Sequence analysis

\textbf{mkESA: enhanced suffix array construction tool}

Robert Homann\textsuperscript{1,2,*}, David Fleer\textsuperscript{2}, Robert Giegerich\textsuperscript{2} and Marc Rehmsmeier\textsuperscript{3,†}

\textsuperscript{1}International NRW Graduate School in Bioinformatics and Genome Research, Center for Biotechnology (CeBiTec), Bielefeld University, 33594 Bielefeld, \textsuperscript{2}Technische Fakultät, Bielefeld University, Postfach 100 131, 33501, Bielefeld, Germany and \textsuperscript{3}GMI - Gregor Mendel Institute of Molecular Plant Biology GmbH, Dr. Bohr-Gasse 3, 1030 Vienna, Austria

Received on January 21, 2009; revised on February 19, 2009; accepted on February 20, 2009

\textbf{ABSTRACT}

Summary: We introduce the tool \textit{mkESA}, an open source program for constructing enhanced suffix arrays (ESAs), striving for low memory consumption, yet high practical speed. \textit{mkESA} is a user-friendly program written in portable C99, based on a parallelized version of the Deep-Shallow suffix array construction algorithm, which is known for its high speed and small memory usage. The tool handles large FASTA files with multiple sequences, and computes suffix arrays and various additional tables, such as the LCP table (longest common prefix) or the inverse suffix array, from given sequence data.

Availability: The source code of \textit{mkESA} is freely available under the terms of the GNU General Public License (GPL) version 2 at http://bibiserv.techfak.uni-bielefeld.de/mkesa/.

Contact: rhomann@techfak.uni-bielefeld.de

\section{1 INTRODUCTION}

The program \textit{mkESA} is a software tool for constructing enhanced suffix arrays (ESAs) from biological sequence data. The ESA is an index data structure for textual data, introduced in Abouelhoda et al. (2004) as an extension of the well-known suffix array (Manber and Myers, 1993). The ESA is equivalent to the suffix tree, another very important, but more space consuming full-text index data structure (Gusfield, 1997). The major advantages of ESAs over suffix trees are their lower space overhead, improved locality of reference and simple storing to files.

A suffix array for text \(T\) of length \(n\) is a table of size \(n+1\) that lists the start positions of the suffixes of \(T\) in lexicographic order. Using a suffix array, exact string queries can be answered in \(O(m\log n)\) time, where \(m\) is the length of the query, instead of \(O(m + n)\) time without a suffix array. ESAs are composed of a suffix array and additional tables that can be used to improve query performance [e.g. \(O(m + \log n)\) time using the LCP table, called \(Hgt\) array in Manber and Myers (1993)], or enabling efficient implementation of more advanced queries (e.g. finding maximum unique matches). Thus, ESAs are fundamental technology in sequence analysis.

Many interesting problems on sequences from the field of computational biology can be solved efficiently by transforming sequence data into (enhanced) suffix arrays [see, for instance, (Beckstette et al., 2006; De Bonis et al., 2002; Krumsiek et al., 2007; Rahmann, 2003)]. Linear-time algorithms for suffix array construction have been proposed as well as algorithms that are fast in practice and/or tuned for space efficiency, rendering use of suffix arrays feasible for large datasets; see Puglisi et al. (2007) for a comprehensive overview. In addition, by the results of Abouelhoda et al. (2004), any program using suffix trees can be transformed so to employ ESAs instead and benefit from the advantages offered by that data structure.

Despite the great interest in suffix arrays in the literature, only few actual programs for ESA construction are available. Most existing programs are useful for mere suffix array construction, and do not address specifics of computational biology such as handling multiple sequences and very large datasets. A notable exception is the widely used \textit{mkvtree} program (http://www.vmatch.de/). \textit{mkvtree} can read common file formats such as FASTA and keeps sequences separated from their descriptions. An ESA generated by \textit{mkvtree} may contain multiple sequences, stored so that a match can easily be mapped to its corresponding sequence. The program is available free of charge as part of the \textit{Vmatch} package, but, unfortunately, in binary form and for non-commercial purposes only. This implies that software relying on \textit{mkvtree} cannot be distributed easily since the terms of the \textit{Vmatch} license agreement restrict the legal use of \textit{mkvtree}. Software that requires using \textit{mkvtree} also requires all users to obtain the \textit{Vmatch} package, if available for their platform of choice, and have them sign a license agreement, too.

We have implemented the alternative open source software tool \textit{mkESA}, using the Deep-Shallow algorithm (Manzini and Ferragina, 2004) for in-memory suffix array construction instead of multikey quicksort as used by \textit{mkvtree}. Thus, \textit{mkESA} is efficient even for highly repetitive sequence data, and is fast as long as all data can be held in main memory. As further improvement, our implementation of Deep-Shallow can use multiple CPUs for increased speed.

\section{2 IMPLEMENTATION}

With \textit{mkvtree} being the most widely spread program for ESA construction, we tried to pick up all of the important ideas implemented in \textit{mkvtree} and improve upon its weaknesses. \textit{mkESA} has been designed so to produce output as compatible with \textit{mkvtree} as possible. The files generated by \textit{mkESA} are in fact the same as those made by \textit{mkvtree}, meaning that data produced by \textit{mkESA} can be processed by programs that expect \textit{mkvtree}-generated ESAs.
We measured the time and space consumption for building suffix arrays. We presented our implementation of the 'Deep-Shallow' algorithm of Manzini and Ferragina (2004) for suffix array construction. This algorithm belongs to the family of 'lightweight' suffix sorting algorithms, covering algorithms that use only very small additional space besides the suffix array and the input text, i.e. only $O((5+\epsilon)n)$ bytes space for a text of length $n$, and using 32 bit integers for the skip table (Beckstette et al., 2006). It is worth noting that mkESA can incrementally add additional tables when they are needed.

