Constructing Antidictionaries in Output-Sensitive Space

Lorraine Ayad Golnaz Badkobeh Gabriele Fici
Alice Héliou Solon Pissis

LSD/LAW 2019

London, UK, 7-8 Feb. 2019

Minimal Absent Words

Definition

A word v is an absent word of some word w if v does not occur as a factor in w.

An absent word is minimal if all its proper factors occur in the word w.

Minimal Absent Words

Definition

A word v is an absent word of some word w if v does not occur as a factor in w.

An absent word is minimal if all its proper factors occur in the word w.

Example

Let $w = abaab$. The minimal absent words (MAWs) for w are:

$$\mathcal{M}_w = \{aaa, aaba, bab, bb\}$$

Minimal Absent Words

Definition

A word v is an absent word of some word w if v does not occur as a factor in w.

An absent word is minimal if all its proper factors occur in the word w.

Example

Let $w = abaab$. The minimal absent words (MAWs) for w are:

$$\mathcal{M}_w = \{aaa, aaba, bab, bb\}$$

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Applications of Minimal Absent Words

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Applications of Minimal Absent Words

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Most of the times, a reduced antidictionary $\mathcal{M}^{\ell}$ is considered, consisting of those MAWs whose length is bounded by some threshold ℓ.

Properties of Minimal Absent Words

The theory of MAWs is well developed. For example, it is know that:

Theorem

1. *A word of length n has $\mathcal{O}(n)$ different MAWs, which can be stored occupying $\mathcal{O}(n)$ total space.*
2. *One can compute the antidictionary of a word of length n in $\mathcal{O}(n)$ time and space.*
3. *Any word of length n can be reconstructed in $\mathcal{O}(n)$ time and space from its (complete) antidictionary.*
4. *The maximal length of a MAW equals 2 + the maximal length of a repeated factor. Thus, for a random[a] word of length n, the longest MAW has length $\Theta(\log_{|\Sigma|} n)$.*

[a] generated by a Bernoulli i.i.d. source

Algorithms for Computing Minimal Absent Words

There exist several efficient algorithms for computing the (reduced) antidictionary of a word of length n, e.g.:

- $O(n)$ time and space using a global data structure (e.g., SA) [Barton, Héliou, Mouchard, Pissis, 2014]

  — can be executed in external memory [Héliou, Pissis, Puglisi, 2017]

- $O(n) + |\mathcal{M}^\ell|$ time using $O(\min\{n, \ell z\})$ space, where z is the size of the LZ77 factorization, using the truncated DAWG [Fujishige, Takuya, Diptarama, 2018]

Algorithms for Computing Minimal Absent Words

There exist several efficient algorithms for computing the (reduced) antidictionary of a word of length n, e.g.:

- $O(n)$ time and space using a global data structure (e.g., SA) [Barton, Héliou, Mouchard, Pissis, 2014]

  — can be executed in external memory [Héliou, Pissis, Puglisi, 2017]

- $O(n) + |\mathcal{M}^\ell|$ time using $O(\min\{n, \ell z\})$ space, where z is the size of the LZ77 factorization, using the truncated DAWG [Fujishige, Takuya, Diptarama, 2018]

However, all these algorithms require $\Omega(n)$ space due to the construction of a global data structure on the input word.

Number and Distribution of Minimal Absent Words

The total number and the distribution of lengths of MAWs has been studied for several sequences.

Number and Distribution of Minimal Absent Words

The total number and the distribution of lengths of MAWs has been studied for several sequences.

Example

In the human genome ($n \approx 3 \times 10^9$) we have $||\mathcal{M}^{12} \approx 10^6|| = o(n)$ (while $||\mathcal{M}^{10}|| = 0$).

Number and Distribution of Minimal Absent Words

The total number and the distribution of lengths of MAWs has been studied for several sequences.

Example

In the human genome ($n \approx 3 \times 10^9$) we have $||\mathcal{M}^{12} \approx 10^6|| = o(n)$ (while $||\mathcal{M}^{10}|| = 0$).

Problem

Compute the (reduced) antidictionary in output-sensitive space.

Strategy

Idea:

- Divide the input word y into k words each of which, alone, fits in the internal memory, with a suitable overlap of length ℓ so as not to lose information.

$$y = y_1 \# y_2 \# \cdots \# y_k, \qquad \# \notin \Sigma$$

Strategy

Idea:

- Divide the input word y into k words each of which, alone, fits in the internal memory, with a suitable overlap of length ℓ so as not to lose information.

$$y = y_1 \# y_2 \# \cdots \# y_k, \qquad \# \notin \Sigma$$

- Then compute the MAWs of the input word y incrementally, from the MAWs of the concatenation of these k words.

