*Bioinformatics*
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Applications Note

Subject Section

# emMAW: Computing Minimal Absent Words in External Memory

**Alice Heliou [1,*], Solon P. Pissis [2,*] and Simon J. Puglisi [3]**

[1] Inria Saclay & Laboratoire d'Informatique de l'École Polytechnique (LIX), CNRS UMR 7161, France
[2] Department of Informatics, King's College London, London WC2R 2LS, UK and
[3] Department of Computer Science, University of Helsinki, Helsinki FI-00014, Finland

* To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

### Abstract

**Motivation:** The biological significance of minimal absent words has been investigated in genomes of organisms from all domains of life. For instance, three minimal absent words of the human genome were found in Ebola virus genomes (Silva et al., Bioinf., 2015). There exists an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all minimal absent words of a sequence of length $n$ on a fixed-sized alphabet based on suffix arrays (Barton et al., BMC Bioinf., 2014). A standard implementation of this algorithm, when applied to a large sequence of length $n$, requires more than $20n$ bytes of RAM. Such memory requirements are a significant hurdle to the computation of minimal absent words in large data sets.
**Results:** We present emMAW, the first external-memory algorithm for computing minimal absent words. A free open-source implementation of our algorithm is made available. This allows for computation of minimal absent words on far bigger data sets than was previously possible. Our implementation requires less than 3 hours on a standard workstation to process the full human genome when as little as 1 GB of RAM is made available. We stress that our implementation, despite making use of external memory, is fast; indeed, even on relatively smaller data sets when enough RAM is available to hold all necessary data structures, it is less than two times slower than state-of-the-art internal-memory implementations.
**Availability:** https://github.com/solonas13/maw (Free software under the terms of the GNU GPL)
**Contact:** alice.heliou@lix.polytechnique.fr, solon.pissis@kcl.ac.uk
**Supplementary information:** Supplementary data are available at *Bioinformatics* online.

## 1 Introduction

Computational methods for the study and detection of *absent* or *avoided* words in genomic sequences have received much attention recently (Almirantis *et al.*, 2017; Belazzougui and Cunial, 2015).

From a combinatorial perspective, given a sequence $y$ of length $n$, an *absent word* is any word that does not occur as a factor (subword) of $y$. The number of absent words (of length at most $n$) is exponential in $n$. However, the number of certain classes (subsets) of these words is only linear in $n$. This is the case for *minimal absent words*; that is, words absent from $y$ whose all proper factors occur in $y$ (Béal *et al.*, 2000). An upper bound on the number of minimal absent words is known to be $\mathcal{O}(\sigma n)$ (Crochemore *et al.*, 1998), where $\sigma$ is the alphabet's size. This bound is asymptotically tight (Almirantis *et al.*, 2017; Mignosi *et al.*, 2002).

From a biological perspective, absent or avoided words may represent a spectrum of information. They can be hardly-tolerated nucleotide sequences because their structure influences negatively the stability of the chromatin or other functional genomic conformation; they can represent targets of restriction endonucleases; or, more generally, their presence in wide parts of the genome may be hardly tolerated for less known reasons (Almirantis *et al.*, 2017). There have been many studies on the biological significance of such words (Hampikian and Andersen, 2007; Silva *et al.*, 2015; Almirantis *et al.*, 2017).

On the algorithmic side, an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing minimal absent words (on a fixed-sized alphabet) based on automata is known for some time (Crochemore *et al.*, 1998). More recently, computation of minimal absent words using more space-efficient data structures, such as the Burrows-Wheeler transform (Belazzougui *et al.*, 2013) or suffix arrays (Barton *et al.*, 2014), has been considered;

© The Author 2015. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

and a few $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space suffix-array-based algorithms are known (Barton *et al.*, 2014, 2015). Of these algorithms, an implementation of the algorithm presented in (Barton *et al.*, 2014) is currently, and to the best of our knowledge, the fastest available for computing minimal absent words. The advantage of suffix-array-based algorithms for computing minimal absent words is that they are very fast in practice and they are more space-efficient than tree/automata-based algorithms. However, the internal memory requirements of these algorithms, when applied to large data sets, make computation impossible without using large-scale computer clusters.

