Journal Title Here, 2022, pp. 1–5

doi: DOI HERE
Advance Access Publication Date: Day Month Year
Paper

PAPER

Local alignment search with k-bounded matching statistics

Tommi Mäklin,[1,*] Jarno N. Alanko,[2] Elena Biagi[2] and Simon J. Puglisi[2]

[1]Department of Mathematics and Statistics, University of Helsinki, Pietari Kalmin katu 5, 00560, Helsinki, Finland and [2]Department of Computer Science, University of Helsinki, Pietari Kalmin katu 5, 00560, Helsinki, Finland
[*]Corresponding author. tommi.maklin@helsinki.fi
FOR PUBLISHER ONLY Received on Date Month Year; revised on Date Month Year; accepted on Date Month Year

Abstract

Finding high-quality local alignments between a query sequence and sequences contained in large genomic databases is a fundamental problem in computational genomics, at the core of thousands of biological analysis pipelines. Here, we describe an algorithm for approximate local alignment search based on the longest common suffix lengths of consecutive k-mer matches between a target and a query sequence (a 'k-bounded matching statistic'). We compute the k-bounded matching statistics using the Spectral Burrows-Wheeler transform (SBWT), a data structure that enables computationally efficient queries and an indexing. Our algorithm is available as a set of three `kbo` Rust packages that provide a command-line interface, a graphical user interface that runs in a browser and does not require server-side processing, and a library that can be accessed by other tools.

Key words: sequence alignment, genomics, Spectral Burrows-Wheeler transform, k-bounded matching statistics

Introduction

Genome analysis pipelines often begin with a step where the DNA contained in a genome sequencing run is mapped against a database of known reference sequences computationally. Tools implementing these algorithms make use of a number of indexing structures with different trade-offs in efficiency and accuracy and each come with their own strengths and weaknesses. Broadly speaking, the most typical trade-off is between runtime and accuracy depending on whether the algorithm attempts to find the best global match or search for a number of local matches. The latter is usually preferred in tasks involving large amounts of query and reference data.

Bacterial genomics is a particular field in which the increasing commercial and public health interest in translational research utilizing pathogen genomics has led to explosive growth in data volumes. From an algorithm design perspective, this growth has steered the field towards methods that preferably scale (sub)linearly in the number of reference sequences, as algorithms developed for careful analysis of a few dozen queries at most choke when applied to orders of magnitudes larger databases. Typically, the extra information contained in the larger volume of reference data is enough to justify some loss of accuracy or theoretical guarantees for optimal results. This has led to the adoption of k-mer based approaches that use minimizers or hashing techniques to upscale the throughput when compared to the more traditional approaches built on top of the Burrows-Wheeler transform. TODO: citations, add pseudoalignment

In this paper, we focus on the task of finding local alignment sections within a query sequence in a reference database containing either a small number of bacterial genomes or a larger number of gene sequences. In practice, these represent the two common bacterial genomics use cases of 1) producing reference-based alignments by locally aligning segments from a query against the reference and formatting the output as a string of nucleotide characters and gaps, and 2) comparing the identity and coverage of some genes of interest in a query against the same genes in the reference. For the first task, the most widely used tools are Snippy [**?**], which uses the Burrows-Wheeler aligner BWA-MEM [**?**] and thus guarantees some locality in the results, and split k-mer analysis SKA [**? ?**] which discards locality entirely in favour of rapid queries implemented using split k-mer hashing. The second approach is typically solved with locality preserving methods such as BWA-MEM, BLAST, or minimap2 [**?**] TODO citations / other relevant methods?. Our approach, called `kbo`[1], is competitive in both tasks 1 and 2.

`kbo` is built on top of the Spectral Burrows-Wheeler transform (SBWT), which allows rapid k-mer lookup in compact space [**?**]. Recently, it was shown that by adding longest common suffix information to the SBWT [**?**], it is possible to quickly compute the length of the longest matching suffix of each k–mer in the query against the indexed k-mers [**?**]. We show that these *k-bounded matching statistics*

[1] `https://github.com/tmaklin/kbo`

© The Author 2022. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

Fig. 1. Draft Overview of the kbo algorithms.

enable transforming the otherwise locality-discarding SBWT k-mer matching into an output format that preserves locality of consecutive k-mers in the query with a high probability TODO this should be proven or otherwise shown in the manuscript.

