# DISTRIBUTED SUFFIX TREES AND THEIR APPLICATION TO LARGE-SCALE GENOMIC ANALYSIS

RAPHAËL CLIFFORD and MAREK SERGOT

*Department of Computing, Imperial College, London.*
*E-mail: raphael@clifford.net and m.sergot@ic.ac.uk*

We have recently presented a variant of the suffix tree which allows much larger genome sequence databases to be analysed efficiently. The new data structure, termed the *distributed suffix tree* (DST), is designed for distributed memory parallel computing environments (e.g. Beowulf clusters). It tackles the memory bottleneck by constructing subtrees of the full suffix tree independently. The standard operations on suffix trees of biological importance are easily translatable to this new data structure. While none of these operations on the DST require inter-process communication, many have optimal expected parallel running times.

## 1. Introduction

The suffix tree is the key data structure of computational pattern matching allowing a multitude of sophisticated operations to be performed efficiently (see e.g. [1, 5]). In the field of bioinformatics these operations include whole genome alignment [3], analysis of repetitive elements [8], and fast protein classification [4], amongst many others. However, the main obstacle to more widespread acceptance of these methods remains that of memory use. Suffix trees have high memory overheads, and the poor memory locality, both of their construction and of querying algorithms, make disk-based implementations highly problematic [7].

We have presented in [2] two new data structures for problems of intermediate size—that is, problems larger than can be handled by existing suffix tree/array methods but small enough that the input can be stored entirely in real memory—a range of at least an order of magnitude. To give some indication, the new methods allow us to store and analyse the whole human genome, perform cross species pattern matching on all available bacterial genomes at once, or search a large EST database, using a small cluster of standard PC's. The data structures are termed the *distributed suffix tree* (DST) and the *paged suffix tree* (PST). Both are based on a new

extension of Ukkonen's suffix tree construction algorithm [9] which allows subtrees of a suffix tree to be constructed efficiently in space proportional to the size of the resultant data structure and not the whole suffix tree. This enables a suffix tree to be distributed over a number of computing nodes and queried in parallel (the DST) or for a single node to compute independent subtrees successively (the PST). By effectively splitting the input string lexicographically (not into contiguous substrings), it can readily be shown that all the most popular biologically inspired operations on suffix trees exhibit optimal or near optimal parallel speedups. Furthermore problems which would previously have been impossible to solve due to their size can now be tackled efficiently, either in parallel or serial and with modest hardware requirements. Here we will focus on the distributed version, the DST.

The DST construction algorithm has been implemented in C on an 8 processor distributed memory parallel computer, increasing by a factor of 7.65 the size of the largest database that could be indexed. Exact set matching and repeat finding procedures for random data have also been implemented and performed on the DST. The results showed substantial speedups (with average efficiencies in excess of 90% and 99%, respectively) and exhibited good scalability, confirming the theoretical analysis. For systematically biased genetic data, preliminary results show that simple load balancing schemes can successfully increase the parallel efficiency of biological operations to close to 90%.

The method is simple to apply. Almost any current bioinformatic technique that relies on suffix trees can be modified to take advantage of DSTs, greatly extending the range of problem sizes that can be tackled. Also, complex or time consuming queries, such as the preliminary stages of matching all ESTs against the human genome, can be performed with optimal or near optimal efficiency in parallel. In the next section we first describe the new data structure. We then present the expected time efficiencies of a representative sample of operations on the DST.

## 2. Distributed Suffix Trees

A suffix tree of input string $t$ is a compacted trie of the suffixes of $t$. We define a *sparse suffix tree* (SST) of input string $t$ to be a compacted trie of a subset of the suffixes of $t$. Here, we are interested in the special case where all the suffixes in this subset start with the same prefix $z$ and assume from now on that all SSTs are of this type. *Distributed suffix trees* (DST)

are simply collections of SSTs defined in this way.

Usually, a single SST will be held at each computing node and the union of the path labels of the leaves of these SSTs will be the full set of suffixes of $t$. In other words, every suffix of $t$ will be represented by exactly one SST at exactly one of the computing nodes. An example DST and the corresponding standard suffix tree are given in Figures 1 and 2.

In this case the prefixes for the 6 different SSTs are $aa$, $ac$, $ca$, $cc$, $a\$$ and $\$$. Each SST has been connected to a central root node. The main difference of the *sparse suffix links* is that in the standard suffix tree the suffix links can point the full width of the tree. In the DST the new links point only to nodes that are within the same SST. This allows the SSTs to be constructed independently without any inter-process communication.

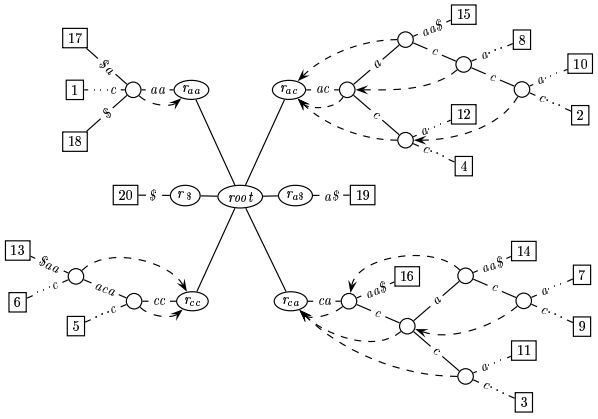


Figure 1. The SSTs for $aacacccacacaccacaaa\$$ with their respective root nodes labelled $r_{aa}, r_{ac}, r_{ca}, r_{cc}, r_{a\$}$ and $r_\$$. The sparse suffix links for the valid sets $V_{aa}, V_{ac}, V_{ca}, V_{cc}$, $V_{a\$}$ and $V_\$$ are marked with dashed arrows. Note that the final suffixes, $a\$$ and $\$$, are included but typically will not be used.

