Journal of Computer Science & Its Applications, December 2006, Vol. 13 No.1

A Subquadratic Time Generation of $|\sum|^q$ Dictionary from String(s) using Suffix Tree

Ezekiel F. Adebiyi
Department of Computer and Information Sciences
Covenant University
PMB 1023, Ota, Nigeria.
E-mail: eadebiyi@sdsc.edu

Abstract.
Given a set of alphabets $\sum$ (whose size is $|\sum|$) and set of subject strings $X = \{x_1, x_2, \ldots, x_k\}$ over $\sum$, such that $n = \sum_{i=1..k} xi$, the problem considered in this paper, is the generation of items of the dictionary $|\sum|^q$, where q is a positive integer greater than 1, from the strings in X. This stringology problem finds application in the bioinformatics problem of motifs finding[10] and recently in the selection of oligonucleotides for Microarray and PCR[13, 14]. Microarray and PCR are both academic and industrial tools of Bioinformatics used in Life Sciences.

A naive algorithm that runs in $O(|\sum|^q)$ time was given recently by Zheng et al.[13, 14] to solve this problem. In this paper, we present a novel algorithm based on the suffix tree that solves this problem in subquadratic time.

Keywords. *Stringology, Suffix tree, dictionary, seeds, motifs, oligonucleotides, microarray, PCR.*

1.0 Introduction

In bioinformatics, the following stringology problem is an important step in motif finding[10] and recently in the selection of oligonucleotides for microarray and PCR[13, 14]. Typical setup include k subject strings $X = \{x_1, x_2, \ldots, x_k\}$, such that $n = \sum_{i=1..k} xi$, where strings $x_1, x_2, \ldots, x_k$ are generated over a set of alphabets $\sum$, whose size is $a = |\sum|$. The problem is then to list out all $|\sum|^q$ substrings that appear in X, where q is a positive integer greater than 1. Since all possible of $|\sum|^q$ substrings, not known in advance may occur in X, a naive approach employed in [13, 14] is to create $|\sum|^q$ buckets (if q = 2 and $|\sum| = 4$ for DNA, $4^2 = 16$ buckets include AA, AC, AG, AT, …, TT) and slide a ruler of length q through each subject strings to collect all possible $|\sum|^q$ substrings. To collect the occurrence statistics of the $|\sum|^q$ buckets that occur in X, in the worst case, a $O(|\sum|^q)$ time is required. We show in this paper how a novel algorithm designed using the suffix tree is cleverly engineered to do this in O(n ln n/ln a) time. For compactness, let us call the $|\sum|^q$ substrings seeds of length q (henceforth, q-seeds). Our resulting algorithm has been used in Adebiyi[3] and Adebiyi and Olarenwaju[4] to design efficient sequential and parallel algorithm for oligonucleotides selection.

1.1 Organization of the paper

This paper is structured as follows. In section 2, we discuss the suffix tree and present our novel algorithm in section 3. We discuss the implementation of our algorithm and our experimental experience using this algorithm in section 4. We conclude this paper in section 5.

1.0 Suffix tree

Some good materials on suffix tree can be found in Adebiyi[1] and Gusfield[6]. We
give a brief description below. The following informal description is taken from [1].

A suffix tree is a lexicographically inter-connected data structure, that provide efficient access to all substrings of a strings, over which it is built. This data structure can be constructed and represented in linear time and space. And this has enable the solution of many strings problem in linear time. The construction of a suffix tree in linear time can be found in Weiner[12], McCreight[8] and Ukkonen[11]. A recent paper by Kurtz[7] discussed how an economical construction of suffix tree with respect to space can be done.

We give here the definition of a suffix tree for an arbitrary string x of length N over an alphabet $\sum$ as present in [6]. We follow this to show how a suffix tree can be built for a set of strings.

Definition 1 A suffix tree of a N-character string x is a rooted directed tree with exactly N leaves, numbered 1 to N. Each internal node, other than the root, has at least two children and each edge is labelled with a nonempty substring of x. No two edges out a node can have edge-labels beginning with the same characters. The key feature of the suffix tree is that for any leaf i, the label of the path from the root to leaf i exactly spells out the suffix of x that start at position i. Note that the definition above does not guarantee the existence of a suffix tree for any string x. The problem is that if a preffix of a suffix of x matches a suffix of x, the path for the later suffix would not end at a leaf. Therefore, to guarantee the existence of a suffix tree for any string x, we place at the end of x a special symbol that is not in the alphabet $\sum$. We use in this paper, the symbol \$, for the termination character. Below is a suffix tree for x = GTATCTAGG. The number at the leaves indicate the starting position of the corresponding suffixes.

