Efficient pattern matching in degenerate strings with the Burrows–Wheeler transform

Jacqueline W. Daykin$^{1,2,3}$ Richard Groult$^{4,3}$ Yannick Guesnet$^3$
Thierry Lecroq$^3$ Arnaud Lefebvre$^3$ Martine Léonard$^3$ Laurent Mouchard$^3$ Élise Prieur-Gaston$^3$ Bruce Watson$^{5,6}$

1 Aberystwyth Univ. (Mauritius Branch Campus), Mauritius
2 King’s College London, UK
3 Normandie Univ, UNIROUEN, LITIS, 76000 Rouen, France
4 Univ. de Picardie Jules Verne, Amiens, France
5 Stellenbosch Univ., South Africa
6 CAIR, CSIR Meraka, Pretoria, South Africa

Seqbio 2017, Lille, France, November 7th, 2017
Outline

1. Introduction

2. Pattern matching in degenerate strings with the BWT

3. Experiments
Outline

1. Introduction
2. Pattern matching in degenerate strings with the BWT
3. Experiments
Burrow-Wheeler Transform (BWT)

**Definition**

Let $x$ be a string built on a finite alphabet $\Sigma$. The BWT of $x$ is defined as the pair $(L, h)$ where $L$ is the last column of the matrix $M_x$ formed by all the sorted cyclic rotations of $x$ and $h$ is the index of $x$ in this matrix.
Burrow-Wheeler Transform (BWT)

Definition

Let $x$ be a string built on a finite alphabet $\Sigma$. The BWT of $x$ is defined as the pair $(L, h)$ where $L$ is the last column of the matrix $M_x$ formed by all the sorted cyclic rotations of $x$ and $h$ is the index of $x$ in this matrix.

$x = \text{BANANA}$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>B</th>
<th>A</th>
<th>N</th>
<th>A</th>
<th>N</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>N</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>N</td>
<td>A</td>
<td>N</td>
</tr>
</tbody>
</table>
Burrow-Wheeler Transform (BWT)

**Definition**

Let \( x \) be a string built on a finite alphabet \( \Sigma \). The BWT of \( x \) is defined as the pair \((L, h)\) where \( L \) is the last column of the matrix \( M_x \) formed by all the sorted cyclic rotations of \( x \) and \( h \) is the index of \( x \) in this matrix.

\[
\begin{align*}
\text{x = BANANA} & \quad \text{SA} & \quad \text{BWT} \\
\end{align*}
\]

\[\text{BWT(BANANA)} = (\text{NNBAAA}, 4)\]
Assume

\((i, j)\) is the interval in the SA of a text \(t\) of the suffixes of \(t\) starting with \(p[k + 1 \ldots m]\)

then

\((i', j')\) is the interval in the SA of \(t\) of the suffixes of \(t\) starting with \(p[k \ldots m]\)

with

\[i' = C[c] + \text{rank}_c(BWT, i - 1) + 1\] \[j' = C[c] + \text{rank}_c(BWT, j)\]

where

\[c = p[k],\ C[c] = \#\{i \mid t[i] < c\}\] and

\[\text{rank}_c(x, i)\] gives the number of occurrences of the letter \(c\) in the prefix \(x[1 \ldots i]\).
### Backward search – example

| → | 1 | 6 | A | B | A | N | A | N |
| → | 2 | 4 | A | N | A | B | A | N |
| → | 3 | 2 | A | N | A | N | A | B |
| → | 4 | 1 | B | A | N | A | N | A |
| → | 5 | 5 | N | A | B | A | N | A |
| → | 6 | 3 | N | A | N | A | B | A |

(1, 6) is the interval in the SA of BANANA of suffixes starting with \( \varepsilon \)
Introduction
Pattern matching in degenerate strings with the BWT
Experiments

Backward search – example

\[ \rightarrow 1 \quad 6 \quad A \quad B \quad A \quad N \quad A \quad N \]
\[ \quad 2 \quad 4 \quad A \quad N \quad A \quad B \quad A \quad N \]
\[ \quad 3 \quad 2 \quad A \quad N \quad A \quad N \quad A \quad B \]
\[ \quad 4 \quad 1 \quad B \quad A \quad N \quad A \quad N \quad A \]
\[ \quad 5 \quad 5 \quad N \quad A \quad B \quad A \quad N \quad A \]
\[ \rightarrow \quad 6 \quad 3 \quad N \quad A \quad N \quad A \quad B \quad A \]

