On Undetected Redundancy in the Burrows-Wheeler Transform

Uwe Baier

Institute of Theoretical Computer Science
Ulm University
Data Compression

Why should we compress our data?...

...people from informatics should know best themselves...

- huge amount of data, storage can be expensive
- not every method can be implemented using streaming and parallelism ⇒ memory is an even more limited resource
- some compressed representations allow methods of string analysis to be performed much faster
In this talk

BWT Preliminaries

Tunneled BWT

Practical Implementation

Experimental Results

Conclusion
Context-Based Compression

Observation: similar contexts tend to be succeeded (or preceded) by similar characters

- In English texts, the letter q always is followed by an u
- The string eer tends to be preceded by a b

Can we use this knowledge to compress data?


BWT and sorted suffixes of $S = \text{easypeasy}$

```
y $ 
 e asy$
 e asypeasy$
 p easy$
 $ easypeasy$
 y peasy$
 a sy$
 a sypeasy$
 s y$
 s ypeasy$
```
BWT - What is it?

“The BWT is a string generated by concatenating all cyclic preceding characters of the lexicographically sorted suffixes of a string $S$.”

**BWT generation of** $S = \text{easypeasy}\$

<table>
<thead>
<tr>
<th>prec. char.</th>
<th>suffixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$$</td>
</tr>
<tr>
<td>s</td>
<td>y$</td>
</tr>
<tr>
<td>a</td>
<td>sy$</td>
</tr>
<tr>
<td>e</td>
<td>asy$</td>
</tr>
<tr>
<td>p</td>
<td>easy$</td>
</tr>
<tr>
<td>y</td>
<td>peasy$</td>
</tr>
<tr>
<td>s</td>
<td>ypeasy$</td>
</tr>
<tr>
<td>a</td>
<td>sypeasy$</td>
</tr>
<tr>
<td>e</td>
<td>asypeasy$</td>
</tr>
<tr>
<td>$</td>
<td>easypeasy$</td>
</tr>
</tbody>
</table>
BWT - What is it?

“The BWT L is a string generated by concatenating all cyclic preceding characters of the lexicographically sorted suffixes of a string S.”

BWT generation of $S = \text{easy} \text{peasy} \$

<table>
<thead>
<tr>
<th>prec. char.</th>
<th>suffixes</th>
<th>L</th>
<th>sorted suffixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$</td>
<td>y</td>
<td>$</td>
</tr>
<tr>
<td>s</td>
<td>y$</td>
<td>e</td>
<td>asy$</td>
</tr>
<tr>
<td>a</td>
<td>sy$</td>
<td>e</td>
<td>asy$</td>
</tr>
<tr>
<td>e</td>
<td>asy$</td>
<td>p</td>
<td>easy$</td>
</tr>
<tr>
<td>p</td>
<td>easy$</td>
<td>$</td>
<td>easypeasy$</td>
</tr>
<tr>
<td>y</td>
<td>peasy$</td>
<td>y</td>
<td>peasy$</td>
</tr>
<tr>
<td>s</td>
<td>ypeasy$</td>
<td>a</td>
<td>sy$</td>
</tr>
<tr>
<td>a</td>
<td>sypeasy$</td>
<td>a</td>
<td>sypeasy$</td>
</tr>
<tr>
<td>e</td>
<td>asypeasy$</td>
<td>e</td>
<td>aspeasy$</td>
</tr>
<tr>
<td>$</td>
<td>easypeasy$</td>
<td>s</td>
<td>y$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>ypeasy$</td>
</tr>
</tbody>
</table>
**BWT - What is it?**

“The BWT L is a string generated by concatenating all cyclic preceding characters of the lexicographically sorted suffixes of a string S.”

