Finding Maximal Exact Matches in Graph

G. Trapani

Università di Pisa

13/05/2024

UNIVERSITÀ DI PISA

Contents

Prerequisites
BWT Transform: Pseudocode

Assuming our strings' indices are 1-based, we now give a non-efficient algorithm to calculate the BWT transform of a string.

```
def BWT(s: str) -> str:
        T = []
        for character in s:
                s = s[len(s)] + s[1..len(s)-1]
                T.push(s)
        sort_lexicographically(T)
        return last_column(T)
```

Prerequisites

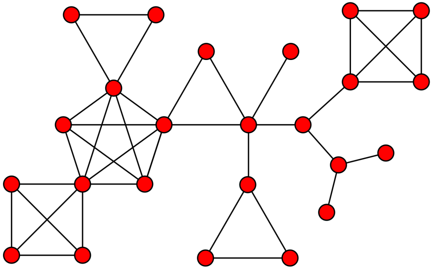
BWT Transform: Example

Transformation				
1. Input	2. All rotations	3. Sort into lexical order	4. Take the last column	5. Output
BANANA$	BANANA$ $BANANA A$BANAN NA$BANA ANA$BAN NANA$BA ANANA$B BANANA$	$BANANA BANANA$ A$BANAN ANA$BAN ANANA$B BANANA$ NA$BANA NANA$BA	$BANANA BANANA$ A$BANAN ANA$BAN ANANA$B BANANA$ NA$BANA NANA$BA	A$NNBAA

UNIVERSITÀ DI PISA

Definition (Block Graph)

We call **block graph** an undirected graph in which every biconnected component (i.e. a **block**) is a clique.



Definition (Elastic Founder Graph)

Consider a block graph $G = (V, E, l)$ with $l : V \rightarrow \Sigma^+$. We call such a graph an **indexable Elastic Founder Graph** if the **semi-repeat-free** property holds: for each v in block V_i, $l(v)$ occurs in G only as prefix of paths starting with some $w \in V_i$.

Let $Q \in \Sigma^+$ be a query string, κ be a threshold, $lext(i, P, j)$ the left extension of the string $P[i..j]$ and $rext(i, P, j)$ the right extension.

Definition (LEFTMAX)

A match $([x..y], (i, P, j))$ of $Q[x..y]$ in G satisfies the *LEFTMAX* property if and only if

$$x = 1 \ \vee \ lext(i, P, j) = \emptyset \ \vee \ Q[x-1] \notin lext(i, P, j)$$

We can analogously define the RIGHTMAX property.

Let $Q \in \Sigma^+$ be a query string, κ be a threshold, $lext(i, P, j)$ the left extension of the string $P[i..j]$ and $rext(i, P, j)$ the right extension.

Definition (LEFTMAX)

A match $([x..y], (i, P, j))$ of $Q[x..y]$ in G satisfies the *LEFTMAX* property if and only if

$$x = 1 \ \vee \ lext(i, P, j) = \emptyset \ \vee \ Q[x-1] \notin lext(i, P, j)$$

We can analogously define the RIGHTMAX property.

Let $Q \in \Sigma^+$ be a query string, κ be a threshold, $lext(i, P, j)$ the left extension of the string $P[i..j]$ and $rext(i, P, j)$ the right extension.

Definition (LEFTMAX)

A match $([x..y], (i, P, j))$ of $Q[x..y]$ in G satisfies the *LEFTMAX* property if and only if

$$x = 1 \;\vee\; lext(i, P, j) = \emptyset \;\vee\; Q[x-1] \notin lext(i, P, j)$$

We can analogously define the RIGHTMAX property.

Definition (κ-MEM)

A match $([x..y], (i, P, j))$ of $Q[x..y]$ in G is called a κ-MEM if it satisfies all the following conditions:

1. $LEFTMAX \vee |lext(i, P, j)| \geq 2$
2. $RIGHTMAX \vee |rext(i, P, j)| \geq 2$
3. $y - x + 1 \geq \kappa$

MEMs in Node Labels

Node MEMs: definition

Definition (Node MEM)

A κ-MEM between a query string Q and the label $l(v)$ of a vertex $v \in V$ is called a node-MEM.