### 2.1. *Operations on the DST*

Gusfield [5] provides what can be regarded as a canonical list of major bioinformatic techniques on suffix trees. All are readily translated to a DST wihtout incurring any communication overheads. We summarise the analysis of five representative problems from this list. They are Longest Common Substring, Exact Local Matching (LCS, ELM), Maximal Repeat Finding (MRF), All Pairs Suffix-Prefix (APSP) [6] and Exact Set Matching

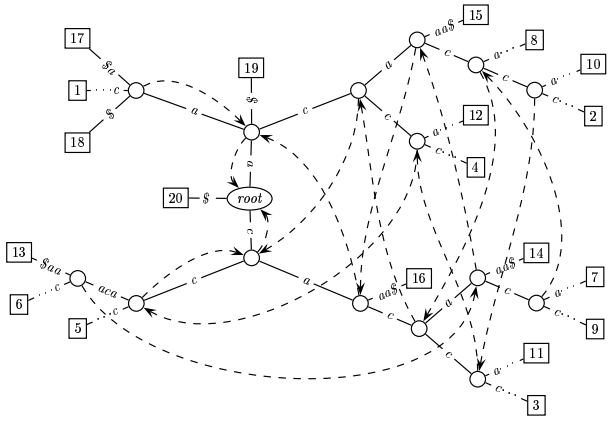


Figure 2. The standard suffix tree of *aacacccacacaccacaaa*\$ with standard suffix links. This is for comparison with the merged tree in Figure 1. See the text for further explanation.

(ESM). Full descriptions along with their serial solutions using suffix trees can be found in Gusfield [5] and elsewhere.

Because we are interested in average case and not worst case analysis we make the commonly used assumption that the input characters are independent and uniformly distributed (i.u.d.). In practice, this assumption may not hold, of course, requiring some form of load balancing for systematically biased data.

We suppose that there are $k$ computing nodes and assume for simplicity that $k = \sigma^{|z|}$, where $\sigma$ is the alphabet size and $z$ is the fixed prefix. Table 1 compares the expected running times for the solution of these five problems using the fastest serial method (based on a standard suffix tree) and a parallel method (based on distributed suffix trees).

Table 1. Post-construction average time complexities for five different problems using standard and distributed suffix trees with $k$ computing nodes. $r$ is the number of strings for the All Pairs Suffix-Prefix problem and the number of patterns for Exact Set Matching.

| Problem | Expected Running Time | |
|---|---|---|
| | Standard ST (Serial) | Distributed ST (Parallel) |
| LCS and ELM | $O(n)$ | $O(n/k)$ |
| MRF | $O(n)$ | $O(n/k)$ |
| APSP | $O(n + r^2)$ | $O((n + r^2)/k)$ |
| ESM | $O(r \log n)$ | $O((r \log n)/k)$ |

Experimental results on random data showed substantial speedups consistent with the theoretical analysis. For systematically biased biological data, we were able to devise simple load balancing schemes that increased the parallel efficiency of the operations to close to 90%. For example, we were able to increase the parallel efficiency of maximal repeat finding on human chromosome X using 16 computing nodes from 61% to 89%. Simple load balancing schemes for the other problems listed above gave similar improvements in efficiency for real biological data.

**Acknowledgements**

This work was undertaken while the first author was a PhD student at Imperial College London supported by an EPSRC studentship. We should like to thank Costas Iliopoulos and Wojtek Rytter for a number of valuable suggestions.

**References**

1. A. Apostolico. The myriad virtues of subword trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 85–96. Springer-Verlag, 1985.
2. R. Clifford and M. Sergot Distributed and Paged Suffix Trees for Large Genetic Databases. *Proc. 14th Annual Symposium on Combinatorial Pattern Matching.* 2003.
3. A. Delcher, S. Kasif, R. Fleischmann, J. Peterson, O. White, and S. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.
4. B. Dorohonceanu and C. Nevill-Manning. Accelerating protein classification using suffix trees. In *Proc. 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 126–133, 2000.
5. D. Gusfield. *Algorithms on strings, trees and sequences. Computer Science and Computational Biology.* Cambridge University Press, 1997.
6. D. Gusfield, G. M. Landau, and D. Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Information Processing Letters*, 41:181–185, 1992.
7. J. Kärkkäinen and E. Ukkonen. Sparse suffix trees. In *COCOON '96, Hong Kong*, LNCS 1090, pages 219–230. Springer-Verlag, 1996.
8. S. Kurtz and C. Schleiermacher. Reputer: Fast computation of maximal repeats in complete genomes. *Bioinformatics*, 15(5):426–427, 1999.
9. E. Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14:249–260, 1995.