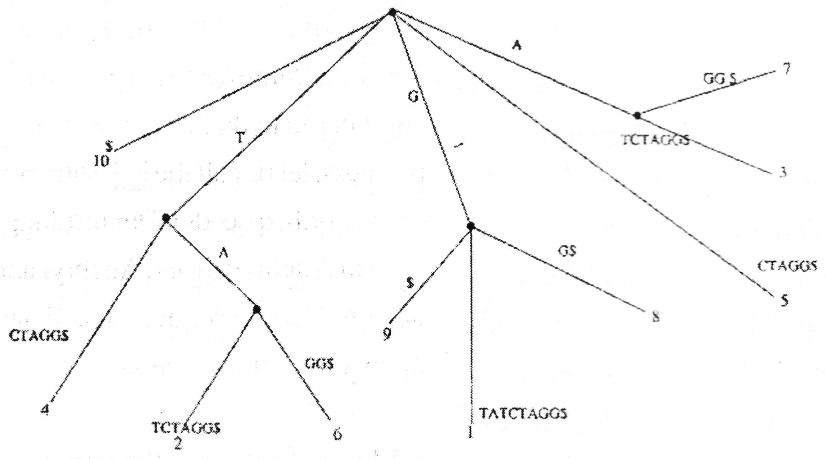


Fig. 1. The suffix tree for $x = GTATCTAGG$

The suffix tree constructed for a set of strings is called a generalized suffix tree, can be easily achieved by consecutively building the suffix tree for each string in the set.

The resulting suffix tree is built in time proportional to the sum of all the string lengths. The leaf number of the single string suffix tree can easily be converted to two numbers, one identifying the string and the other identifying the starting position in that string. Below is the generalized suffix tree (henceforth GST) for the strings x1 = TACTA and x2 = CACTCA.

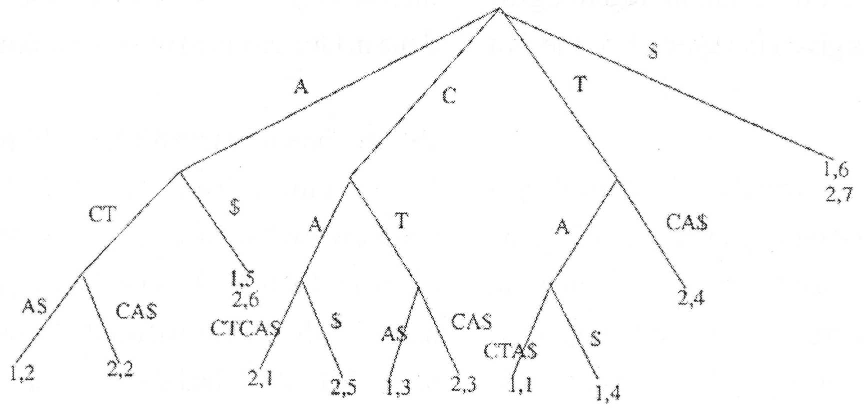


Fig. 2. The generalized suffix tree (GST) for $x_1 = TACTA$ and $x_2 = CACTCA$

3.0 The Subquadratic running time Algorithm

We state first some important preliminaries. Note that two types of exact repeats exist: maximal and non-maximal. An exact repeat β is maximal if aβc and bβd occur in S for some a≠b and c≠d, a, b, c, d ε Σ. Otherwise the repeat is non-maximal.

A maximal repeat is a proper substring of another maximal repeat, called the supermaximal repeat. Note that the q-seeds formally described above can be define as follows.

Definition 2 A q-seed is a substring of length q that occur in at least one location in the set X.

Let each bucket obtained via q-seed i be B_i. Else where[2], using the suffix tree for solving a different problem, we use among other definitions and observation the following definition and corollary to characterize seeds to extract them. We discovered that the characterization also finds application here partly.

Observation 1 A substring that is completely non-maximal in all strings in {xi,| i = 1...k} cannot be a possible seed, since it is contained in some supermaximal repeats.

Definition 3 A repeat seed is a repeat (i.e., non-maximal, maximal or supermaximal) in at least two of the subject strings.

Noting the above, the seeds of interest in this paper as defined in definition 2 can be characterized using the following corollary:

Corollary 1 In addition to the q-length constraint, observe that definition 3 above will capture all the q-seeds of the dictionary 4^q extractable from the strings in X, except for the q-seeds that exist only on one sequence. And because maximality as depicted in observation 1 is not a constraint in definition 2, the q-seeds we desire can also end between two nodes, in addition to those that exist at the nodes.

Using the following definitions and notations, the algorithm encapsulating all the observations regarding the generation of q-seeds is given in figures 3, 5, and 4 of the appendix.