The main advantage of kbo lies in its effective use of the SBWT data structure to perform comparisons in a rapid and parallelisable manner. As an example, pairwise comparison of two 5Mb bacterial genomes (task 1) takes 1-2 seconds using a consumer-grade laptop and looking up a dozen genes of interest (task 2) less than a second. Neither of the tasks requires a separate indexing step, as SBWT construction is efficient enough to embed it in the query steps. Additionally, kbo is available as both a command-line interface and a WebAssembly distributable which enables running queries in the browser without sending any data to a remote server and without performing a local install of the tool.

Methods

Overview

Our tool, kbo, implements two main modes of operation: "find" matches the k-mers in a query sequence with the reference, and reports the local alignment segments found within the reference. Find is useful for problems that can be solved with BLAST. The second mode, "map", maps the query sequence against a reference sequence, and reports the nucleotide sequence of the alignment relative to the reference. Map solves the same problem as Snippy [**?**] and the map command in SKA [**?**]. A command-line interface to these two commands is provided by the kbo-cli Rust package [2] and the WebAssembly graphical user interface in the kbo-gui package [3]. An overview of the operation is provided in Fig. 1 and in the following paragraphs.

Under the hood, kbo-cli and kbo-gui call the functions provided by the kbo core library. This library is responsible for calling the sbwt Rust crate[4] to perform both indexing of the reference (in kbo find) or the query sequence (kbo map) and query the constructed index with the k-mers from the query (in kbo find) or the reference (in kbo map). The query step consists of extracting the k-mer matches and querying the longest common suffix (LCS) array for the matching statistics (MS) as described in a previous publication [**?**].

The resulting MS vector is fed to a *derandomization* algorithm introduced in this paper. This algorithm takes the MS vector and zeroes values that have a high probability to result from a random longest common suffix match. Using the derandomized values, kbo can then translate the resulting derandomized MS vector into a character string representing various levels of compatibility between the query and the reference. The character representation may further be refined to resolve single nucleotide polymorphisms (SNPs) in the query, as the plain derandomized MS vector does not contain enough information to distinguish between a SNP and a single base insertion. In kbo map, the refinement step is required, while in kbo find, the derandomized MS vector alone is enough to determine the output values. The next sections will cover each step in detail and introduce the derandomization, translation, and refinement algorithms.

[2] `https://github.com/tmaklin/kbo-cli`

[3] `https://github.com/tmaklin/kbo-gui`

[4] `https://docs.rs/sbwt`

k-mer matching with the spectral Burrows-Wheeler transform

In this section, we give a brief overview of the SBWT data structure that lies at the heart kbo. We provide here only the key definitions necessary to grasp its function and use; the motivation and theory behind the definitions are described in detail in prior work [**? ? ?**].

The SBWT sorts the last characters of the distinct k-mers in the input by the *colexicographic* (right-to-left lexicographic) order of their prefixes of length $k-1$. To give the precise definition, we first need to introduce the concept of the *padded k-spectrum*. First, the k-spectrum of a string is the set of distinct k-mers in the string.

Definition 1 (k-Spectrum) The k-spectrum $\mathcal{S}_k(S)$ of string a S is the set of distinct k-mers $\{S[i..i+k-1] \mid i = 1, \ldots, |S|-k+1\}$. The k-spectrum $\mathcal{S}_k(S_1, \ldots, S_m)$ of a set of strings $S_1, \ldots, S_m$ is the union $\bigcup_{i=1}^{m} \mathcal{S}_k(S_i)$.

Informally, a *padded k-spectrum* adds a minimal set of \$-padded *dummy* k-mers to ensure that in the de Bruijn graph of $S_1, \ldots, S_m$ and the dummer k-mers, every non-dummy k-mer has an incoming path of length at least k.

Definition 2 (Padded k-Spectrum) Let $R = \mathcal{S}_k(S_1, \ldots, S_m)$ be a k-spectrum with alphabet Σ, and let $R' \subseteq R$ be the set of k-mers Y such that $Y[1..k-1]$ is not a suffix of any k-mer in R. The padded k-spectrum is the set $\mathcal{S}_k^+(S_1, \ldots, S_m) = R \cup \{\$^k\} \cup \bigcup_{Y \in R'} \{\$^{k-i} Y[1..i] \mid i = 1, \ldots, k-1\}$, with special character $\$ \notin \Sigma$ and $\$ < c$ for all $c \in \Sigma$.