Strategy

Idea:

- Divide the input word y into k words each of which, alone, fits in the internal memory, with a suitable overlap of length ℓ so as not to lose information.

$$y = y_1 \# y_2 \# \cdots \# y_k, \qquad \# \notin \Sigma$$

- Then compute the MAWs of the input word y incrementally, from the MAWs of the concatenation of these k words.

Formally, we state the following

Problem

Given k words $y_1, y_2, \ldots, y_k$ over an alphabet Σ and an integer $\ell > 0$, compute the set $\mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$ of minimal absent words of length at most ℓ of $y = y_1\#y_2\#\ldots\#y_k$, $\# \notin \Sigma$.

Theoretical Results

Here is an illustration of the theoretical setting: Let $y = y_1 \# y_2$.

We are allowed to store in internal memory y_1 and y_2 but not y.

Our goal is to compute $\mathcal{M}_y^\ell$ from $\mathcal{M}_{y_1}^\ell$ and $\mathcal{M}_{y_2}^\ell$.

Theoretical Results

Here is an illustration of the theoretical setting: Let $y = y_1 \# y_2$.

We are allowed to store in internal memory y_1 and y_2 but not y.

Our goal is to compute $\mathcal{M}_y^\ell$ from $\mathcal{M}_{y_1}^\ell$ and $\mathcal{M}_{y_2}^\ell$.

Let $x \in \mathcal{M}_y^\ell$. We separate two cases:

1. x belongs to $\mathcal{M}_{y_1}^\ell \cup \mathcal{M}_{y_2}^\ell$ (Case 1)
2. x does not belong to $\mathcal{M}_{y_1}^\ell \cup \mathcal{M}_{y_2}^\ell$ (Case 2)

Theoretical Results

Lemma (Case 1)

A word $x \in \mathcal{M}^{\ell}_{y_1}$ (resp. $x \in \mathcal{M}^{\ell}_{y_2}$) belongs to $\mathcal{M}^{\ell}_{y}$ if and only if x is a superword of a word in $\mathcal{M}^{\ell}_{y_2}$ (resp. in $\mathcal{M}^{\ell}_{y_1}$).

Theoretical Results

Lemma (Case 1)

A word $x \in \mathcal{M}^{\ell}_{y_1}$ (resp. $x \in \mathcal{M}^{\ell}_{y_2}$) belongs to $\mathcal{M}^{\ell}_{y}$ if and only if x is a superword of a word in $\mathcal{M}^{\ell}_{y_2}$ (resp. in $\mathcal{M}^{\ell}_{y_1}$).

Example

Let $y_1 = \text{abaab}$, $y_2 = \text{bbaaab}$ and $\ell = 5$. $y = \text{abaab\#bbaaab}$. We have $\mathcal{M}^{\ell}_{y_1} = \{\texttt{bb,aaa,bab,aaba}\}$ and
$\mathcal{M}^{\ell}_{y_2} = \{\texttt{bbb,aaaa,baab,aba,bab,abb}\}$.
The word bab is contained in $\mathcal{M}^{\ell}_{y_1} \cap \mathcal{M}^{\ell}_{y_2}$ so it belongs to $\mathcal{M}^{\ell}_{y}$. The word aaba $\in \mathcal{M}^{\ell}_{y_1}$ is a superword of aba $\in \mathcal{M}^{\ell}_{y_2}$ hence aaba $\in \mathcal{M}^{\ell}_{y}$. On the other hand, the words bbb, aaaa and abb are superwords of words in $\mathcal{M}^{\ell}_{y_1}$, hence they belong to $\mathcal{M}^{\ell}_{y}$. The remaining MAWs are not superwords of MAWs of the other word.

$$\mathcal{M}^{\ell}_{y} \cap (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}) = \{\texttt{aaaa,bab,aaba,abb,bbb}\}.$$

Theoretical Results

We define the *reduced sets* of MAWs, $\mathcal{R}^{\ell}_{y_i}$, as those sets obtained from $\mathcal{M}^{\ell}_{y_i}$ after removing those words that are superwords of a word in $\mathcal{M}^{\ell}_{y_j}$, $\{i, j\} = \{1, 2\}$.

We define the *reduced sets* of MAWs, $\mathcal{R}^{\ell}_{y_i}$, as those sets obtained from $\mathcal{M}^{\ell}_{y_i}$ after removing those words that are superwords of a word in $\mathcal{M}^{\ell}_{y_j}$, $\{i, j\} = \{1, 2\}$.