To give an algorithm to find node-MEMs we must consider the text

$$T_{\text{nodes}} = \prod_{v \in V} 0 \times l(v).$$

We also need a data structure supporting the following operations over a bitvector B:

1. `r = rank(B, i)` in $O(1)$ with $r = \sum_{j=1}^{i} B[j]$,
2. `j = select(B, r)` in $O(1)$ with $j \leq i$ the position of the r-th 1 in B.

Università di Pisa

Definition (Node MEM)

A κ-MEM between a query string Q and the label $l(v)$ of a vertex $v \in V$ is called a node-MEM.

To give an algorithm to find node-MEMs we must consider the text

$$T_{\text{nodes}} = \prod_{v \in V} 0 \times l(v).$$

We also need a data structure supporting the following operations over a bitvector B:

1. `r = rank(B, i)` in $O(1)$ with $r = \sum_{j=1}^{i} B[j]$,
2. `j = select(B, r)` in $O(1)$ with $j \leq i$ the position of the r-th 1 in B.

Definition (Node MEM)

A κ-MEM between a query string Q and the label $l(v)$ of a vertex $v \in V$ is called a node-MEM.

To give an algorithm to find node-MEMs we must consider the text

$$T_{\text{nodes}} = \prod_{v \in V} 0 \times l(v).$$

We also need a data structure supporting the following operations over a bitvector B:

1. `r = rank(B, i)` in $O(1)$ with $r = \sum_{j=1}^{i} B[j]$,
2. `j = select(B, r)` in $O(1)$ with $j \leq i$ the position of the r-th 1 in B.

We assume we have at our disposal the following procedures:

1. `mems_using_bidirectional_bwts` which takes as input two bidirectional BWT indices on strings T, Q and a threshold κ and outputs Q' MEM strings with $|Q'| \geq \kappa$ and for each string Q' four BWT intervals $([i_T..j_T], [i'_T..j'_T], [i_Q..j_Q], [i'_Q..j'_Q])$ which represent the maximal matches of T and Q in $O(|T| + |Q|)$ time;
2. `mems_from_bidirectional_bwt` which takes as input four BWT intervals and outputs the corresponding MEM string Q' in $O(|Q'|)$.

For both of these algorithms, one may refer to Algorithm 11.3 and Algorithm 11.4 from Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing (Belazzougui et alia).

MEMs in Node Labels

Algorithm to find k-node MEMs: prerequisites

We assume we have at our disposal the following procedures:

1. `mems_using_bidirectional_bwts` which takes as input two bidirectional BWT indices on strings T, Q and a threshold κ and outputs Q' MEM strings with $|Q'| \geq \kappa$ and for each string Q' four BWT intervals $([i_T..j_T], [i'_T..j'_T], [i_Q..j_Q], [i'_Q..j'_Q])$ which represent the maximal matches of T and Q in $O(|T| + |Q|)$ time;
2. `mems_from_bidirectional_bwt` which takes as input four BWT intervals and outputs the corresponding MEM string Q' in $O(|Q'|)$.

For both of these algorithms, one may refer to Algorithm 11.3 and Algorithm 11.4 from Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing (Belazzougui et alia).

We assume we have at our disposal the following procedures:

1. `mems_using_bidirectional_bwts` which takes as input two bidirectional BWT indices on strings T, Q and a threshold κ and outputs Q' MEM strings with $|Q'| \geq \kappa$ and for each string Q' four BWT intervals $([i_T..j_T], [i'_T..j'_T], [i_Q..j_Q], [i'_Q..j'_Q])$ which represent the maximal matches of T and Q in $O(|T| + |Q|)$ time;
2. `mems_from_bidirectional_bwt` which takes as input four BWT intervals and outputs the corresponding MEM string Q' in $O(|Q'|)$.

For both of these algorithms, one may refer to Algorithm 11.3 and Algorithm 11.4 from Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing (Belazzougui et alia).

MEMs in Node Labels

Algorithm to find k-node MEMs

We can now describe the algorithm to find node MEMs.

