Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: 15th December 2016
Manuscript Category

Bit Packed Suffix Array

Geetika Babu Bangera [1], Daljeet Singh Chhabra [2]* and Prachi Poddar [3]

[1]gbangera@cs.stonybrook.edu, 110899604
[2]dtchhabra@cs.stonybrook.edu, 110465230 and
[3]ppodar@cs.stonybrook.edu, 111111111.

*To whom correspondence should be addressed.

Abstract

Motivation: As the volume of data available for computing continues its exponential growth, compressing in all forms grows in importance. The general suffix array implementation uses one byte per character in memory to store the input text. However, the DNA/RNA alphabet contains only 4 characters (A, C, G, T) which can be encoded using 2 bits of data. This will reduce the total size of the text by huge margin (around 75%) and is ideal when the input file (transcriptomes) are in GBs or greater than that.
Results: The time to create the suffix array approximately doubles but the size used to store the transcriptome text reduces by 75%.

1 Overview

The project entails programming a suffix array library using binary encoding, along with associated functions - creating and search. We have used RapMap as the base code for this project and stripped it down to act as a wrapper to parse input and generate output files. This version of RapMap that uses the encoded suffix array library is referred to as "Packed RapMap" hereafter in the report. The results for the performance comparison between RapMap and Packed RapMap are provided in the end.

2 Approach

LibDivSufSort is an open source suffix array library. We modified LibDivSufSort to operate on compressed text. For this purpose, we introduce the notion of Transcriptome Stream. A transcriptome stream contains the concatenated transcripts from input text in compressed form using 2 bits per nucleotide. Besides the compressed text, it also stores a bit array that indicates where each transcript ends. The Transcriptome Stream is implemented as a C++ class and the modified LibDivSufSort operates on objects of this class.

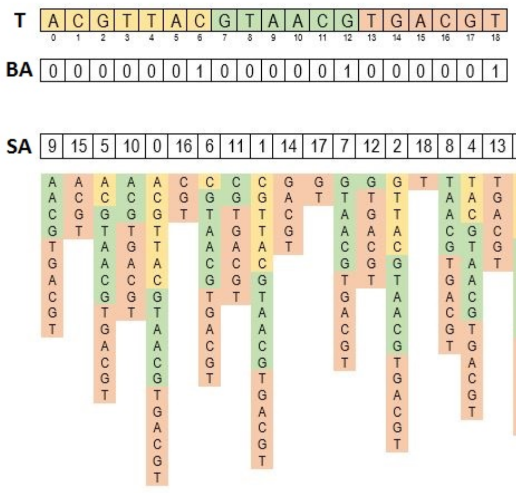

Fig. 1. SA when $ is removed from transcriptome

2.1 SA Construction

Instead of using one byte per character, only two bits are used to store each character: A -> 00, C -> 01, G -> 10, T -> 11.
Since we have only 2 bits per character, the terminator '$' used by original RapMap cannot be used. The end of transcript can only be determined by the bit array. If we do not use the information in bit array while creating the suffix array, it leads us to the problem that the suffixes of transcriptome that span across multiple transcripts occur in wrong places in the suffix array as explained in fig **??**.

In the example above, consider the pattern 'ACGT'. It occurs 4 times in the concatenated transcriptome. However, out of the four occurrences, two times the pattern crosses the transcriptome boundary and therefore must not be included in the result. The main problem here is that the correct indexes where the pattern match for 'ACGT' should succeed are no more contiguous in the suffix array. Ideally, they should occur together in order for the pattern search results to be correct without changing the asymptotic complexity of the creation or search algorithm.

This is not an issue in the original RapMap as '$' is present at the end of every transcript. The characters are lexicographically sorted as shown below:

$ < A < C < G < T

This makes sure that if the same pattern appears multiple times in the transcriptome, the '$' in the middle of the suffix makes the pattern lexicographically smaller than other patterns which do not have a '$' sign in the middle. This is shown in fig **??**.

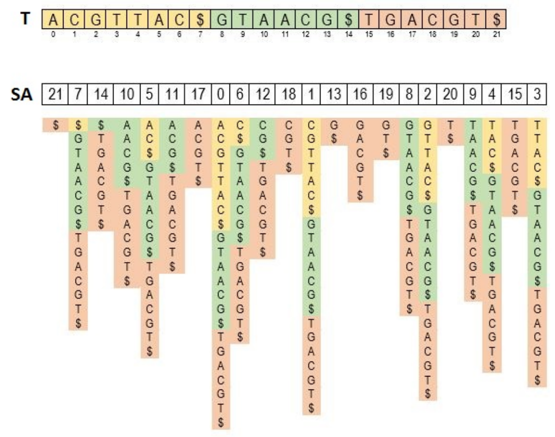
Fig. 2. SA using default RapMap

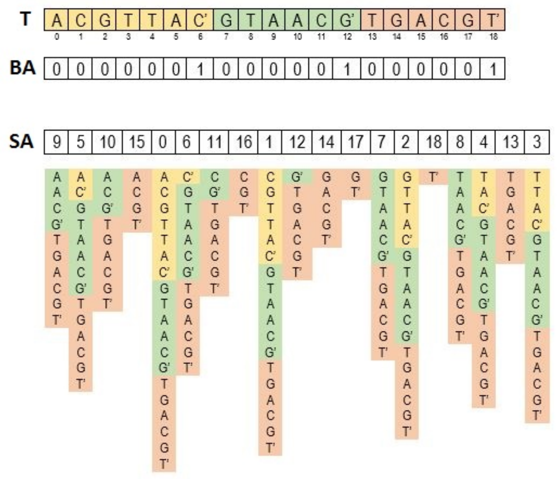
Fig. 3. SA Created with Bit Packed data

To resolve this issue, we introduce the notion of prime characters. The prime characters (A', C', G' and T') are defined as follows: A' < A < C' < C < G' < G < T' < T

We replace the last character of each transcript by its equivalent prime character. That is - replace A -> A', C -> C', G -> G' and T -> T'. Note that this is just a notational replacement and is not directly implemented in code. The number of bits per nucleotide is only 2 and therefore we cannot have more than 4 characters(A,C,G,T). The primes are only calculated while creating the suffix array by consulting the bit array that marks the end of transcripts. The following figure shows the SA that is created using the substituted text.