**Our Contributions.** We present emMAW, the *first* external-memory algorithm for computing minimal absent words. A free open-source implementation of our algorithm is made available. This allows for computation of minimal absent words on far bigger data sets than was previously possible on commodity desktop computers. We also provide here benchmark results using mainly real data. Specifically, we show that our implementation requires less than 3 hours to process the full (forward and reverse complement) human genome when as little as 1 GB of RAM is made available. Note that the state-of-the-art implementation of (Barton *et al.*, 2014) requires more than 140 GB of RAM for the same assignment. Even on relatively smaller data sets when enough RAM is available to hold all necessary data structures, we show that our new implementation is still competitive with state-of-the-art internal-memory implementations.

## 2 Methods

Let $y = y[0]y[1] \mathinner{.\,.} y[n-1]$ be a *word* of *length* $n = |y|$ on a finite ordered *alphabet* $\Sigma$ of size $\sigma = |\Sigma| = \mathcal{O}(1)$. For two positions $i$ and $j$ on $y$, we denote by $y[i \mathinner{.\,.} j] = y[i] \mathinner{.\,.} y[j]$ the *factor* of $y$ that starts at position $i$ and ends at position $j$. A *suffix* is a factor that ends at position $n-1$, and a *proper* factor is a factor different from $y$ itself. Let $x$ be a word of length $0 < m \leq n$. We say that $x$ *occurs* at the *starting position* $i$ in $y$ when $x = y[i \mathinner{.\,.} i+m-1]$; and that $x$ is an *absent word* of $y$ if it does not occur in $y$. The absent word $x$ of $y$ is *minimal* iff all its proper factors occur in $y$. A *repeated pair* $R$ in $y$ is a triple $(i, j, w)$ such that $i$ and $j$ are starting positions of word $w$ in $y$. Moreover we have that: $R$ is *left maximal* iff $y[i-1] \neq y[j-1]$; $R$ is *right maximal* iff $y[i+|w|] \neq y[j+|w|]$; $R$ is *maximal* iff it is left maximal and right maximal. We denote by SA the *suffix array* of $y$ of length $n$, that is, an array of size $n$ storing the starting positions of all (lexicographically) sorted suffixes of $y$, i.e. for all $1 \leq r < n$, we have $y[\text{SA}[r-1] \mathinner{.\,.} n-1] < y[\text{SA}[r] \mathinner{.\,.} n-1]$. Its inverse bijection is denoted by iSA. The *Burrows-Wheeler transform*, denoted by BWT, is defined by $\text{BWT}[i] = y[\text{SA}[i]-1]$, unless $\text{SA}[i] = 0$, in which case $\text{BWT}[i] = \#$, where $\#$ is a letter not from $\Sigma$. Let $\text{lcp}(r, s)$ denote the length of the longest common prefix between $y[\text{SA}[r] \mathinner{.\,.} n-1]$ and $y[\text{SA}[s] \mathinner{.\,.} n-1]$, for all positions $r$, $s$ on $y$, and 0 otherwise. We denote by LCP the *longest common prefix* array of $y$ defined by $\text{LCP}[r] = \text{lcp}(r-1, r)$, for all $1 \leq r < n$, and $\text{LCP}[0] = 0$.

We analyse the proposed algorithm in the external memory (EM) model of computation; see (Vitter, 2006) for details. By $M$ we denote the RAM (internal memory) size and by $B$ the disk (external memory) block size, both measured in units of $\Theta(\log n)$-bit words. We further assume that $M = \Omega(\log n)$ and $M = \mathcal{O}(n)$. In the EM model, each transfer of $B$ words between internal and external memory is called an IO, and, hence, an algorithm's complexity is mainly measured in IOs; see, for instance, Kärkkäinen *et al.* (2017) for constructing SA in the EM model.

