BIOINFORMATICS APPLICATIONS NOTE Vol. 29 no. 6 2013, pages 802–804
doi:10.1093/bioinformatics/btt042

Sequence analysis

Advance Access publication January 24, 2013

essaMEM: finding maximal exact matches using enhanced sparse suffix arrays

Michaël Vyverman[1,*], Bernard De Baets[2], Veerle Fack[1] and Peter Dawyndt[1]

[1]Department of Applied Mathematics and Computer Science and [2]Department of Mathematical Modelling, Statistics and Bioinformatics, Ghent University, Ghent B-9000, Belgium

Associate Editor: Martin Bishop

ABSTRACT

Summary: We have developed essaMEM, a tool for finding maximal exact matches that can be used in genome comparison and read mapping. essaMEM enhances an existing sparse suffix array implementation with a sparse child array. Tests indicate that the enhanced algorithm for finding maximal exact matches is much faster, while maintaining the same memory footprint. In this way, sparse suffix arrays remain competitive with the more complex compressed suffix arrays.

Availability: Source code is freely available at https://github.ugent.be/ComputationalBiology/essaMEM.

Contact: Michael.Vyverman@UGent.be

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on November 13, 2012; revised on December 21, 2012; accepted on January 7, 2013

1 INTRODUCTION

Maximal exact matches (MEMs) are exact matches between two sequences that cannot be extended to the left or right without introducing a mismatch. In addition, extra cardinality constraints can be imposed, leading to the concepts of maximal unique matches (MUMs) and maximal almost-unique matches. These matches are widely used as seeds for sequence comparisons and alignment tools, such as MUMmer (Kurtz *et al*., 2004).

Algorithms for memory-efficient MEM-finding can be divided into online and indexed methods. Algorithms of the former type construct a (compressed) index structure for the concatenation of both sequences and iterate over the index to find the MEMs or MUMs (Hon and Sadakane, 2002). essaMEM belongs to the class of indexed MEM-finding algorithms, which match one sequence against an index of the other sequence. The advantage of indexed MEM-finding over online MEM-finding algorithms is the reusability of the constructed index. Originally, suffix trees (Kurtz *et al*., 2004) or enhanced suffix arrays (ESA) (Abouelhoda *et al*., 2004) were used to find MEMs. However, the size of these types of indexes is several times larger than the size of the indexed sequence. ESAs consist of four arrays [suffix, longest common prefix (LCP), child and suffix link arrays] that contain parts of the information stored in suffix trees and together reach the full expressiveness of suffix trees (Abouelhoda *et al*., 2004). Several algorithms build on the idea of ESAs by using more memory-efficient variants of the arrays.

Khan *et al*. (2009) suggest the use of sparse suffix arrays (SSA). SSAs index only every Kth suffix of the sequence. The parameter K is called the *sparseness factor*. Their SSA-based algorithm, *sparseMEM*, is able to find MEMs faster than previous methods, while using less memory. As a result of this lower memory footprint, SSAs can also index larger genomes than previous methods. However, for large values of K, the execution time increases dramatically.

More recently, an algorithm using (enhanced) compressed suffix arrays (CSA) was presented by Ohlebusch *et al*. (2010). In contrast to SSAs, these compressed counterparts index every Kth suffix array value. It was shown how the CSA-based MEM-finding algorithm, *backwardMEM*, outperforms sparseMEM, except when memory is abundant.

In this article, we optimize the method by Khan *et al*. by supplementing SSAs with a sparse child array for large sparseness factors. We show that the new index structure outperforms the previous design, while maintaining the same memory footprint. Furthermore, when combining both the suffix link and child arrays, we achieve a complete enhanced sparse suffix array (ESSA), which has the same expressiveness as suffix trees for substrings larger than K. We show that ESSAs are competitive for MEM-finding with the CSA-based method by Ohlebusch *et al*. and outperform commonly used methods like MUMmer (Kurtz *et al*., 2004) and Vmatch (http://www.vmatch.de/). This indicates that, although compressed index structures have recently become popular (Navarro and Mäkinen, 2007; Vyverman *et al*., 2012), the use of ESSA-based algorithms can be a viable option for further research.