For example, if $S_1 =$ ACGT, $S_2 =$ GACG and $k = 3$, then $\mathcal{S}_3(X_1, X_2) =$ {ACG, CGT, GAC} and $\mathcal{S}_3^+(X_1, X_2) =$ {ACG, CGT, GAC, \$\$\$, \$\$G, \$GA}.

We are now ready to define the SBWT.

Definition 3 (Spectral Burrows-Wheeler transform (SBWT) [**?**]) Let R^+ be a padded k-spectrum and let $S_1, \ldots, S_{|R|}$ be the elements of R^+ in colexicographic order. The SBWT is the sequence of sets $A_1, \ldots, A_{|R|}$ with $A_i \subseteq \Sigma$ such that $A_i = \emptyset$ if $i > 1$ and $S_i[2..k] = S_{i-1}[2..k]$, otherwise $A_i = \{c \in \Sigma \mid S_i[2..k]c \in R^+\}$.

The Longest Common Suffix array is the key to efficient computation of the k-bounded matching statistics (to be defined in the next section):

Definition 4 (Longest Common Suffix (LCS) Array [**?**]) Let R^+ be a padded k-spectrum and let X_i be the colexicographically i-th k-mer of R^+. The LCS array is an array of length $|R^+|$ s.t. $LCS[1] = 0$, and for $i > 1$, $LCS[i]$ is the length of the longest common suffix of S_{i-1} and S_i.

TODO: example or point to fig 2. In the above definition, we consider the empty string as a common suffix of any two k-mers, so that the longest common suffix is well-defined for any pair of k-mers.

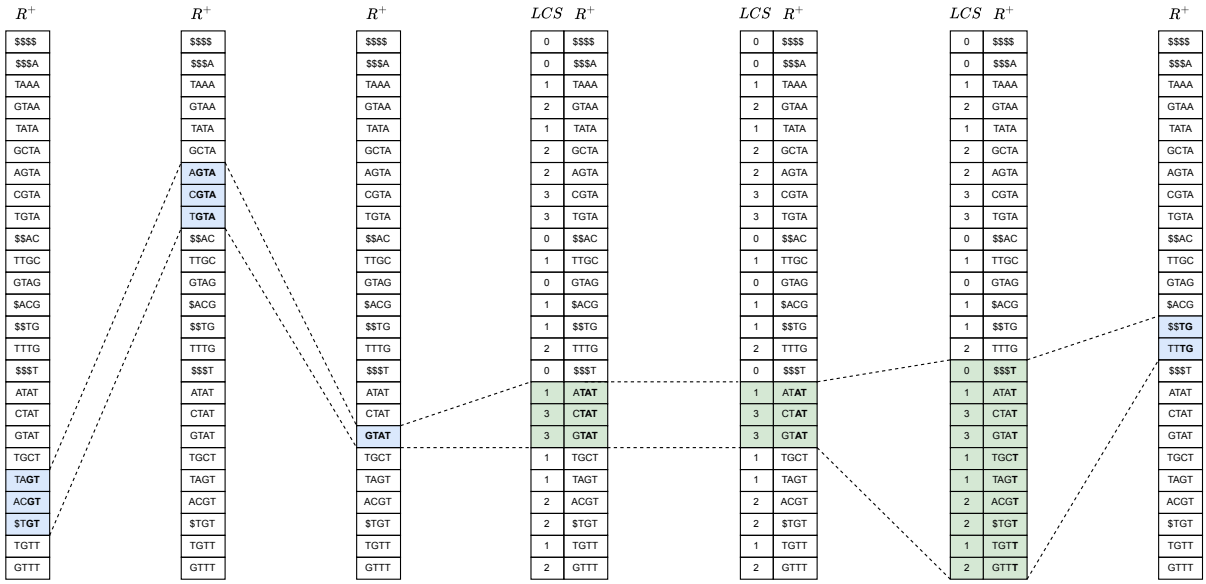

Fig. 2. Illustration of *ExtendRight* (resulting in blue intervals) and *ContractLeft* operations (resulting in green intervals) to query the string GTATG.