**BWT generation of** \(S = \text{easypeasy}$

<table>
<thead>
<tr>
<th>prec. char.</th>
<th>suffixes</th>
<th>L</th>
<th>sorted suffixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$</td>
<td>y</td>
<td>$</td>
</tr>
<tr>
<td>s</td>
<td>y$</td>
<td>e</td>
<td>asy$</td>
</tr>
<tr>
<td>a</td>
<td>sy$</td>
<td>e</td>
<td>asy$</td>
</tr>
<tr>
<td>e</td>
<td>asy$</td>
<td>p</td>
<td>easy$</td>
</tr>
<tr>
<td>p</td>
<td>easy$</td>
<td>$</td>
<td>easypeasy$</td>
</tr>
<tr>
<td>y</td>
<td>peasy$</td>
<td>y</td>
<td>peasy$</td>
</tr>
<tr>
<td>s</td>
<td>ypeasy$</td>
<td>a</td>
<td>sy$</td>
</tr>
<tr>
<td>a</td>
<td>sypeasy$</td>
<td>a</td>
<td>sypeasy$</td>
</tr>
<tr>
<td>e</td>
<td>asypeasy$</td>
<td>s</td>
<td>y$</td>
</tr>
<tr>
<td>$</td>
<td>easypeasy$</td>
<td>s</td>
<td>ypeasy$</td>
</tr>
</tbody>
</table>
BWT - Use for Data Compression?

- BWT places characters preceding the same context near to each other

  ![Diagram](chart.png)

- character distribution of small portions of BWT is skew

Common Compression Approaches

- transform local to global skewness (MTF [Ryabko, 1980])
  + entropy coding (Huffman-Coding [Huffman, 1952])

- run-length-encoding

  \[\cdots \text{aaaaaa} \cdots \Rightarrow \cdots \text{a01} \cdots\]

  \(6 \times 2 \Rightarrow 6 = (101)_2\)
BWT - Inverting

- generate F (first characters of sorted suffixes) by sorting L

```
L     F
y     $  
ey
e     a
     a
e     e
p     e
$s    e
y     p
a     s
     s
a     s
a     y
s     y
```

- $k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

⇒ collecting characters in $L$ during a walk through $L$ using correspondence yields the reversed original sequence
BWT - Inverting Example

\[ k\text{-th occurrence of character } c \text{ in } L \text{ corresponds to } k\text{-th occurrence of character } c \text{ in } F \]

\[
\begin{align*}
L & \quad F \\
\text{y} & \quad \$ \\
\text{e} & \quad a \\
\text{e} & \quad a \\
\text{p} & \quad e \\
\$ & \quad e \\
\text{y} & \quad p \\
\text{a} & \quad s \\
\text{a} & \quad s \\
\text{s} & \quad y \\
\text{s} & \quad y \\
\end{align*}
\]

\[ S = \quad \$ \]
BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

\[ S = y$ \]
BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

\[
\begin{align*}
L & \quad F \\
y & \quad $ \\
e & \quad a \\
e & \quad a \\
p & \quad e \\
$ & \quad e \\
y & \quad p \\
a & \quad s \\
a & \quad s \\
s & \quad y \\
s & \quad y \\
S = sy$ & 
\end{align*}
\]
BWT - Inverting Example

*k*-th occurrence of character c in L corresponds to 
k*-th occurrence of character c in F

\[
\begin{align*}
L & \quad F \\
y & \quad \$ \\
e & \quad a \\
e & \quad a \\
p & \quad e \\
$ & \quad e \\
y & \quad p \\
a & \quad s \\
a & \quad s \\
s & \quad y \\
s & \quad y \\
S & = \quad \text{asy}$
\end{align*}
\]
BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

$L$

\[
\begin{align*}
y & \leftarrow \text{a} \\
e & \leftarrow \text{p} \\
e & \leftarrow \text{e} \\
p & \leftarrow \text{e} \\
$ & \leftarrow \text{e} \\
y & \leftarrow \text{p} \\
a & \leftarrow \text{s} \\
a & \leftarrow \text{s} \\
s & \leftarrow \text{y} \\
s & \leftarrow \text{y}
\end{align*}
\]

$F$

\[
\begin{align*}
$ & \leftarrow \text{a} \\
a & \leftarrow \text{a} \\
e & \leftarrow \text{e} \\
e & \leftarrow \text{e} \\
y & \leftarrow \text{y} \\
y & \leftarrow \text{y}
\end{align*}
\]

$S = \text{easy}$$
BWT - Inverting Example

The \( k \)-th occurrence of character \( c \) in \( L \) corresponds to the \( k \)-th occurrence of character \( c \) in \( F \).

\[
\begin{align*}
L & \quad F \\
y & \quad $ \\
e & \quad a \\
e & \quad a \\
p & \quad e \\
$ & \quad e \\
y & \quad p \\
a & \quad s \\
a & \quad s \\
s & \quad y \\
s & \quad y \\
S & = \quad \text{peasy}$
\end{align*}
\]
### BWT - Inverting Example

The `k`-th occurrence of character `c` in `L` corresponds to the `k`-th occurrence of character `c` in `F`.