### 3 RUNTIME BENCHMARKS

We compared the performance of mkESA with other programs for suffix array construction, namely mkvtree, mksary, and mkESA processed FASTA files, the other programs processed the bare sequence data with FASTA headers removed so that all programs had comparable workloads. Only 'parallel mkESA' and 'parallel mksary' (Table 2) made explicit use of multiple CPU cores. Measurements were taken on a Sun Fire X4450 (4 Intel Xeon CPUs at 2.93 GHz, 16 cores, 96 GB RAM) running Solaris 10. The programs were compiled with gcc 4.1.1 using flags -mm64 -O3 -fomit-frame-pointer. Each experiment was repeated four times in a row; the best (shortest elapsed time) of the results are displayed in Table 2. Our results show comparable memory consumption for all tested programs, while mkESA is usually the fastest among them, even when using only one CPU.

### 4 CONCLUSION

We presented mkESA, a portable, lightweight, multithreaded and fast program for constructing enhanced suffix arrays. We carefully tested the software on a variety of UNIX-like operating systems and hardware architectures, including recent versions of Linux, Solaris, Mac OS X, FreeBSD, OpenBSD and NetBSD. Its ability to generate output compatible with mkvtree makes mkESA a convenient open source drop-in replacement for earlier programs.

**Conflict of Interest:** none declared.

### REFERENCES

Abouelhoda,M. et al. (2004) Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, 2, 53–86.

Beckstette,M. (2006) Fast index based algorithms and software for matching position specific scoring matrices. *BMC Bioinformatics*, 7.

De Bonfieti,F. et al. (2008) Optimal spaced alignments of short sequence reads. *Bioinformatics*, 24, i174–i180.

Giegerich,R. et al. (2003) Efficient implementation of lavy suffix trees. *Softw. Pract. Exp.*, 33, 1035–1049.

Gusfield,D. (1997) *Algorithms on Strings, Trees, and Sequences*. Computer Science and Computational Biology. Cambridge University Press, New York, NY, USA.

HedB,M. et al. (2002) Efficient multiple genome alignment. *Bioinformatics*, 18, S312–S320.

Keimund,J. et al. (2007) Gepard: a rapid and sensitive tool for creating dotplots on genome scale. *Bioinformatics*, 23, 1026–1028.

Manber,U. and Myers,E. (1993) *Suffix Arrays: a new method for on-line string searches*. **SIAM J. Comput.**, 22, 934–948.

Manzini,G. (2004) Two space saving tricks for linear time LCP array computation. In Hagerup,T. and Katajainen,J. eds, Proceedings of 9th Scandinavian Workshop on Algorithm Theory (SWAT '06), Vol. 311 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 372–383.

Manzini,G. and Ferragina,P. (2004) Engineering a lightweight suffix array construction algorithm. *Algorithmica*, 40, 33–50.

Puglisi,S.J. et al. (2007) A taxonomy of suffix array construction algorithms. *ACM Comput. Surv.*, 39, 4.

Rahmann,S. (2003) Fast large scale oligonucleotide selection using the longest common factor approach. *J. Bioinform. Comput. Biol.*, 1, 343–361.

### Table 1. Datasets used for performance measurements

| Name  | Description                        | Size          | $\sigma$ |
|-------|------------------------------------|---------------|----------|
| chr1  | Chromosome 1 human genome          | 219 (219) MB  | 4        |
| fmdv  | Foot/mouth disease virus genomes   | 65 (64) MB    | 4        |
| trem  | UniprotKB/Prot rel. 56.4           | 181 (140) MB  | 20       |
| f25   | 25th Fibonacci string              | 2836 (2110) MB| 20       |
| f30   | 30th Fibonacci string              | 73 (73) kB    | 2        |
| spro  | UniprotKB/EMBL rel. 39.4           | 785            | 2        |
| chr1  | Chromosome 1 human genome          | 813 (813) kB  | 2        |

Size are given as file sizes, followed by sizes of files with FASTA headers removed in parentheses. Alphabet sizes are given as $\sigma$. We included Fibonacci strings since these are hard on many suffix tree and suffix array construction algorithms due to their high repetitiveness. They impose the worst case for the number of nodes in a suffix tree, $2n$, and thus, e.g. trigger the worst case running time of $O(n^2)$ of the WOTD suffix tree construction algorithm (Giegerich et al., 2003). Datasets 'fmdv' is a non-artificial example for highly repetitive sequence data, with similar impact on performance (Table 2).

### Table 2. Results of performance measurements

| Name  | mkESA | Parallel mkESA | mksary |
|-------|-------|----------------|--------|
| chr1  | 212  | 1097           | 252    |
| fmdv  | 224  | 1097           | 99     |
| trem  | 62   | 707            | 705    |
| f25   | 11   | 707            | 63     |
| f30   | 0.1  | 705            | 2.5    |
| spro  | 1.9  | 785            | 47     |
| chr1  | 224  | 1097           | 252    |
| fmdv  | 224  | 1097           | 99     |
| trem  | 62   | 707            | 705    |
| f25   | 11   | 707            | 63     |
| f30   | 0.1  | 705            | 2.5    |
| spro  | 1.9  | 785            | 47     |