(1, 6) is the interval in the SA of BANANA of suffixes starting with \( \varepsilon \)

\[ \rightarrow 1 \quad 6 \quad A \quad B \quad A \quad N \quad A \quad N \]
\[ \quad 2 \quad 4 \quad A \quad N \quad A \quad B \quad A \quad N \]
\[ \rightarrow \quad 3 \quad 2 \quad A \quad N \quad A \quad N \quad A \quad B \]
\[ \quad 4 \quad 1 \quad B \quad A \quad N \quad A \quad N \quad A \]
\[ \quad 5 \quad 5 \quad N \quad A \quad B \quad A \quad N \quad A \]
\[ \quad 6 \quad 3 \quad N \quad A \quad N \quad A \quad B \quad A \]

(6, 6) is the interval in the SA of BANANA of suffixes starting with A
Backward search – example

1 6 A B A N A N N
→ 2 4 A N A B A N
→ 3 2 A N A N A B
  4 1 B A N A N A
  5 5 N A B A N A
  6 3 N A N A B A

(2, 3) is the interval in the SA of BANANA of suffixes starting with AN
Backward search – example

(2, 3) is the interval in the SA of BANANA of suffixes starting with AN

(6, 6) is the interval in the SA of BANANA of suffixes starting with NAN
Degenerate strings

**Definition**

Given an alphabet $\Sigma$ we define a new alphabet $\Delta_\Sigma$ as the non-empty subsets of $\Sigma$:

$$\Delta_\Sigma = \mathcal{P}(\Sigma) \setminus \{\emptyset\}$$

Singletons are called **solid** letters.

*Degenerate or indeterminate* strings on an alphabet $\Sigma$ are strings of $\Delta_\Sigma$.

$$\Sigma = \{a, b, c, d, e\}$$

$$t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\}$$
Pattern matching on degenerate strings

**Definition**

Given 2 degenerate strings $p$ and $t$ find all the positions $0 \leq j < |t| - |p|$ on $t$ where $p[i] \cap t[i + j] \neq \emptyset$ for $0 \leq i < |p|$. 

$p = \{a\} \cdot \{c, d\}$ occurs at positions 3 and 4 in 
$t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\}$
Pattern matching on degenerate strings

Definition
Given 2 degenerate strings \( p \) and \( t \) find all the positions \( 0 \leq j < |t| - |p| \) on \( t \) where \( p[i] \cap t[i+j] \neq \emptyset \) for \( 0 \leq i < |p| \).

\[
p = \{a\} \cdot \{c, d\} \text{ occurs at positions 3 and 4 in } t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\}
\]
Pattern matching on degenerate strings

**Definition**

Given 2 degenerate strings $p$ and $t$ find all the positions $0 \leq j < |t| - |p|$ on $t$ where $p[i] \cap t[i+j] \neq \emptyset$ for $0 \leq i < |p|$.

$p = \{a\} \cdot \{c, d\}$ occurs at positions 3 and 4 in

$t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\}$
Burrows-Wheeler transform on degenerate strings (D-BWT)

Given an order on $\Delta_\Sigma$ denoted by the usual symbol $<$, we can compute the BWT of a degenerate string $x$ in the same way as for a regular string.

$t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\}$

$A = \{a, b, c\}, B = \{a, d\}, C = \{c, d\}, D = \{c, e\}$

$A < B < C < D$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>
Burrows-Wheeler transform on degenerate strings (D-BWT)

Jacqueline W. Daykin and Bruce Watson
A Text Transformation Scheme for Degenerate Strings

Jacqueline W. Daykin and Bruce Watson
Indeterminate String Factorizations and Degenerate Text Transformations
Mathematics in Computer Science 11(2) (2017) 209–218
Lin Huang, Victoria Popic and Serafim Batzoglou
Short read alignment with populations of genomes
*Bioinformatics* 29(13) (2013) i361–i370

Represent a collection of genomes called *reference multi-genome* and do pattern matching
SNPs (aka SNVs or substitutions)

4-letter alphabet $\{A, C, G, T\} \rightarrow$ 16-letter IUPAC encoding $\rightarrow$ 4-bit Gray code (to minimize # separate intervals during the search with the BWT)

Indels (insertions-deletions)

Corresponding sequences padded with surrounding bases (length depending on read length) are concatenated at the end of the reference multi-genome (separated by a special character)

Inversions, translocations and duplications

only both ends of the events are concatenated at the end of the reference multi-genome
Outline

1 Introduction

2 Pattern matching in degenerate strings with the BWT

3 Experiments
Backward search on the D-BWT: formalization

**OneStep** \((H, k, C, BWT = (L, h), p) = ((((r, s))) \mid r = C[c] + \text{rank}_c(L, i - 1) + 1, s = C[c] + \text{rank}_c(L, j), r \leq s, (i, j) \in H, c \in \Delta_{\Sigma} \text{ and } c \cap p[k] \neq \emptyset).)