When then k-mers in R^+ colexicographically sorted, the subset of k-mers in R^+ that share a string α as a suffix are next to each other. This is called the *colexicographic range* of α. The colexicographic range of a string α that is *longer* than k is defined as the range of k-mers in the sorted list of R^+ that have the last k characters of α as a suffix. The SBWT provides two operations on colexicographic ranges $[\ell, r]$ of any string α:

- ExtendRight(ℓ, r, c) returns the colexicoraphic range of string αc (possibly an empty range).
- ContractLeft(ℓ, r), defined only if $|\alpha| \geq 1$, returns the colexicographic range of $\alpha[2..|\alpha|]$ if $|\alpha| > 1$ or R^+ if $|\alpha| = 1$. TODO:can we write this better? longest proper suffix of α if it exists, R^+ otherwise?

See Alanko et al. [**?**] for details on how to implement these operations using the SBWT.

k-bounded matching statistics

The k-bounded matching statistics are defined as follows:

Definition 5 The k-bounded matching statistics for a query string Q against a set of reference strings $S_1, \ldots, S_m$ is a vector $MS_k[1..|Q|]$ such that $MS_k[i]$ is the largest integer $l \leq k$ such that $Q[i - l + 1..i]$ is a substring of at least one reference $S_1, \ldots, S_m$.

In what follows, we drop the subscript of MS_k since k is assumed constant throughout the paper.

The padded k-spectrum R^+ provides an equivalent and formulation that is easier to compute: the value of $MS[i]$ is the length l of the longest *suffix of a k-mer* in the padded spectrum of $T_1, \ldots, T_m$ that matches $Q[i..i-l+1]$. The k-mers in R^+ that have suffix $Q[i..i-d+1]$ are adjacent in the colexicographically sorted list of k-mers in R^+ (the colexicographic range of $Q[i..i-l+1]$). Algorithm 1 shows pseudocode to compute the MS vector and the colexicographic ranges $[\ell, r]$ corresponding to the longest match at each position.

Algorithm 1 k-bounded matching statistics.

Require: A query string Q and a data structure supporting *ExtendRight* and *ContractLeft* on the padded k-mer spectrum.
Ensure: $MS_k[1..|Q|]$

```
ℓ, r ← [1, n]                          ▷ The current colexicographic interval
l ← 0                                  ▷ The length of the current match, up to k
MS_k[1..|Q|] ← Array of length |Q|.
for i = 1..|Q| do
    while l > 0 and ExtendRight(ℓ, r, Q[i]) = ∅ do
        ℓ, r ← ContractLeft(ℓ, r)
        l ← l − 1
    end while
    if ExtendRight(ℓ, r, Q[i]) ≠ ∅ then
        ℓ, r ← ExtendRight(ℓ, r, Q[i])
        p ← min(k, l + 1)
    end if
    MS_k[i] ← d                        ▷ The frequency of the match is r − ℓ + 1
end for
```

Distribution of longest common suffix lengths in random string matching

Querying the SBWT index for a query string Q produces a matching statistics (MS) vector $MS_k[1..N]$ such that $MS_k[i] \in [0, k], i = 1, \ldots, |Q|, |Q| \in \mathbb{N}^+$ denotes the length of the *longest common suffix* of the corresponding k-mer in Q - ending at index i - and the SBWT. For any reasonably large index, most k-mers will have a non-zero but small matching statistic due to random chance, the probability of which can be approximated by considering random matching of two strings of infinite length.

Let $\mathcal{X}$ be a random variable denoting the length t of the longest common prefix of two uniformly random and infinitely long strings $S_{1,\infty}$ and $S_{2,\infty}$. If $0 < p < 1$ is the probability that any pair of two characters in the two strings $S_{1,\infty}, S_{2,\infty}$ are mismatched, then X follows the geometric distribution with cumulative distribution function (CDF)

$$P(\mathcal{X} \leq t) = 1 - (1-p)^{t+1}, t \in \mathbb{N} \quad (1)$$

Suppose that an index $\mathcal{I}$ contains n uniformly random strings $S_{1,k}, S_{2,k}$ of length k instead. Assuming that k is large enough that the probability that two strings $S_{1,k}, S_{2,k}$ from the index match by chance is negligible, the longest common prefix of these two strings is reasonably approximated by the distribution of the random variable $\mathcal{X}$ (TODO this may need a proof or an example, there is one in Jarno's original writeup).