**Lemma** 2.1. *Let $(i, j, w)$ be a right maximal repeated pair of a word $y$. There exist $0 \leq k < \ell < |y|$, $y[i \mathinner{.\,.} i+|w|] = y[\text{SA}[k] \mathinner{.\,.} \text{SA}[k] + \text{LCP}[k+1]]$ and $y[j \mathinner{.\,.} j+|w|] = y[\text{SA}[\ell] \mathinner{.\,.} \text{SA}[\ell] + \text{LCP}[\ell]]$.*

Proof. Without loss of generality we consider that $\text{iSA}[i] < \text{iSA}[j]$. We have $|w| = \text{lcp}(\text{iSA}[i], \text{iSA}[j])$, thus there exist $m \in (\text{iSA}[i], \text{iSA}[j]]$, such that $|w| = \text{LCP}[m]$. We denote by $\ell$ the largest of these indices and by $k+1$ the smallest; they can be equal. Thus $\text{lcp}(\text{iSA}[i], k) > |w|$ and $\text{lcp}(\ell, \text{iSA}[j]) > |w|$, consequently the equalities hold. □

By Lemma 2.1, we can focus on the following $2n$ factors: $F_{2i} = y[\text{SA}[i] \mathinner{.\,.} \text{SA}[i] + \text{LCP}[i]]$, with $i \in [0 : n-1]$; $F_{2i+1} = y[\text{SA}[i] \mathinner{.\,.} \text{SA}[i] + \text{LCP}[i+1]]$, with $i \in [0 : n-1]$. For each $F_j$, with $j \in [0 : 2n-1]$, we denote by: $B_1[j]$ the set of letters that occur right before the occurrences of $F_j$; $B_2[j]$ the set of letters that occur right before the occurrences of the longest proper prefix of $F_j$.

**Lemma** 2.2 ((Barton *et al.*, 2014)). *$awb$ is a minimal absent word of $y$, with $a, b \in \Sigma$ and $w$ a word, iff there exists $j$ such that $a \in B_2[j] \setminus B_1[j]$ and $wb = F_j$.*

Next, we provide the details of our algorithm for computing minimal absent words in external memory, which we denote by emMAW.

**Pre-processing: Computing** SA**,** LCP**, and** BWT**.** In our implementation we make use of the pSAscan algorithm due to Kärkkäinen *et al.* (2015) to compute SA and the Sparse-$\Phi$ algorithm due to Kärkkäinen and Kempa (2016) to compute LCP in external memory. For computing BWT we use the following easy-to-implement method. If the RAM is not enough for the word to fit inside, we compute BWT block by block. We store in memory $m$ pairs of the form $(i, \text{SA}[i])$ such that they fit in RAM, and we sort the pairs with respect to the $\text{SA}[i]$ field. Then we scan $y$ and the list of sorted pairs. During the scan, we replace the $\text{SA}[i]$ field of each pair with letter $y[\text{SA}[i]-1]$ (except if $\text{SA}[i] = 0$, in which case we replace it with a letter $\#$ not from $\Sigma$). Finally, we sort the pairs with respect to the $i$ field. The letters are a contiguous segment of BWT, and so we store them. We repeat the process, until we have the whole BWT.

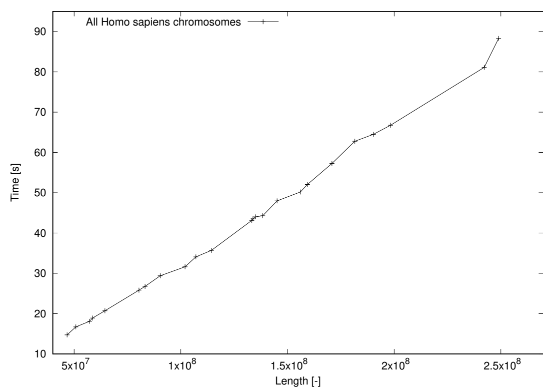
**Stage 1: Computing sets $B_1[j]$ and $B_2[j]$.** Given the word $y$ and its SA and LCP in internal memory, computing sets $B_1[j]$ and $B_2[j]$ can be done in internal memory in time and space $\mathcal{O}(n)$ (Barton *et al.*, 2014). Here we adapt this algorithm to compute sets $B_1[j]$ and $B_2[j]$ in external memory when having SA, LCP, and BWT precomputed and stored in external memory. The main difference is that we do not use the word $y$ itself but rather its BWT. To compute the sets $B_1[j]$ and $B_2[j]$, we scan SA, LCP, and BWT twice: top-down and bottom-up. These data structures are always accessed sequentially. Thus we can store them in external memory and then scan or modify them by transferring in RAM only a segment of entries, whose number is proportional to $M$. Transferring $n$ words from or to external memory requires time $\mathcal{O}(n)$ with $\mathcal{O}(\frac{n}{B})$ IOs (Vitter, 2006).