1. We build a bitvector B to mark the location of 0s in T_{nodes} so that with the rank operation we can identify the corresponding node in G.
2. We call `mems_using_bidirectional_bwts` using bidirectional BWT indices on strings T_{nodes} and Q.
3. For each string Q' we get, we call `mems_from_bidirectional_bwt` on its corresponding BWT intervals.
4. We use B to obtain the tuple (i, P, j) in $O(1)$ time.

The algorithm described above has complexity $O(|T_{\text{nodes}}| + |Q| + N_{\kappa})$ with N_{κ} the number of output MEMs.

UNIVERSITÀ DI PISA

MEMs in Node Labels

Algorithm to find k-node MEMs

We can now describe the algorithm to find node MEMs.

1. We build a bitvector B to mark the location of 0s in T_{nodes} so that with the rank operation we can identify the corresponding node in G.
2. We call `mems_using_bidirectional_bwts` using bidirectional BWT indices on strings T_{nodes} and Q.
3. For each string Q' we get, we call `mems_from_bidirectional_bwt` on its corresponding BWT intervals.
4. We use B to obtain the tuple (i, P, j) in $O(1)$ time.

The algorithm described above has complexity $O(|T_{\text{nodes}}| + |Q| + N_\kappa)$ with N_κ the number of output MEMs.

UNIVERSITÀ DI PISA

We can now describe the algorithm to find node MEMs.

1. We build a bitvector B to mark the location of 0s in T_{nodes} so that with the rank operation we can identify the corresponding node in G.
2. We call `mems_using_bidirectional_bwts` using bidirectional BWT indices on strings T_{nodes} and Q.
3. For each string Q' we get, we call `mems_from_bidirectional_bwt` on its corresponding BWT intervals.
4. We use B to obtain the tuple (i, P, j) in $O(1)$ time.

The algorithm described above has complexity $O(|T_{\text{nodes}}| + |Q| + N_\kappa)$ with N_κ the number of output MEMs.

MEMs in Node Labels
Algorithm to find k-node MEMs

We can now describe the algorithm to find node MEMs.

1. We build a bitvector B to mark the location of 0s in T_{nodes} so that with the rank operation we can identify the corresponding node in G.
2. We call `mems_using_bidirectional_bwts` using bidirectional BWT indices on strings T_{nodes} and Q.
3. For each string Q' we get, we call `mems_from_bidirectional_bwt` on its corresponding BWT intervals.
4. We use B to obtain the tuple (i, P, j) in $O(1)$ time.

The algorithm described above has complexity $O(|T_{\text{nodes}}| + |Q| + N_\kappa)$ with N_κ the number of output MEMs.

We can now describe the algorithm to find node MEMs.

1. We build a bitvector B to mark the location of 0s in T_{nodes} so that with the rank operation we can identify the corresponding node in G.
2. We call `mems_using_bidirectional_bwts` using bidirectional BWT indices on strings T_{nodes} and Q.
3. For each string Q' we get, we call `mems_from_bidirectional_bwt` on its corresponding BWT intervals.
4. We use B to obtain the tuple (i, P, j) in $O(1)$ time.

The algorithm described above has complexity $O(|T_{\text{nodes}}| + |Q| + N_{\kappa})$ with N_{κ} the number of output MEMs.

MEMs in Node Labels

Algorithm to find k-node MEMs

We can now describe the algorithm to find node MEMs.

1. We build a bitvector B to mark the location of 0s in T_{nodes} so that with the rank operation we can identify the corresponding node in G.
2. We call `mems_using_bidirectional_bwts` using bidirectional BWT indices on strings T_{nodes} and Q.
3. For each string Q' we get, we call `mems_from_bidirectional_bwt` on its corresponding BWT intervals.
4. We use B to obtain the tuple (i, P, j) in $O(1)$ time.

The algorithm described above has complexity $O(|T_{\text{nodes}}| + |Q| + N_\kappa)$ with N_κ the number of output MEMs.