Now, let M be a random variable denoting the length of the longest common prefix between some string S_k of length k and the entire index $\mathcal{I}$ containing some number of strings of the same length k. Since distribution of the longest common prefix of the query string S_k and any string $S_i \in \mathcal{I}$ from the index $\mathcal{I}$ is given by the random variable $\mathcal{X}$, we can define a new random variable $\mathcal{M}$ which is the maximum of n independent random variables with the same distribution as $\mathcal{X}$ by setting $\mathcal{M} = \max\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$. Because the variables $\mathcal{X}$ were assumed independent, the CDF of $\mathcal{M}$ is the product of n random variables with the cumulative distribution function from Equation (1)

$$P(M \leq t) = P(\mathcal{X} \leq t)^n = \left(1 - (1-p)^{t+1}\right)^n \quad (2)$$

By noting that $p = 1 - q$, Equation (2) simplifies to the the cumulative distribution function

$$P(M \leq t) = \left(1 - q^{t+1}\right)^n \quad (3)$$

The CDF in Equation (3) can be used to derive a threshold for considering a longest common prefix length value as nonrandom with high probability.

Derandomizing k-bounded matching statistic vectors

Assuming some acceptable probability $0 < p < 1$, p small, for a random match length, we can derive a corresponding threshold $t_p, 0 < t_p \leq k$ for considering a match length nonrandom with some simple algebra from Equation (3)

$$t_p = \text{argmin}_t \left\{ \left(1 - q^{t+1}\right)^n > 1 - p \right\} \quad (4)$$

Todo: solve the equation above for t

Using the threshold t_p from Equation (4), we can create a *derandomized* version $d = d_1, \ldots, d_N$ of the matching statistics vector MS by iterating over MS in reverse and applying the following procedure on the elements $MS[n] = ms_N, \ldots, ms_1$

1. If $MS[n] > t_p$ and $MS[n+1] < MS[n]$, then $d_n = MS[n]$,
2. If $MS[n] = k$, then $d_n = k$,
3. By default $d_n = MS[n+1] - 1$

This will give a vector d of length N containing a positive value whenever the corresponding matching statistic is derived from a real k-mer match with high probability. Zeros in d mark the first base after end of a match, and negative values denote 0-indexed distance from the last matching base. The key to this simple procedure is reading the matching statistics vector MS in reverse and using the threshold t_p to determine when a new match begins after a gap.

Translating derandomized matching statistics into alignments

The derandomized MS vector can be translated into a character representation of the compatibility between the query and a reference sequence. The possible cases are displayed in Figure 3a. We encode the character representation using the following set of characters for the different possible compatibility events:

M: Match between query and reference.
-: Bases in the query that are not found in the reference.
X: Single base mismatch *or* single base insertion into the query.
R: Two consecutive 'R's signify a discontinuity in the alignment. The right 'R' is at the start of a k-mer that is not adjacent to the last character in the k-mer corresponding to the left 'R'. This implies either a deletion of unknown length in the query, or insertion of k-mers from elsewhere in the reference into the query.

The translation t_n for each value in the derandomized MS vector d_n depends on the neighboring values d_{n-1}, d_{n+1} and the threshold t_p. We use the following algorithm to determine the character representation of the events

1. If $d_n > t_p$ and $d_{n+1} > 0$ and $d_{n+1} < t_p$, then t_n = 'R' and t_{n+1} = 'R'.
2. If $d_n <= 0$, then there is a mismatch in the query and we apply
   a. If $d_{n+1} = 1$ and $d_{n-1} > 0$, then t_n = 'X'.
   b. Else t_n = '-'.
3. By default t_n = 'M'

Here we would also like to refer the reader to the documentation for the "translate_ms_vec" and "translate_ms_val" functions in the translate module [5] for the actual implementation of these algorithms.

Applying the translation procedure over a more complex derandomized MS vector with multiple events is illustrated in Figure 3b. Note how this case illustrates that the procedure is not able to distinguish between a single base insertion and a single base substitution. This problem can be resolved by adding a *refinement* step to our translation, which resolves this conundrum by using the nucleotide sequence of the reference.