**Stage 2: Computing the set of minimal absent words.** At this point we have stored in external memory the sets $B_1[j]$ and $B_2[j]$ for all $j \in [0 : 2n-1]$. By applying Lemma 2.2 we can obtain all minimal absent words of $y$ by computing the difference $B_2[j] \setminus B_1[j]$ for all $j \in [0 : 2n-1]$.

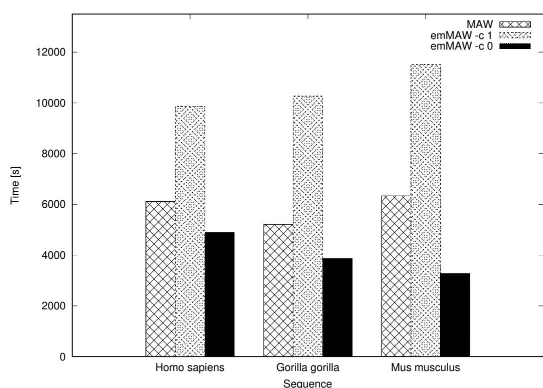
Theorem 2.3. *Given a word of length $n$ and its* SA*,* LCP*, and* BWT *in external memory, algorithm* emMAW *computes all minimal absent words in time $\mathcal{O}(n)$, with $\mathcal{O}(\frac{n}{B})$ IOs, and using $\mathcal{O}(n)$ space in external memory.*

## 3 Results

We implemented algorithm emMAW as a program to compute all minimal absent words of a given sequence. The program was implemented in the C programming language. It is available at `http://github.com/solonas13/maw` under the GNU GPL terms. We used the following two machines in order to evaluate our implementation. The first one, denoted by **M1**, is a desktop PC with $1 \times 8$ cores of Intel(R) Core(TM) i7-4790 CPU at 3.60GHz with 8M Cache and 16GB of DDR3 RAM under 64-bit GNU/Linux. **M1** is equipped with a single SSD disk with capacity 256GB. The second one, denoted by **M2**, is a single node of a cluster computer with $2 \times 10$ cores of Intel(R) Xeon(R) CPU E5-2660 v3 at 2.60GHz with 25M Cache and 384GB of DDR3 RAM under 64-bit GNU/Linux. **M2** is equipped with a disk array with HDD disks with total capacity 524TB.




(a) Elapsed time of emMAW in external memory




(b) Elapsed time of MAW and emMAW in internal memory

**Fig. 1.** Computing minimal absent words in internal and external memory

*External memory*. Our first task was to validate our theoretical findings (Theorem 2.3). To this end, we used as input all chromosome sequences of the *Homo sapiens* genome obtained from the NCBI database (`ftp://ftp.ncbi.nih.gov/genomes/`). We computed all minimal absent words of length at most 11 for each sequence separately. We considered only the $5' \rightarrow 3'$ DNA strand. We had first pre-computed and stored in external memory the necessary data structures. This set of runs was conducted on **M1**. Fig. 1(a) depicts elapsed-time measurements of emMAW (without accounting for the time to construct the data structures) using only 500 MiB of internal memory. The results confirm our theoretical findings: the elapsed time increases linearly with the length of the input sequence. To further evaluate the efficiency of our implementation, we used as input the full genome of *Homo sapiens* without pre-computing the necessary data structures. We computed all minimal absent words of length at most 11 using only $1,000$ MiB of internal memory. We considered both DNA strands. The whole assignment (accounting for the time to construct the data structures) took $9,219$ seconds on **M1** and $9,282$ on **M2**.