Università di Pisa

k-Node MEMs spanning exactly L nodes

Algorithm to find k-Node MEMs spanning exactly L nodes: prerequisites

1. We call P_G^L a path of G spanning exactly L nodes.
2. We define two new symbols c and d not originally part of the alphabet Σ and the two operators:

$$left(u) = \begin{cases} c \text{ if } lext(u) = \{c\} \\ \# \text{ otherwise} \end{cases}$$

$$right(u) = \begin{cases} d \text{ if } rext(u) = \{d\} \\ \# \text{ otherwise} \end{cases}$$

3. We define the text

$$T_L = 0 \times \prod_{u_1,..,u_L \in P_G^L} \left(left(u_1) \times l(u_1) \times \cdots \times l(u_L) \times right(u_L) \times 0 \right).$$

k-Node MEMs spanning exactly L nodes

Algorithm to find k-Node MEMs spanning exactly L nodes: prerequisites

1. We call P_G^L a path of G spanning exactly L nodes.
2. We define two new symbols c and d not originally part of the alphabet Σ and the two operators:

$$left(u) = \begin{cases} c \text{ if } lext(u) = \{c\} \\ \# \text{ otherwise} \end{cases}$$

$$right(u) = \begin{cases} d \text{ if } rext(u) = \{d\} \\ \# \text{ otherwise} \end{cases}$$

3. We define the text

$$T_L = 0 \times \prod_{u_1,..,u_L \in P_G^L} \left(left(u_1) \times l(u_1) \times \cdots \times l(u_L) \times right(u_L) \times 0 \right).$$

k-Node MEMs spanning exactly L nodes

Algorithm to find k-Node MEMs spanning exactly L nodes: prerequisites

1. We call P_G^L a path of G spanning exactly L nodes.
2. We define two new symbols c and d not originally part of the alphabet Σ and the two operators:

$$left(u) = \begin{cases} c \text{ if } lext(u) = \{c\} \\ \# \text{ otherwise} \end{cases}$$

$$right(u) = \begin{cases} d \text{ if } rext(u) = \{d\} \\ \# \text{ otherwise} \end{cases}$$

3. We define the text

$$T_L = 0 \times \prod_{u_1,..,u_L \in P_G^L} \Big(left(u_1) \times l(u_1) \times \cdots \times l(u_L) \times right(u_L) \times 0 \Big).$$

k-Node MEMs spanning exactly L nodes

Algorithm to find k-Node MEMs spanning exactly L nodes

1. We modify the `mems_from_bidirectional_bwt` so that it makes use of the symbols defined before we get the result using the algorithm we defined for κ-node MEMs.
2. The complexity we get is $O(|T_L| + |Q| + M_{\kappa,L})$ with $M_{\kappa,L}$ the number of output MEMs.
3. Let d be the maximum in-degree (or out-degree) of a node, n the total label length of G. Now we can reformulate the time complexity.
4. $|T_L|$ is the concatenation of paths of G of length L: for a node v the number of paths containing $l(v)$ is at most $L \times d^{L-1}$. The complexity can be rewritten as $O(|Q| + M_{\kappa,L} + n \times L \times d^{L-1})$ which is exponential on L.

1. We modify the `mems_from_bidirectional_bwt` so that it makes use of the symbols defined before we get the result using the algorithm we defined for κ-node MEMs.
2. The complexity we get is $O(|T_L| + |Q| + M_{\kappa,L})$ with $M_{\kappa,L}$ the number of output MEMs.
3. Let d be the maximum in-degree (or out-degree) of a node, n the total label length of G. Now we can reformulate the time complexity.
4. $|T_L|$ is the concatenation of paths of G of length L: for a node v the number of paths containing $l(v)$ is at most $L \times d^{L-1}$. The complexity can be rewritten as $O(|Q| + M_{\kappa,L} + n \times L \times d^{L-1})$ which is exponential on L.