Refining the translation to resolve SNPs versus insertions

TODO this needs a description of how the base is deemed as a SNP (substitution?) or an insertion in the first place.

We can resolve the problem of single nucleotide substitution versus an insertion by using the colexicographical ordering stored in the LCS array[**?**]. Briefly, this data structure allows retrieving the neighboring (in the colexicographical ordering) bases from the SBWT and provides an educated guess for the value of the base marked with an X. The SBWT by itself does not store the original sequence but using the LCS array allows us to derive a workaround.

We resolve the 'X's by checking whether the k-mer whose first character overlaps the 'X' has a matching statistic of $k-1$ and, if it does, traversing the colexicographical ordering until we find the k-mer where the position marked by 'X' is the middle base (a split k-mer centered on 'X' [**?**]). We use the middle base from this k-mer to replace the 'X' in the translation. If the matching statistic is less than $k-1$, the character is checked from the k-mer that is $(t_p+1)/2$ characters

[5] https://docs.rs/kbo

Fig. 3. Draft Applying the translation procedure to a derandomized matching statistics vector.. Panel a) shows the result of translating a single position at a time. Panel b) shows the result of translating an entire vector with multiple cases. Examples in both panels assume k-mer size 3 and derandomization threshold 2.

away from the 'X' to ensure account for the possibility of a random match.

Although the colexicographical ordering is not guaranteed to return the exact k-mer that is present in the query at this position, in practice the approach selects the same nucleotide as Snippy and SKA with a high probability (TODO add figure from a simple example).

Use cases

Reference-based alignment of query assemblies with 'kbo map'

Specific to alignment

- alignments are comparable to Snippy and SKA map
- alignment is faster than Snippy or SKA map
- alignment scales easily to 10s of thousands of genomes.
- compare w/ Snippy, SKA map

Finding reference gene sequences in a query assembly with 'kbo find'

Specific to find

- Quick if it's not needed to know which genes exactly are present
- Slower if the gene names (= individual contigs are needed). this could be implemented more efficiently with colors.
- compare w/ blast

Running kbo in the browser

- efficient resource use: no prebuilt indexes, no temporary space, low memory consumption, runs ok on a single thread

Available online [6].

Browser queries for AllTheBacteria genomes

- TODO check feasibility of using some served genomes as the query or the target.
- TODO make this available as an application for the AllTheBacteria project.

Discussion

In no particular order:

- Adding colors: matching statistics queries
- index compression via pangenomes or colorset compression: we can compress 45 000 genomes from the same species into a few gigabytes using the meta-diff indedx in Fulgor, implementing this in SBWT with efficient MS queries would enable simultaneous alignment against tens of thousands of genomes.
- possibilities for gene variant calling with colors
- finimizers: these together with the recombination markers output by `kbo` (RR) could be used to identify potential recombination segments. Finimizers might additionally allow even more efficient queries with a prebuilt index?

[6] https://maklin.fi/kbo

- Mention how the webassembly version enables implementing tools like amrfinder, kaptive, and tree inference in the browser. Use the pks island gene typing as an example of how the custom database inclusion can be utilised.

Code availability

The Rust[7] library, command-line interface, and graphical user interface implementing the methods described here are freely available from Codeberg `https://codeberg.org/themaklin/sablast` and GitHub `https://github.com/tmaklin/kbo` under a MIT[8] and Apache 2.0[9] dual license.

Competing interests

No competing interest is declared.

Author contributions statement

T.M. designed the kbo algorithms and implemented the kbo, kbo-cli, and kbo-gui Rust libraries. J.N.A. implemented the sbwt Rust crate and derived the random match distribution. T.M., J.N.A, E.B., and S.J.P. investigated the relationship between k-bounded matching statistics and local alignments. S.J.P. obtained funding and supervised the study. All authors contributed to the writing, reviewing, and editing of the manuscript.

Acknowledgments

This work was supported in part by the Research Council of Finland via grant 351150. TM was supported by the Research Council of Finland (EuroHPC grant).

[7] `https://www.rust-lang.org/`

[8] `https://opensource.org/license/mit`

[9] `https://www.apache.org/licenses/LICENSE-2.0`