*Internal memory*. We next compared the efficiency of emMAW against the corresponding one of MAW (Barton *et al.*, 2014), the fastest internal-memory implementation, when both exclusively use internal memory (150,000 MiB) for their computations. We considered the full genomes of *Homo sapiens*, *Gorilla gorilla*, and *Mus musculus* genomes, obtained from the NCBI database. We computed all minimal absent words of length at most 11 for each sequence. We considered both DNA strands. For this set of runs, we used **M2** to ensure that the necessary data structures can be constructed and stored in internal memory. We used two options for emMAW: (i) `-c 1` denoting that the necessary data structures must be constructed; (ii) `-c 0` denoting that they have already been pre-computed and can be read from disk. Elapsed-time comparisons are illustrated in Fig. 1(b). The results show that emMAW is less than two times slower than MAW with `-c 1`; most importantly, we see that emMAW becomes faster than MAW with `-c 0`.

## 4 Conclusion

We presented algorithm emMAW, the first external-memory algorithm for computing minimal absent words. Given a sequence of length $n$ and its SA, LCP, and BWT in external memory, emMAW computes all minimal absent words in time $\mathcal{O}(n)$, with $\mathcal{O}(\frac{n}{B})$ IOs, and using $\mathcal{O}(n)$ space in external memory. We also made available an open-source implementation of emMAW. We provided benchmark results showing that our implementation requires less than 3 hours on a standard workstation to process the full human genome when as little as 1 GB of RAM is made available. Our implementation, despite making use of external memory, is fast; indeed, even on relatively smaller data sets when enough RAM is available to hold all necessary data structures, it is less than two times slower than state-of-the-art internal-memory implementations.

## Funding

This work was partially supported by the Academy of Finland via grants 2845984 and 294143.

## References

Almirantis, Y., Charalampopoulos, P., Gao, J., Iliopoulos, C. S., Mohamed, M., Pissis, S. P., and Polychronopoulos, D. (2017). On avoided words, absent words, and their application to biological sequence analysis. *Algorithms for Molecular Biology*, **12**(1), 5.

Barton, C., Heliou, A., Mouchard, L., and Pissis, S. P. (2014). Linear-time computation of minimal absent words using suffix array. *BMC Bioinformatics*, **15**, 388.

Barton, C., Heliou, A., Mouchard, L., and Pissis, S. P. (2015). Parallelising the computation of minimal absent words. In *PPAM, Part II*, volume 9574 of *LNCS*, pages 243–253. Springer.

Béal, M., Mignosi, F., Restivo, A., and Sciortino, M. (2000). Forbidden words in symbolic dynamics. *Advances in Applied Mathematics*, **25**(2), 163–193.

Belazzougui, D. and Cunial, F. (2015). Space-efficient detection of unusual words. In *SPIRE*, volume 9309 of *LNCS*, pages 222–233. Springer.

Belazzougui, D., Cunial, F., Kärkkäinen, J., and Mäkinen, V. (2013). Versatile succinct representations of the bidirectional Burrows–Wheeler transform. In *ESA*, volume 8125 of *LNCS*, pages 133–144. Springer.

Crochemore, M., Mignosi, F., and Restivo, A. (1998). Automata and forbidden words. *Information Processing Letters*, **67**, 111–117.

Hampikian, G. and Andersen, T. (2007). Absent sequences: Nullomers and primes. In *PCB*, pages 355–366. World Scientific.

Kärkkäinen, J. and Kempa, D. (2016). Faster external memory LCP array construction. In *ESA 2016*, volume 57 of *LIPIcs*, pages 61:1–61:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Kärkkäinen, J., Kempa, D., and Puglisi, S. J. (2015). Parallel external memory suffix sorting. In *CPM*, volume 9133 of *LNCS*, pages 329–342. Springer.

Kärkkäinen, J., Kempa, D., Puglisi, S. J., and Zhukova, B. (2017). Engineering external memory induced suffix sorting. In *ALENEX*, pages 98–108. SIAM.

Mignosi, F., Restivo, A., and Sciortino, M. (2002). Words and forbidden factors. *Theoretical Computer Science*, **273**(1-2), 99–117.

Silva, R. M., Pratas, D., Castro, L., Pinho, A. J., and Ferreira, P. J. S. G. (2015). Three minimal sequences found in Ebola virus genomes and absent from human DNA. *Bioinformatics*, **31**(15), 2421–2425.

Vitter, J. S. (2006). Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, **2**(4), 305–474.