1. We modify the `mems_from_bidirectional_bwt` so that it makes use of the symbols defined before we get the result using the algorithm we defined for κ-node MEMs.
2. The complexity we get is $O(|T_L| + |Q| + M_{\kappa,L})$ with $M_{\kappa,L}$ the number of output MEMs.
3. Let d be the maximum in-degree (or out-degree) of a node, n the total label length of G. Now we can reformulate the time complexity.
4. $|T_L|$ is the concatenation of paths of G of length L: for a node v the number of paths containing $l(v)$ is at most $L \times d^{L-1}$. The complexity can be rewritten as $O(|Q| + M_{\kappa,L} + n \times L \times d^{L-1})$ which is exponential on L.

k-Node MEMs spanning exactly L nodes

Algorithm to find k-Node MEMs spanning exactly L nodes

1. We modify the `mems_from_bidirectional_bwt` so that it makes use of the symbols defined before we get the result using the algorithm we defined for κ-node MEMs.
2. The complexity we get is $O(|T_L| + |Q| + M_{\kappa,L})$ with $M_{\kappa,L}$ the number of output MEMs.
3. Let d be the maximum in-degree (or out-degree) of a node, n the total label length of G. Now we can reformulate the time complexity.
4. $|T_L|$ is the concatenation of paths of G of length L: for a node v the number of paths containing $l(v)$ is at most $L \times d^{L-1}$. The complexity can be rewritten as $O(|Q| + M_{\kappa,L} + n \times L \times d^{L-1})$ which is exponential on L.

MEMs in EFGs

k-Node MEMs in EFGs: Remark

Remark

Given an indexable EFG $G = (V, E, l)$, for each $(v, w) \in E$ string $l(v)l(w)$ occurs only as prefix of paths starting with v.

Rephrasing what is written above, all occurrences of some string S in G spanning at least four nodes can be decomposed as $\alpha l(u_2) \dots l(u_{L-1})\beta$ such that:

1. $u_2 \dots u_{L-1}$ is a path in G and $u_2, \dots, u_{L-1}$ are unequivocally identified;
2. $\alpha = l(u_1)[i..||u_1||]$ with $1 \le i \le ||u_1||$ for some $(u_1, u_2) \in E$;
3. $\beta = l(u_L)$ for some $(u_{L-1}, u_L) \in E$ or $\beta = l(uL)(l(uL+1)[1..j])$ with $1 \le j \le ||u_{L+1}||$ for some $(u_{L-1}, u_L), (u_L, u_{L+1}) \in E$.

Note that $\alpha, \beta \neq \epsilon$ and β has as prefix a full node label, α might spell any suffix of a node label.

Remark

Given an indexable EFG $G = (V, E, l)$, for each $(v, w) \in E$ string $l(v)l(w)$ occurs only as prefix of paths starting with v.

Rephrasing what is written above, all occurrences of some string S in G spanning at least four nodes can be decomposed as $\alpha l(u_2) \dots l(u_{L-1})\beta$ such that:

1. $u_2 \dots u_{L-1}$ is a path in G and $u_2, \dots, u_{L-1}$ are unequivocally identified;
2. $\alpha = l(u_1)[i..||u_1||]$ with $1 \leq i \leq ||u_1||$ for some $(u_1, u_2) \in E$;
3. $\beta = l(u_L)$ for some $(u_{L-1}, u_L) \in E$ or $\beta = l(uL)(l(uL+1)[1..j])$ with $1 \leq j \leq ||u_{L+1}||$ for some (u_{L-1}, u_L), $(u_L, u_{L+1}) \in E$.

Note that $\alpha, \beta \neq \epsilon$ and β has as prefix a full node label, α might spell any suffix of a node label.

Remark

Given an indexable EFG $G = (V, E, l)$, for each $(v, w) \in E$ string $l(v)l(w)$ occurs only as prefix of paths starting with v.

Rephrasing what is written above, all occurrences of some string S in G spanning at least four nodes can be decomposed as $\alpha l(u_2) \dots l(u_{L-1})\beta$ such that:

1. $u_2 \dots u_{L-1}$ is a path in G and $u_2, \dots, u_{L-1}$ are unequivocally identified;
2. $\alpha = l(u_1)[i..||u_1||]$ with $1 \leq i \leq ||u_1||$ for some $(u_1, u_2) \in E$;
3. $\beta = l(u_L)$ for some $(u_{L-1}, u_L) \in E$ or $\beta = l(uL)(l(uL+1)[1..j])$ with $1 \leq j \leq ||u_{L+1}||$ for some (u_{L-1}, u_L), $(u_L, u_{L+1}) \in E$.

