean, P. Flicek, and M. Caccamo. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

13 K. Kimura and A. Koike. Analysis of genomic rearrangements by using the Burrows-Wheeler transform of short-read data. *BMC Bioinf.*, 16(suppl.18):S5, 2015.

14 K. Kimura and A. Koike. Ultrafast SNP analysis using the Burrows-Wheeler transform of short-read data. *Bioinformatics*, 31(10):1577–1583, 2015.

15 T.M. Kowalski, S. Grabowski, and S. Deorowicz. Indexing arbitrary-length k-mers in sequencing reads. *PLoS ONE*, 10(7), 2015.

16 B. Langmead and S.L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, 9(4):357–359, 2012.

17 R.M. Leggett and D. MacLean. Reference-free SNP detection: dealing with the data deluge. *BMC Genomics*, 15(4):S10, 2014.

18 R.M. Leggett, R.H. Ramirez-Gonzalez, W. Verweij, C.G. Kawashima, Z. Iqbal, J.D.G. Jones, M. Caccamo, and D. MacLean. Identifying and Classifying Trait Linked Polymorphisms in Non-Reference Species by Walking Coloured de Bruijn Graphs. *PLoS ONE*, 8(3):1–11, 03 2013.

19 C. Lemaitre, L. Ciortuz, and P. Peterlongo. Mapping-free and assembly-free discovery of inversion breakpoints from raw NGS reads. In *AlCoB*, pages 119–130, 2014.

20 H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

21 R. Li, C. Yu, Y. Li, T. W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

22 S. Li, R. Li, H. Li, J. Lu, Y. Li, L. Bolund, M.H. Schierup, and J. Wang. SOAPindel: efficient identification of indels from short paired reads. *Gen. Res.*, 23(1):195–200, 2013.

23 A. Limasset, J.-F. Flot, and P. Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *CoRR*, abs/1711.03336, 2017.

24 F.A. Louza, S. Gog, and G.P. Telles. Inducing enhanced suffix arrays for string collections. *Theor. Comput. Sci.*, 678:22–39, 2017.

25 F.A. Louza, G.P. Telles, S. Hoffmann, and C.D.A. Ciferri. Generalized enhanced suffix array construction in external memory. *Algorithms for Molecular Biology*, 12(1):26, 2017.

26 U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *SODA*, pages 319–327, 1990.

27 S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An extension of the Burrows-Wheeler Transform. *Theoret. Comput. Sci.*, 387(3):298–312, 2007.

28 P. Peterlongo, C. Riou, E. Drezen, and C. Lemaitre. DiscoSnp++: de novo detection of small variants from raw unassembled read set(s). *bioRxiv*, 2017.

29 P. Peterlongo, N. Schnel, N. Pisanti, M.-F. Sagot, and V. Lacroix. Identifying SNPs without a Reference Genome by comparing raw reads. In *SPIRE*, LNCS 6393, pages 147–158, 2010.

30 N. Philippe, M. Salson, T. Lecroq, M. Léonard, T. Commes, and E. Rivals. Querying large read collections in main memory: a versatile data structure. *BMC Bioinf.*, 12:242, 2011.

31 G.A.T. Sacomoto, J. Kielbassa, R. Chikhi, R. Uricaru, P. Antoniou, M.-F. Sagot, P. Peterlongo, and V. Lacroix. KISSPLICE: de-novo calling alternative splicing events from RNA-seq data. *BMC Bioinf.*, 13(S-6):S5, 2012.

32 L. Salmela and E. Rivals. LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514, 2014.

33 L. Salmela, R. Walve, E. Rivals, and E. Ukkonen. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics*, 33(6):799–806, 2017.

34 M. Schirmer, R. D'Amore, U.Z. Ijaz, N. Hall, and C. Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC Bioinf.*, 17(1):125, 2016.

35 J. Schröder, H. Schröder, S.J. Puglisi, R. Sinha, and B. Schmidt. SHREC: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.

36 F. Shi. Suffix Arrays for Multiple Strings: A Method for On-Line Multiple String Searches. In *ASIAN*, LNCS 1179, pages 11–22, 1996.

37 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015.

38 R. Uricaru, G. Rizk, V. Lacroix, E. Quillery, O: Plantard, R. Chikhi, C. Lemaitre, and P. Peterlongo. Reference-free detection of isolated SNPs. *Nuc.Acids Res*, 43(2):e11, 2015.

39 N. Välimäki and E. Rivals. Scalable and Versatile k-mer Indexing for High-Throughput Sequencing Data. In *ISBRA*, LNCS 7875, pages 237–248, 2013.