Let \(\text{Step}(m, C, BWT, p) = \text{OneStep}(\{(1, n)\}, m, C, BWT, p)\) and \(\text{Step}(i, C, BWT, p) = \text{OneStep}(\text{Step}(i + 1, C, BWT, p), i, C, BWT, p)\) for \(1 \leq i \leq m - 1\).

In words, \(\text{Step}(i, C, BWT, p)\) applies step \(m\) through to \(i\) of the backward search. Then \(\text{Step}(1, C, BWT, p)\) contains the intervals in the SA of \(t\) of the suffixes of \(t\) starting with \(p\).
Lemma

The interval \((i, j) \in \text{Step}(k, C, BWT, p)\) if and only if \(p[k..m]\) is a degenerate prefix of \(M_t[h]\) for \(i \leq h \leq j\).

Corollary

The interval \((i, j) \in \text{Step}(1, C, BWT, p)\) if and only if \(p\) is a degenerate prefix of \(M_t[h]\) for \(i \leq h \leq j\).
Lemma

The intervals in $\text{OneStep}((i, j), k, C, BWT, p)$ do not overlap.
Intervals do not overlap

\[ t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\} \]

\[ A = \{a, b, c\}, B = \{a, d\}, C = \{c, d\}, D = \{c, e\} \]

\[ A < B < C < D \]

\[ p = \{c\} \cdot \{d\} \cdot \{b\} \]

\[ b \in A \text{ and } d \in B, C \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
Intervals do not overlap

\[ t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\} \]

\[ A = \{a, b, c\}, B = \{a, d\}, C = \{c, d\}, D = \{c, e\} \]

\[ A < B < C < D \]

\[ p = \{c\} \cdot \{d\} \cdot \{b\} \]

\[ b \in A \text{ and } d \in B, C \]

\[
\begin{array}{cccccc}
1 & 3 & A & B & A & D & C \\
2 & 5 & A & D & C & A & B \\
\Rightarrow & 3 & 4 & B & A & D & C & A \\
\Rightarrow & 4 & 2 & C & A & B & A & D \\
5 & 1 & D & C & A & B & A \\
\end{array}
\]
**Lemma**

The intervals in $\text{OneStep}((i, j), (i', j'), k, C, BWT, p)$ with $i \leq j < i' \leq j'$ do not overlap.

**Corollary**

Let $H$ be a set of non-overlapping intervals. The intervals in $\text{OneStep}(H, k, C, BWT, p)$ do not overlap.
Intervals do not overlap

\[ t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\} \]

\[ A = \{a, b, c\}, \quad B = \{a, d\}, \quad C = \{c, d\}, \quad D = \{c, e\} \]

\[ A < B < C < D \]

\[ p = \{c\} \cdot \{d\} \cdot \{b\} \]

\[ b \in A, \quad d \in B, \quad C \quad \text{and} \quad c \in A, \quad D \]

\[
\begin{array}{cccccc}
1 & 3 & A & B & A & D \\
2 & 5 & A & D & C & A \\
\Rightarrow & 3 & 4 & B & A & D & C & A \\
\Rightarrow & 4 & 2 & C & A & B & A & D \\
5 & 1 & D & C & A & B & A
\end{array}
\]
Intervals do not overlap

\[ t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\} \]

\[ A = \{a, b, c\}, B = \{a, d\}, C = \{c, d\}, D = \{c, e\} \]

\[ A < B < C < D \]

\[ p = \{c\} \cdot \{d\} \cdot \{b\} \]

\[ b \in A, \ d \in B, \ C \text{ and } c \in A, \ D \]

\[ \Rightarrow \]

\[ 1 \quad 3 \quad A \quad B \quad A \quad D \quad C \]

\[ 2 \quad 5 \quad A \quad D \quad C \quad A \quad B \]

\[ 3 \quad 4 \quad B \quad A \quad D \quad C \quad A \]

\[ 4 \quad 2 \quad C \quad A \quad B \quad A \quad D \]

\[ \Rightarrow \]

\[ 5 \quad 1 \quad D \quad C \quad A \quad B \quad A \]
Consecutive intervals can merge