Note that $\alpha, \beta \neq \epsilon$ and β has as prefix a full node label, α might spell any suffix of a node label.

First, we take the text $T_3' = \prod_{(u,v),(v,w)\in E}\big(l(u)l(v)l(w) \times 0\big)$.

1. We mark all implicit or explicit nodes $\bar{p}$ such that the corresponding root-to-$\bar{p}$ path spells $l(u)l(v)$ for some $(u, v) \in E$, so that we can query in constant time if $\bar{p}$ is such a node.
2. We compute pointers from each node $\bar{p}$ to an arbitrarily chosen leaf in the subtree rooted at $\bar{p}$;
3. for each node $v \in V$ of the indexable EFG we build trie T_v for the set of strings $\overline{l(u)} : (u, v) \in E$;
4. for each leaf, we store the corresponding path uvw and the starting position of the suffix inside $l(u)l(v)l(w)$.

First, we take the text $T_3' = \prod_{(u,v),(v,w)\in E} \big(l(u)l(v)l(w) \times 0\big)$.

1. We mark all implicit or explicit nodes $\bar{p}$ such that the corresponding root-to-$\bar{p}$ path spells $l(u)l(v)$ for some $(u, v) \in E$, so that we can query in constant time if $\bar{p}$ is such a node.
2. We compute pointers from each node $\bar{p}$ to an arbitrarily chosen leaf in the subtree rooted at $\bar{p}$;
3. for each node $v \in V$ of the indexable EFG we build trie T_v for the set of strings $l(\bar{u}) : (u, v) \in E$;
4. for each leaf, we store the corresponding path uvw and the starting position of the suffix inside $l(u)l(v)l(w)$.

First, we take the text $T'_3 = \prod_{(u,v),(v,w)\in E}\big(l(u)l(v)l(w) \times 0\big)$.

1. We mark all implicit or explicit nodes $\bar{p}$ such that the corresponding root-to-$\bar{p}$ path spells $l(u)l(v)$ for some $(u,v) \in E$, so that we can query in constant time if $\bar{p}$ is such a node.
2. We compute pointers from each node $\bar{p}$ to an arbitrarily chosen leaf in the subtree rooted at $\bar{p}$;
3. for each node $v \in \bar{V}$ of the indexable EFG we build trie T_v for the set of strings $l(\bar{u}) : (u,v) \in E$;
4. for each leaf, we store the corresponding path uvw and the starting position of the suffix inside $l(u)l(v)l(w)$.

First, we take the text $T'_3 = \prod_{(u,v),(v,w)\in E}\big(l(u)l(v)l(w)\times 0\big)$.

1. We mark all implicit or explicit nodes $\bar{p}$ such that the corresponding root-to-$\bar{p}$ path spells $l(u)l(v)$ for some $(u,v)\in E$, so that we can query in constant time if $\bar{p}$ is such a node.
2. We compute pointers from each node $\bar{p}$ to an arbitrarily chosen leaf in the subtree rooted at $\bar{p}$;
3. for each node $v \in V$ of the indexable EFG we build trie T_v for the set of strings $\overleftarrow{l(u)} : (u,v)\in E$;
4. for each leaf, we store the corresponding path uvw and the starting position of the suffix inside $l(u)l(v)l(w)$.

MEMs in EFGs

k-Node MEMs in EFGs spanning more than 3 nodes: preprocessing

First, we take the text $T_3' = \prod_{(u,v),(v,w)\in E}\left(l(u)l(v)l(w) \times 0\right)$.