Lemma (Case 2)

Let $x \in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$. *Then* x *has a prefix* x_i *in* $\mathcal{R}^{\ell}_{y_i}$ *and a suffix* x_j *in* $\mathcal{R}^{\ell}_{y_j}$, *for* i, j *such that* $\{i, j\} = \{1, 2\}$.

Theoretical Results

We define the *reduced sets* of MAWs, $\mathcal{R}^{\ell}_{y_i}$, as those sets obtained from $\mathcal{M}^{\ell}_{y_i}$ after removing those words that are superwords of a word in $\mathcal{M}^{\ell}_{y_j}$, $\{i, j\} = \{1, 2\}$.

Lemma (Case 2)

Let $x \in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$. Then x has a prefix x_i in $\mathcal{R}^{\ell}_{y_i}$ and a suffix x_j in $\mathcal{R}^{\ell}_{y_j}$, for i, j such that $\{i, j\} = \{1, 2\}$.

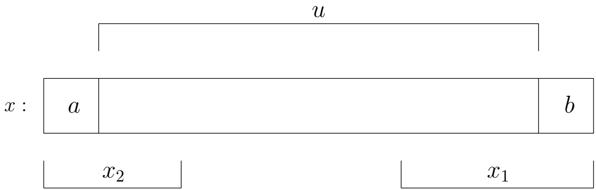
Example

Let $y_1 =$ abaab and $y_2 =$ bbaaab. $y =$ abaab#bbaaab. We have $\mathcal{R}^{\ell}_{y_1} = \{bb,aaa\}$ and $\mathcal{R}^{\ell}_{y_2} = \{$baab,aba$\}$.

Consider $x =$ abaaa $\in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$ (Case 2 MAW).

There is a MAW$x_2 \in \mathcal{R}^{\ell}_{y_2}$ that is a prefix of abaa and this is aba. Analogously, there is an $x_1 \in \mathcal{R}^{\ell}_{y_1}$ that is a suffix of abaaa and this is aaa.

Theoretical Results



Example

Let $y_1 =$ abaab and $y_2 =$ bbaaab. $y =$ abaab#bbaaab. We have $\mathcal{R}^{\ell}_{y_1} = \{bb,aaa\}$ and $\mathcal{R}^{\ell}_{y_2} = \{$baab,aba$\}$.

Consider $x =$ abaaa $\in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$ (Case 2 MAW).

There is a MAW$x_2 \in \mathcal{R}^{\ell}_{y_2}$ that is a prefix of abaa and this is aba. Analogously, there is an $x_1 \in \mathcal{R}^{\ell}_{y_1}$ that is a suffix of abaaa and this is aaa.

Theoretical Results

We come to the following general result, which is the theoretical basis of our algorithm:

Theorem

Let $N > 1$, and let $x \in \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N}$. Then, either $x \in \mathcal{M}^{\ell}_{y_1\#\ldots\#y_{N-1}} \cup \mathcal{M}^{\ell}_{y_N}$ (Case 1 MAWs) or, otherwise, $x \in \mathcal{M}^{\ell}_{y_i\#y_N} \setminus (\mathcal{M}^{\ell}_{y_i} \cup \mathcal{M}^{\ell}_{y_N})$ for some i. Moreover, in this latter case, x has a prefix in $\mathcal{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ and a suffix in $\mathcal{R}^{\ell}_{y_N}$, or the converse, i.e., x has a prefix in $\mathcal{R}^{\ell}_{y_N}$ and a suffix in $\mathcal{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ (Case 2 MAWs).

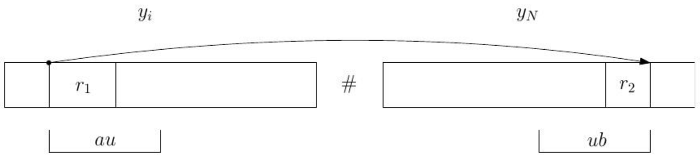
The Algorithm

At the Nth step, we have in memory the set $\mathcal{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$. Our algorithm works as follows:

1. We read word y_N from the disk and compute $\mathcal{M}^{\ell}_{y_N}$ in time $\mathcal{O}(|y_N|)$.
2. We compute Case 1 MAWs using the first Lemma.
3. For every $i \in [1, N-1]$, we perform the following to compute Case 2 MAWs:

The Algorithm

1. Read word y_i from the disk. Construct the suffix tree T_x of word $x = y_i \# y_N$ in time $\mathcal{O}(|y_i| + |y_N|)$. Use T_x to locate all occurrences of elements of $\mathcal{R}^{\ell}_{y_N}$ in y_i.
2. Compute the set $\mathcal{M}^{\ell}_{y_i \# y_N}$ and output the words.
3. Suppose au occurs in y_i and ub in y_N. Check whether au starts where a word r_1 of $\mathcal{R}^{\ell}_{y_N}$ starts and ub ends where a word r_2 of $\mathcal{R}^{\ell}_{y_1 \# \ldots \# y_{N-1}}$ ends. If this is the case and $|u| \geq \max\{|r_1|, |r_2|\} - 1$, then aub is added to our output set M, otherwise discard it. The case when au occurs in y_N and ub in y_i is treated analogously.



The Algorithm

Let $\textsc{MaxIn}$ be the length of the longest word in $\{y_1, \ldots, y_k\}$ and $\textsc{MaxOut} = \max\{|| \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N} || : N \in [1, k]\}$.

Theorem

Given k words $y_1, y_2, \ldots, y_k$ and an integer $\ell > 0$, all $\mathcal{M}^{\ell}_{y_1}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^{k} || \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N} ||)$ total time using $\mathcal{O}(\textsc{MaxIn} + \textsc{MaxOut})$ space, where $n = |y_1\# \ldots \#y_k|$.

Proof-of-Concept Experiments

The algorithm has been implemented in the `C++` programming language.
(The implementation can be made available upon request.)

Proof-of-Concept Experiments

The algorithm has been implemented in the `C++` programming language. (The implementation can be made available upon request.)

As input dataset here we used the entire human genome (version hg38), which has an approximate size of 3.1GB. The experiments were conducted on a machine with an Intel Core i5-4690 CPU at 3.50 GHz and 128GB of memory running GNU/Linux.

Proof-of-Concept Experiments

The algorithm has been implemented in the `C++` programming language. (The implementation can be made available upon request.)

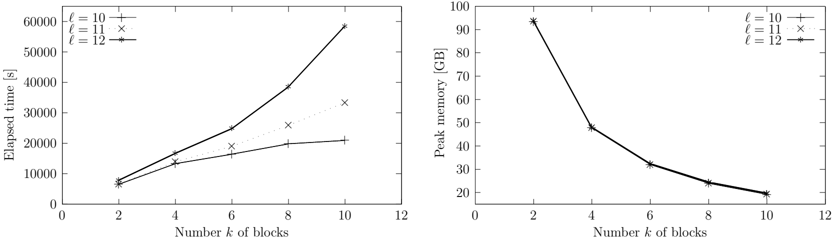
As input dataset here we used the entire human genome (version hg38), which has an approximate size of 3.1GB. The experiments were conducted on a machine with an Intel Core i5-4690 CPU at 3.50 GHz and 128GB of memory running GNU/Linux.

We ran the program by splitting the genome into $k = 2, 4, 6, 8, 10$ blocks and setting $\ell = 10, 11, 12$.

Proof-of-Concept Experiments

The figure depicts the change in elapsed time and peak memory usage as k and ℓ increase (space-time tradeoff).

Graph (a) shows an increase of time as k and ℓ increase. Graph (b) shows a decrease in memory as k increases.



(a) **(b)**

Conclusion and Open Problems

We presented a new technique for constructing antidictionaries in output-sensitive space.

Conclusion and Open Problems

We presented a new technique for constructing antidictionaries in output-sensitive space.

The importance of our contribution is underlined by the following:

1. Any space-efficient algorithm designed for global data structures can be directly applied to the k blocks in our technique to further reduce the working space.

Conclusion and Open Problems

We presented a new technique for constructing antidictionaries in output-sensitive space.

The importance of our contribution is underlined by the following:

1. Any space-efficient algorithm designed for global data structures can be directly applied to the k blocks in our technique to further reduce the working space.
2. There is a connection between MAWs and other word regularities. Our technique could potentially be applied to computing these regularities in output-sensitive space.

Conclusion and Open Problems

We presented a new technique for constructing antidictionaries in output-sensitive space.

The importance of our contribution is underlined by the following:

1. Any space-efficient algorithm designed for global data structures can be directly applied to the k blocks in our technique to further reduce the working space.
2. There is a connection between MAWs and other word regularities. Our technique could potentially be applied to computing these regularities in output-sensitive space.
3. Our technique could serve as a basis for a new parallelisation scheme for constructing antidictionaries, in which several blocks are processed concurrently.

Conclusion and Open Problems

Thank you!