Lemma

\[ \text{Merge}(\text{OneStep}(((i, j), (j + 1, j')), k, C, BWT, p)) = \text{OneStep}(((i, j')), k, C, BWT, p). \]
Consecutive intervals can merge

\[ t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\} \]

\[ A = \{a, b, c\}, B = \{a, d\}, C = \{c, d\}, D = \{c, e\} \]

\[ A < B < C < D \]

\[ p = \{c\} \cdot \{d\} \cdot \{b\} \]

\[ b \in A, \ d \in B, C \text{ and } c \in A, D \]

\[ \begin{array}{cccccc}
1 & 3 & A & B & A & D \\
2 & 5 & A & D & C & A \\
\Rightarrow & 3 & 4 & B & A & D \\
\Rightarrow & 4 & 2 & C & A & B \\
5 & 1 & D & C & A & B \\
\end{array} \]
Consecutive intervals can merge

\[ t = \{c, e\} \cdot \{c, d\} \cdot \{a, b, c\} \cdot \{a, d\} \cdot \{a, b, c\} \]
\[ A = \{a, b, c\}, B = \{a, d\}, C = \{c, d\}, D = \{c, e\} \]
\[ A < B < C < D \]
\[ p = \{c\} \cdot \{d\} \cdot \{b\} \]
\[ b \in A, \ d \in B, \ C \text{ and } c \in A, \ D \]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>A</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>→</td>
<td>3</td>
<td>4</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>→</td>
<td>4</td>
<td>2</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>D</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>
Consecutive intervals can merge

\[ t = \{c, \ e\} \cdot \{c, \ d\} \cdot \{a, \ b, \ c\} \cdot \{a, \ d\} \cdot \{a, \ b, \ c\} \]

\[ A = \{a, \ b, \ c\}, \ B = \{a, \ d\}, \ C = \{c, \ d\}, \ D = \{c, \ e\} \]

\[ A < B < C < D \]

\[ b \in A, \ d \in B, \ C \text{ and } c \in A, \ D \]

\[ \Rightarrow \]

\[
\begin{array}{cccccc}
& 1 & 3 & A & B & A & D & C \\
2 & 5 & A & D & C & A & B \\
3 & 4 & B & A & D & C & A \\
4 & 2 & C & A & B & A & D \\
& 5 & 1 & D & C & A & B & A \\
\end{array}
\]
A degenerate string is said to be conservative if its number of non-solid letters is upper-bounded by a fixed positive constant $q$.

**Theorem**

Let $t$ be a conservative degenerate string over a constant size alphabet. Let the number of degenerate letters of $t$ be bounded by a constant $q$. Then given the BWT of $t$, all the intervals in the BWT of occurrences of a pattern $p$ of length $m$ can be detected in time $O(qm^2)$. 
Outline

1. Introduction
2. Pattern matching in degenerate strings with the BWT
3. Experiments
Experiments

Degenerate patterns of length 8 in a solid string of length 5MB, $\sigma = 4$

![Graph showing experiment results for degenerate patterns. The x-axis represents the number of patterns, and the y-axis represents time in cs. The graph compares Hybrid, BNDM, and DBS methods.]
Experiments

One degenerate pattern of length 8 in a conservative degenerate string
Experiments

Degenerate patterns of length 8 in a solid string, $\sigma = 4$

![Graph showing time (cs) vs. number of patterns for different algorithms: Hybrid, BNDM, DBS. The graph illustrates how the time increases linearly with the number of patterns.](image-url)
Experiments

Degenerate patterns of length 8 in a solid string, $\sigma = 8$
Experiments

Intervals

The graph shows the performance comparison between RB Tree and List for pattern matching in degenerate strings with the BWT. The x-axis represents the length of the pattern, while the y-axis represents the time in seconds. The graph indicates that RB Tree performs better than List for longer patterns, with a decrease in time as the pattern length increases.
Perspectives

- Average case analysis
- Efficient data structure for handling intervals
- Using different order on the alphabet
- ...
Thank you for your attention!