1. We mark all implicit or explicit nodes $\bar{p}$ such that the corresponding root-to-$\bar{p}$ path spells $l(u)l(v)$ for some $(u,v) \in E$, so that we can query in constant time if $\bar{p}$ is such a node.
2. We compute pointers from each node $\bar{p}$ to an arbitrarily chosen leaf in the subtree rooted at $\bar{p}$;
3. for each node $v \in V$ of the indexable EFG we build trie T_v for the set of strings $l(\bar{u}) : (u,v) \in E$;
4. for each leaf, we store the corresponding path uvw and the starting position of the suffix inside $l(u)l(v)l(w)$.

Let $\bar{p}$ be the suffix tree node of T'_3 reached from the root by spelling $Q[1..y]$ in the suffix tree until we cannot continue with $Q[y+1]$:

1. If we cannot continue with a 0, $Q[1..y]$ spans no more than 3 nodes, so we can discard it. We can now consider matching $Q[2..y]$ in G taking the suffix link of $\bar{p}$.
2. If we can continue with a 0 and the occurrences of $Q[1..y]$ span no more than 2 nodes, we proceed as in the previous step.
3. In this case, $Q[1..y] = \alpha l(u_2)l(u_3)$ for exactly one $u_2 \in V$, with $(u_2, u_3) \in E$, we follow the suffix link walk from $\bar{p}$ until we find the marked node $\bar{q}$ corresponding to $l(u_2)l(u_3)$: from $\bar{q}$ we try to match $Q[y+1..]$ until failure, matching $Q[y+1..y']$ and reaching node $\bar{r}$.

MEMs in EFGs

k-Node MEMs in EFGs spanning more than 3 nodes: processing

Let $\bar{p}$ be the suffix tree node of T'_3 reached from the root by spelling $Q[1..y]$ in the suffix tree until we cannot continue with $Q[y+1]$:

1. If we cannot continue with a 0, $Q[1..y]$ spans no more than 3 nodes, so we can discard it. We can now consider matching $Q[2..y]$ in G taking the suffix link of $\bar{p}$.
2. If we can continue with a 0 and the occurrences of $Q[1..y]$ span no more than 2 nodes, we proceed as in the previous step.
3. In this case, $Q[1..y] = \alpha l(u_2)l(u_3)$ for exactly one $u_2 \in V$, with $(u_2, u_3) \in E$, we follow the suffix link walk from $\bar{p}$ until we find the marked node $\bar{q}$ corresponding to $l(u_2)l(u_3)$: from $\bar{q}$ we try to match $Q[y+1..]$ until failure, matching $Q[y+1..y']$ and reaching node $\bar{r}$.

Let $\bar{p}$ be the suffix tree node of T_3' reached from the root by spelling $Q[1..y]$ in the suffix tree until we cannot continue with $Q[y+1]$:

1. If we cannot continue with a 0, $Q[1..y]$ spans no more than 3 nodes, so we can discard it. We can now consider matching $Q[2..y]$ in G taking the suffix link of $\bar{p}$.
2. If we can continue with a 0 and the occurrences of $Q[1..y]$ span no more than 2 nodes, we proceed as in the previous step.
3. In this case, $Q[1..y] = \alpha l(u_2)l(u_3)$ for exactly one $u_2 \in V$, with $(u_2, u_3) \in E$, we follow the suffix link walk from $\bar{p}$ until we find the marked node $\bar{q}$ corresponding to $l(u_2)l(u_3)$: from $\bar{q}$ we try to match $Q[y+1..]$ until failure, matching $Q[y+1..y']$ and reaching node $\bar{r}$.

MEMs in EFGs

k-Node MEMs in EFGs spanning more than 3 nodes: processing

Let $\bar{p}$ be the suffix tree node of T'_3 reached from the root by spelling $Q[1..y]$ in the suffix tree until we cannot continue with $Q[y+1]$:

1. If we cannot continue with a 0, $Q[1..y]$ spans no more than 3 nodes, so we can discard it. We can now consider matching $Q[2..y]$ in G taking the suffix link of $\bar{p}$.
2. If we can continue with a 0 and the occurrences of $Q[1..y]$ span no more than 2 nodes, we proceed as in the previous step.
3. In this case, $Q[1..y] = \alpha l(u_2) l(u_3)$ for exactly one $u_2 \in V$, with $(u_2, u_3) \in E$, we follow the suffix link walk from $\bar{p}$ until we find the marked node $\bar{q}$ corresponding to $l(u_2)l(u_3)$: from $\bar{q}$ we try to match $Q[y+1..]$ until failure, matching $Q[y+1..y']$ and reaching node $\bar{r}$.