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td>e</td>
</tr>
<tr>
<td>$</td>
<td>e</td>
</tr>
<tr>
<td>y</td>
<td>p</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
</tbody>
</table>

\[ S = \text{ypeasy}$ \]
BWT - Inverting Example

\textit{k-th occurrence of character c in L corresponds to k-th occurrence of character c in F}

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td>e</td>
</tr>
<tr>
<td>$</td>
<td>e</td>
</tr>
<tr>
<td>y</td>
<td>p</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>a</td>
<td>y</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
</tbody>
</table>

\[ S = \text{sypeasy}$ \]
BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$$</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td>e</td>
</tr>
<tr>
<td>$$</td>
<td>e</td>
</tr>
<tr>
<td>y</td>
<td>p</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>a</td>
<td>$$</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
</tbody>
</table>

$S = \text{asypeasy}$
BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

$L$
\[ y \rightarrow e \rightarrow e \rightarrow p \rightarrow \$

$F$
\[ \$ \rightarrow a \rightarrow a \rightarrow e \rightarrow e \rightarrow p \rightarrow s \rightarrow s \rightarrow y \rightarrow y \]

\[ S = \text{easypeasy}\$ \]
BWT - Inverting Example

\( k \)-th occurrence of character \( c \) in \( L \) corresponds to \( k \)-th occurrence of character \( c \) in \( F \)

\[
\begin{align*}
L & \quad F \\
\text{end} & \quad \text{end} \\
\text{y} & \quad \$ \\
\text{a} & \quad \text{a} \\
\text{e} & \quad \text{e} \\
\text{p} & \quad \text{e} \\
\$ & \quad \text{e} \\
\text{y} & \quad \text{p} \\
\text{a} & \quad \text{s} \\
\text{a} & \quad \text{s} \\
\text{s} & \quad \text{y} \\
\text{s} & \quad \text{y} \\
\end{align*}
\]

\( S = \text{easypeasy}\$ \)
Observation on Contexts

similar contexts tend to be preceded by the same character
⇒ similar contexts tend to be preceded by the same substrings

sorted suffixes and corr. prefixes of $S = \text{easypeasy}$

\[
\begin{array}{ll}
\text{easypeasy} & $ \\
\text{easype} & \text{asy}$ \\
\text{e} & \text{asypeasy}$ \\
\text{easyp} & \text{easy}$ \\
\varepsilon & \text{easypeasy}$ \\
\text{easy} & \text{peasy}$ \\
\text{easypea} & \text{sy}$ \\
\text{ea} & \text{sypeasy}$ \\
\text{easypeas} & \text{y}$ \\
\text{eas} & \text{ypeasy}$
\end{array}
\]

- Can we use this?
1. determine a set of blocks (equal consecutive preceding substrings) to be tunneled

```
  : :
ea sypeasy$
easypeas y$
eas ypeasy$
```

2. determine the corresponding columns in L and F for each block

3. cross out all entries from the columns in L and F, except for the uppermost ones

4. remove positions which were crossed out both in F and L

5. result: shortened L and two bitvectors cntL and cntF saving the remaining crosses
BWT Tunneling - Example

1. determine a set of blocks (equal consecutive preceding substrings) to be tunneled