- seeded_x: A boolean variable labelling all locations (leafs (seed ending in between or at the end) and nodes (seed ending at the end or in between two edges)) where seeds are found. And seeded_x is true if a leaf or a node spelled (after Sagot[9]),a seed.
- bucklen: Total # of entries in a bucket, bucklen $\leftarrow$ 0.
- hitscount: Total # of buckets, i.e. Total # of q-mers extracted, hitscount $\leftarrow$ 0.
- lencount: Accumulated sum of edge lengths as we do the depth first traverse of the suffix tree.
- succ: successive children of a node x.
- depth: accumulated length of edges along a path in a suffix tree.

Lemma 1. Using corollary 1, all the q-seeds of the dictionary $|\Sigma|^q$ extractable from the strings in X can be done in O(n ln n/ln a) worst case time.

Proof. The suffix tree for X can be build in O(n) time. Note that for each q-seeds required for a particular q length, we will not need to traverse all nodes and edges of the suffix tree. But considering a worst case scenario, in which case, we need to traverse all edges and nodes. Blumer[5] had shown that the number of nodes in a suffix tree for a random text x on an alphabet of size a = $|\Sigma|$, under a model of equiprobability and independence of the characters in X, is F(n) ~ (Fc + Fo(n))n, where in this case

$$F_c = \frac{1}{\ln a}[a \ln a - (a-1)\ln(a-1)] \tag{1}$$

and

$$F_o(n) = \frac{(a-1)}{\ln a} \sum_{j\neq 0}[1 - e^{-2\pi ij/\ln a}]\Gamma\left(-1 + \frac{2\pi ij}{\ln a}\right)e^{-2\pi ij \ln n/\ln a} \tag{2}$$

is a small oscillating factor with exponentially increasing period and n accounts for the prefixes of X. It is straightforward now to extend that the number of edges of a suffix is bound by O(n ln n/ln a). Note that the number of edges on the path from the root to any leaf is (ln n/ln a) and the numer of leaf is O(n). Therefore, the number of edges on a suffix tree can not be more than O(n ln n/ln a). Since the traversing executed in our algorithm is a function of the number of nodes and edges, therefore, our algorithm worst case run time (plus the time required to build suffix tree) is bound from above by O(n ln n/ln a).

4.0 Experimental Experience

We have implemented our subquadratic algorithm in C under Linux based on Kurtz[7] space efficient implementation of a modified McCreight suffix tree and tested it also under Linux on a PC with Pentium IV CPU and 512MB RAM. We bench-marked our algorithm (Alg B) with the naive algorithm in [13, 14] (Alg A).

To compare output, the dataset used is the barley ESTs from HARVEST 1.45 downloaded from http://harvest.ucr.edu/. It contains k = 53; 240 unigenes, with n = 43, 464, 144 bases. We downloaded from http://harvest.ucr.edu/ another 7.41 KB of EST unigenes to compare their efficiency.

Algorithm A and algorithm B, for q = 11 and q = 12 extracted the following same number of seeds in table 1(a,b), grouped according to the number of times, they occurred. For q = 13, our algorithm (Alg B) produced the seeds of table 1(c), for which the naive algorithm (Alg A) failed to run. This is because the exponential space requirement of the naive algorithm is astronomical too large to be accommodated on the PC we used to carry out our experiment.

As expected theoretically, the run times (on the same system) of the two algorithms, that is, Alg A and Alg B on the 7.41 KB of EST unigenes dataset are depicted in table 1(d). For q = 13, because of the astronomical increase in 4^q, algorithm A will not run.

Table 1. Distribution of seeds extracted from the barley ESTs for a) $l = 33$ ($q = 11$) ,b) $l = 36$ ($q = 12$), c) $l = 39$ ($q = 13$) and d) the running times ($\times 10$) of algorithm of A and B for different values of q on the x-axis

(a)

# of occurrences	# of seeds
1-9	2801351
10-19	924719
20-29	199790
30-39	55737
40-49	20820
50-79173	26694

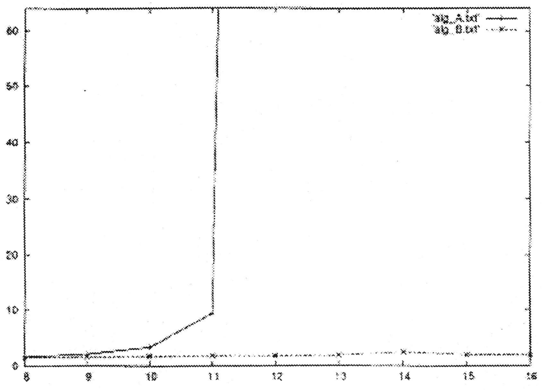
(b)

# of occurrences	# of seeds
1-9	11232806
10-19	303522
20-29	34718
30-39	9465
40-49	4065
50-67117	6931

(c)

# of occurrences	# of seeds
1-9	20385811
10-19	77375
20-29	9747
30-39	3084
40-49	1621
50-58110	3408

(d)



5.0 Conclusion

A subquadratic algorithm has been designed to generate the $|\Sigma|^q$ dictionary which finds applications in the bioinformatics problems of motifs finding and selection of oligonucleotides. The algorithm is designed using the suffix tree, cleverly engineered via the fine lexicographic interconnected structure of the suffix tree to run in a subqradratic time instead of an existing novel algorithm that runs in exponential time.