Theorem (Algorithm complexity)

Let alphabet Σ be of constant size, and let $G = (V, E, l)$ be an indexable Elastic Founder Graph of height H, that is, the maximum number of nodes in a block of G is H. The algorithm to find κ-node-MEMs spanning $L > 3$ nodes has time complexity $O(nH^2 + |Q| + M_\kappa)$ with $n = \sum_{v \in V} |v|$ and M_κ the number of output MEMs.

Proof.

It derives from the complexity of the algorithm to find κ-node MEMs spanning exactly L nodes given before. □

Theorem (Algorithm complexity)

Let alphabet Σ be of constant size, and let $G = (V, E, I)$ be an indexable Elastic Founder Graph of height H, that is, the maximum number of nodes in a block of G is H. The algorithm to find κ-node-MEMs spanning $L > 3$ nodes has time complexity $O(nH^2 + |Q| + M_\kappa)$ with $n = \sum_{v \in V} |v|$ and M_κ the number of output MEMs.

Proof.

It derives from the complexity of the algorithm to find κ-node MEMs spanning exactly L nodes given before. □

Corollaries

Corollary

The results we have given for κ-node MEMs, κ-node MEMs spanning exactly L nodes and EFGs hold when $Q[1..m]$ is replaced by a set of queries of total length m.

Corollary

The algorithms we have given before (including the corollary above) can be modified to report only MEMs that occur in text T formed by concatenating the rows (ignoring gaps and adding separator symbols) of the input MSA of the indexable EFG.

This can be done in additional $O(|T| + r \log r)$ time and $O(r \log n)$ bits of space, and with multiplicative factor $O(\log \log n)$ added to the running times of the respective algorithms, where r is the number of equal-letter runs in the BWT of T.

Corollaries

Corollary

The results we have given for κ-node MEMs, κ-node MEMs spanning exactly L nodes and EFGs hold when $Q[1..m]$ is replaced by a set of queries of total length m.

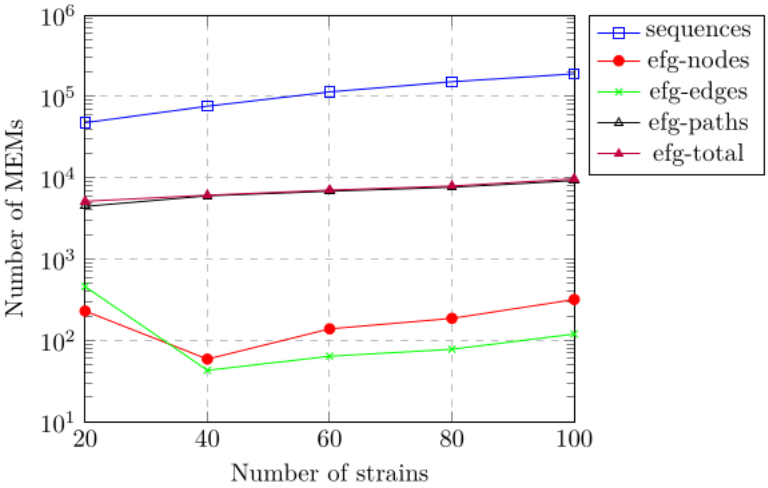
Corollary

The algorithms we have given before (including the corollary above) can be modified to report only MEMs that occur in text T formed by concatenating the rows (ignoring gaps and adding separator symbols) of the input MSA of the indexable EFG.

This can be done in additional $O(|T| + r \log r)$ time and $O(r \log n)$ bits of space, and with multiplicative factor $O(\log \log n)$ added to the running times of the respective algorithms, where r is the number of equal-letter runs in the BWT of T.

Experimental results

Number of MEMs with different indices and varying number of covid19 strains



Experimental results

Number of BWT runs with different indexes and varying number of covid19 strains