2 METHODS

Although different indexed MEM-finding algorithms roughly share the same approach, the implementation of common algorithmic stages can vary greatly because of the specific design of the index structure used. As we improve on the algorithm of Khan *et al*., we only give a brief overview of the improvements and additions made to this method. A more detailed description can be found in the Supplementary Material (Section S1), and for more details on the algorithm of Khan *et al*., we refer to the original article (Khan *et al*., 2009).

We present two major improvements to the original SSA design. The first is the addition of a sparse child array to the SSA index structure of Khan *et al*. (2009). This array, as defined in the study conducted by Abouelhoda *et al*. (2004), allows constant time access to child nodes in

*To whom correspondence should be addressed.

a virtual sparse suffix tree that is simulated by the SA and LCP arrays. The sparsification of the child array is possible because child array operations only require knowledge of intervals within LCP arrays, for which the definition remains unchanged when introducing sparseness. The second improvement is the introduction of a *skip* parameter s, which introduces sparseness in the query sequence, resulting in a performance trade-off between two stages of the algorithm. When optimized, this parameter can lead to a significant increase in performance.

The first step in indexed MEM-finding algorithms consists of constructing an index structure for a reference sequence. For the construction of the sparse child array, we used the algorithm described in the study conducted by Abouelhoda *et al.* (2004).

In a second phase, all suffixes of a query sequence are matched against the index structure until a mismatch occurs, or the user-set *minimum length* L is reached. The output of this phase consists of all right maximal matches of minimum length $L-K+1$. Khan *et al.* combine a binary search algorithm for matching characters and suffix link support, which recovers computations made for the previous suffix, to increase the performance of this phase. essaMEM uses a faster matching algorithm using the sparse child array. Although suffix links can still be used with this approach, the combination of a sparse child array and suffix links does not lead to further improvements in execution time for the MEM-finding algorithm. Tests have shown that the use of the skip parameter has the same functionality as suffix link support, but it has a higher impact on the mapping time and affects a broader range of sparseness factors. The combination of child arrays and suffix links might, however, still be of interest for designing other algorithms.

The final step of the MEM-finding algorithm requires checking all right maximal matches for left maximality. As this phase is usually faster than the previous matching phase, essaMEM introduces a trade-off parameter s to increase the input of this phase and decrease the number of suffixes matched in the previous phase by a factor s. As a result, the matching phase generates all right maximal matches of minimum length $L-s\cdot K+1$.

3 RESULTS

essaMEM is open source and can be used as drop-in replacement for tools that require MEM-finding. In particular, essaMEM supports all MUMmer v3.23 options. We evaluated the performance of essaMEM against *sparseMEM*, *MUMmer*, *Vmatch* and *backwardMEM* using all relevant datasets provided previously (Khan *et al.*, 2009). The default setting of essaMEM features a sparse child array and an estimate for the optimized skip parameter, but lacks suffix links support. Because the sparse child array has the same size as the inverse suffix array (only required for suffix link support), essaMEM has the same memory footprint as the index structure used by Khan *et al.* For optimizing s, the runtime for a maximum of five successive values of s is taken. The largest value of s is set to be the largest value for which $L-s\cdot K+1\geq 10$. Timing results do not include the index construction phase, and the resident set size was measured to determine the memory footprint of the programs. The program parameters are the same as used in the study conducted by Khan *et al.* (2009) and Ohlebusch *et al.* (2010). However, we explore a much larger interval of sparseness and compression factors than previously reported by other authors.

All MEM-finding algorithms are tested on six pairs of megabase-sized genomes and two sequencing read datasets. SparseMEM and essaMEM are also tested on two pairs of gigabase-sized sequences. Figure 1 depicts the memory-time trade-offs presented by the different algorithms for finding all

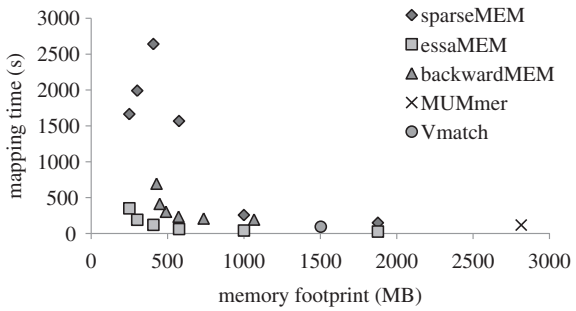

Fig. 1. Scatterplot showing the memory-time trade-offs for finding all MEMs of minimum length 50 between *D.melanogaster* (169 Mb) and *D.yakuba* (166 Mb). The data points are acquired by setting the sparseness/compression factor K to the following values (from right to left): 1, 2, 4, 8, 16 and 32. There is only one data point for MUMmer and Vmatch

MEMs between *Drosophila melanogaster* and *Drosophila yakuba*. The other results, which can be found in the Supplementary Material, depict a similar behaviour.