Acknowledgment

We thank the anonymous referee of our paper accepted at the Intl Workshop and Conf. on new trends in the Mathematical and Computer Science with application to real world problems, June 19-23, 2006, whose careful review work has necessitated the new technical exposition in this paper. This work is partially supported by the Covenant University Senate Research Grant 2004/2005. Part of this work was done, while the author was at LIRMM, France on a CNRS-NEPAD special grant.

References

1. Adebiyi, E. F. (2002) Pattern Discovery in Biology and String Sorting: Theory and Experimentation. Shaker Publisher, Aachen, Germany.
2. Adebiyi, E. F., Kaufmann, M (2002) Extracting Common Motifs under the Levenshtein Measure: Theory and Experimentation. Proc. of the 2nd Workshop on Algorithm of Bioinformatics (WABI), Italy and also appeared in Lecture Notes in Computer Science, 2452, 140-156.
3. Adebiyi, E. F. (2006) Using Suffix Tree for Efficient Selection of Unique oligos for large EST databases, Submitted to WABI 2006 and International Journal of Bioinformatics and Computational Biology (IJBCB)
4. Adebiyi, E. F and Oyelade, J. O. (2006) A Comparative Analysis of existing Oligonucleotides Selection Algorithms and Optimal Parallel Oligos Selection for large EST Databases (Extended Abstract). Accepted (peer reviewed) proc. of the Intl Workshop and Conf. on new trends in the Mathematical and Computer Science with application to real world problems, June 19-23, 2006.
5. A. Blumer A. Ehrenfeucht, and D. Haussler. Average Size of Suffix trees and DAWGS. Discrete Applied Mathematics, 24, 37-45, 1989.
6. D. Gusfield. Algorithms on Strings, Trees and Sequences. Cambridge University Press, New York, 1997.
7. Kurtz, S. (1999) Reducing the Space Requirement of Suffix trees. Software-Practice and Experience, 29(13):1149-1171.
8. McCreight, E. M. (1976) A Space-Economical Suffix tree Construction Algorithm. Journal of ACM 23(2): 262-272.
9. Sagot, M-F. Spelling Approximate Repeated or Common Motifs using a Suffix tree, LNCS 1380, 111-127, 1998.
10. Tompa, M. (1999) An Exact Method for Finding Short Motifs in Sequences with Application to the Ribosome Binding Site Problem, 7th Intl. Conf. Intelligent Systems for Molecular Biology (ISMB), 262-271.
11. Ukkonen, E. (1995) On-line Construction of Suffix trees. Algorithmica, 14:249-260.
12. Weiner, P. (1973) Linear Pattern Matching Algorithm. Proc. 14th IEEE Sym. on Switching and Automata theory, 1-11.
13. Zheng, J., Close, T., Jiang, T., and Lonardi, S. (2003) Efficient Selection of Unique and Popular Oligos for large EST Databases, CPM 2003, LNCS 2676, 384-401.
14. Zheng, J., Close, T., Jiang, T., and Lonardi, S. (2004) Efficient Selection of Unique and Popular Oligos for large EST Databases, Bioinformatics, 20(13), 2101-2112.

Appendix

```
1.  FINDHITS($X, q$)
2.  $ST \leftarrow$ Build $GST(X)$
3.  for root children $x$ (in depth first format)
4.     if($x$ is a leaf)
5.        $len$ = # of characters that constitute the leaf branch.
6.        if($len \geq q$)
7.           $seeded_x = 1$
8.           $hitscount = hitscount + 1$
9.           $bucklen_{hitscount} = 1$
10.          $B_{hitscount} \leftarrow x$
11.    else if($x$ is a node)
12.       $lencount = depth$, $bucklen = 0$
13.       if($q == lencount \mid\mid q < lencount$)
14.          $seeded_x = 1$
15.          hitscount=hitscount+1
16.          LISTOCCPOS($x$, $ST$, $hitscount$)
17.          $B_{hitscount} \leftarrow x$
18.    else DEPTHFIRST($x$, $lencount$, $q$, $ST$)
```

Fig. 3. The subquadratic algorithm for extracting q-seeds