Note that SA created is similar to the one that is created using the earlier implementation that uses characters. The only difference is that there are no '$' inside the suffixes. The relative order of the suffixes in the SA created using old approach and the SA created using the new approach is exactly the same. This means that the patterns which do not span across different transcripts are now contiguous in the suffix array.

2.2 SA Search

With the introduction of prime characters, we get a SA in which the all the suffixes whose prefix match a pattern are contiguous in the SA. However, introducing primes has one side effect on the search algorithm. Recall that : A' < A < C' < C < G' < G < T' < T

Now consider the example in fig **??**. If we use the relationship between primes mentioned above, the search for pattern 'ACGT' will return only 1 result(SA[4]). However the correct solution must report 2 results - SA[3] and SA[4]. This is because T' < T and so the match fails. This is incorrect because the match that occurs at the end of transcript is a valid match.

If we treat primes as equal to their respective characters then the search will report 4 results: SA[1] - SA[4]. This is also incorrect because the suffixes at index SA[1], SA[2] span across two different transcripts.

To solve these problems, we make 2 modifications to the search algorithm -

1. Treat primes equal to their respective characters i.e. $A' = A < C' = C < G' = G < T' = T$
2. While matching the i^{th} character in the transcriptome with the j^{th} character of pattern, check if the $i - 1^{th}$ character occurs at the end of transcriptome. If yes, treat the text as inherently smaller than the pattern and look for pattern matches in the right half of the binary search.

With these 2 changes, the search method reports all the correct occurrences of the pattern in the text.

3 Results

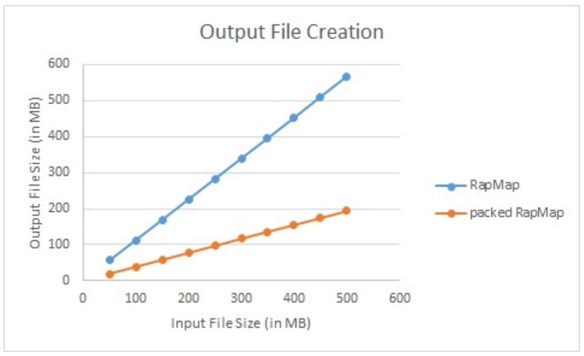

Fig. 4.

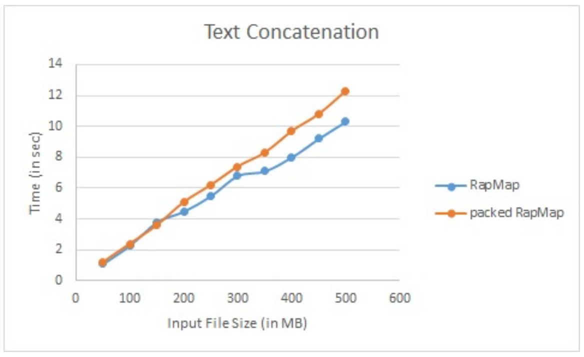

Fig. 5.

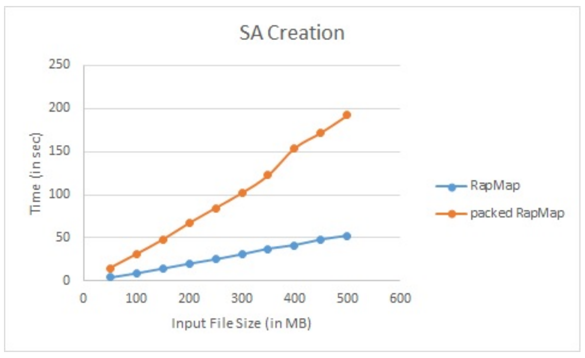

Fig. 6.

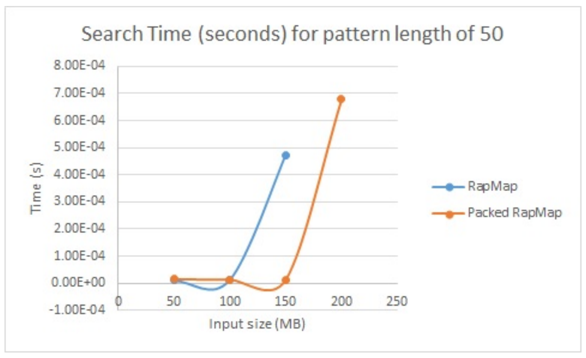

Fig. 8.

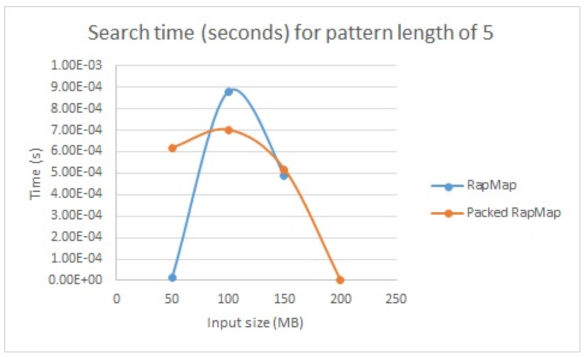

Fig. 7.