MUMmer and Vmatch have one of the lowest mapping times among the tested programs. Their memory requirements are, however, higher than those of the tools using compressed or sparse index structures. Furthermore, they do not allow setting a memory-time trade-off.

The tests show that sparseMEM is clearly outperformed by essaMEM and backwardMEM. Although sparseMEM is fast for small values of K, its performance steeply decreases when K is increased, which can be explained by the diminishing use of suffix links. For the largest values of K, a decrease in the runtime of the mapping phase has a positive effect on the overall runtime of sparseMEM.

A similar decrease in runtime for the mapping phase can be observed using the sparse child array in essaMEM. This is, however, countered by the diminishing effect of the skip parameter and can, therefore, not be seen in Figure 1. The lower running time for the mapping phase might be explained by a combination of the use of a sparse child array that can match more than one character at the same time, smaller minimum lengths in the matching phase and improved I/O performance. When s is increased, the runtime decreases manifold. This effect is, however, limited by the theoretical bound $s\cdot K\leq L$. The effects of the various improvements made to the original SSA-based design are discussed in the Supplementary Material.

The results also indicate that essaMEM is in general somewhat faster than backwardMEM for comparable memory settings. The difference in runtime is larger when either memory is abundant (low values of K), or the number of MEMs found is large, as backwardMEM seems to have a stronger dependency on the output size. This behaviour can be seen in Figure 1, where the runtime of backwardMEM steeply increases for $K>8$. In contrast to SSA-based methods, however, backwardMEM puts no restriction on the maximum value of K and could thus be used for small L settings in combination with a high compression factor.

In terms of memory consumption, backwardMEM starts with a lower memory footprint at no compression, but the memory footprint of essaMEM decreases faster. Theoretically, the CSA

Table 1. The real size of the index structure built by sparseMEM, essaMEM and backwardMEM for the *D.melanogaster* (169 Mb) genome

K	sparseMEM (Mb)	essaMEM (Mb)	backwardMEM (Mb)
1	1861	1861	1031
2	997	997	709
4	576	576	548
8	370	370	468
16	268	268	427
32	218	218	407

sparseMEM and essaMEM share the same memory footprint.

index requires $4/K + 1.375$ bytes per input character, and the ESSA index requires $9/K + 1$ bytes per input character. As a result, both indexes theoretically have the same memory requirements for $K \approx 13$. The actual allocated memory for the sparse or compressed index structures for the *D.melanogaster* genome is given in Table 1. For this dataset, the index size of the MEM-finding tools is equal for K between 4 and 8.

ACKNOWLEDGEMENTS

All authors acknowledge the support of Ghent University (MRP Bioinformatics: from nucleotides to networks). The computational resources and services used in this work were kindly provided by Ghent University, the Flemish Supercomputer Center.

Funding: Agency for Innovation by Science and Technology of the Flemish government (SB-101609 to M.V.).

Conflict of Interest: none declared.

REFERENCES

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

Hon,W. and Sadakane,K. (2002) Space-economical algorithms for finding maximal unique matches. In: *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, pp. 144–152.

Khan,Z. *et al.* (2009) A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays. *Bioinformatics*, **25**, 1609–1616.

Kurtz,S. *et al.* (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.

Navarro,G. and Mäkinen,V. (2007) Compressed full-text indexes. *ACM Comput. Surv.*, **39**, Article 2.

Ohlebusch,E. *et al.* (2010) Computing matching statistics and maximal exact matches on compressed full-text indexes. In: *Proceedings of the 17th Annual Symposium on String Processing and Information Retrieval*. Springer, pp. 347–358.

Vyverman,M. *et al.* (2012) Prospects and limitations of full-text index structures in genome analysis. *Nucleic Acids Res.*, **40**, 6993–7015.