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$$$</td>
</tr>
<tr>
<td>$e$</td>
<td>$a$</td>
</tr>
<tr>
<td>$e$</td>
<td>$a$</td>
</tr>
<tr>
<td>$p$</td>
<td>$e$</td>
</tr>
<tr>
<td>$$</td>
<td>$e$</td>
</tr>
<tr>
<td>$y$</td>
<td>$p$</td>
</tr>
<tr>
<td>$a$</td>
<td>$s$</td>
</tr>
<tr>
<td>$a$</td>
<td>$s$</td>
</tr>
<tr>
<td>$e$</td>
<td>$y$</td>
</tr>
<tr>
<td>$a$</td>
<td>$y$</td>
</tr>
<tr>
<td>$s$</td>
<td>$y$</td>
</tr>
</tbody>
</table>
2. determine the corresponding columns in L and F for each block

```
L
y
  e
  e
  p
  $  
y
F
$  
  a
  a
  e
  e
  p
  s
  s
  y
  y
  e
  e
  a
  a
  s
  s
```
3. cross out all entries from the columns in L and F, except for the uppermost ones

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td>e</td>
</tr>
<tr>
<td>$</td>
<td>y</td>
</tr>
<tr>
<td>y</td>
<td>p</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
</tbody>
</table>
BWT Tunneling - Example

4. remove positions which were crossed out both in F and L

\[
\begin{array}{cccc}
\text{L} & & & \text{F} \\
\text{y} & & & \$ \\
\text{e} & & & a \\
\text{x} & \text{x} & & \text{x} \\
\text{e} & & & a \\
\text{p} & & & \text{e} \\
\text{y} & & & \text{p} \\
\text{a} & & & \text{s} \\
\text{x} & \text{x} & \text{x} & \text{x} \\
\text{s} & & & y \\
\text{e} & & & y \\
\end{array}
\]
### BWT Tunneling - Example

5. result: shortened L and two bitvectors cntL and cntF saving the remaining crosses

<table>
<thead>
<tr>
<th>L</th>
<th>cntL</th>
<th>cntF</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Tunneling - Recap

tunneling removes all entries from a block except for
- the uppermost row
- the rightmost column

▶ tunneling reduces run-lengths in \( L \) at cost of increasing the number of runs in \( \text{cntL} \) and \( \text{cntF} \) - is it worth it?
▶ Can we invert a tunneled BWT?
Tunneled BWT - Inverting

- sort regular characters in L to free places in F

<table>
<thead>
<tr>
<th>L</th>
<th>cntL</th>
<th>cntF</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td></td>
<td></td>
<td>$</td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td></td>
<td>e</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td>p</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td>s</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
<td>y</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
<td>y</td>
</tr>
</tbody>
</table>

- \(k\)-th occurrence of character \(c\) in L corresponds to \(k\)-th occurrence of character \(c\) in F
- use uppermost row of a tunnel for all rows of a block
- when entering a tunnel, save offset to uppermost row to get back to correct “lane” after tunnel
Tunneled BWT - Inverting Example

\( k \)-th occurrence of character \( c \) in \( L \) corresponds to \( k \)-th occurrence of character \( c \) in \( F \)

\[
\begin{align*}
L & \quad F \\
\text{y} & \quad \$ \\
\text{e} & \quad \text{a} \\
\text{p} & \quad \text{e} \\
\$ & \quad \times \\
\text{y} & \quad \text{p} \\
\text{a} & \quad \text{s} \\
\text{s} & \quad \text{y} \\
\times & \quad \times \\
\text{s} & \\
\end{align*}
\]

\( S = \$ \)
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

\[
S = y$

$y$

$e$

$p$

$\$$

$y$

$p$

$a$

$s$

$s$

$y$

$\$$
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

\[ L \quad \text{F} \]
\[
\begin{array}{c}
\text{y} \\
\text{e} \\
\text{p} \\
\text{y} \\
\text{a} \\
\text{s} \\
\end{array}
\begin{array}{c}
\text{y} \\
\text{a} \\
\text{e} \\
\times \\
\text{p} \\
\text{s} \\
\end{array}
\]

$S = sy$
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>$$</td>
</tr>
<tr>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td>e</td>
</tr>
<tr>
<td>$$</td>
<td>$\times$</td>
</tr>
<tr>
<td>y</td>
<td>p</td>
</tr>
<tr>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>s</td>
<td>y</td>
</tr>
</tbody>
</table>

$S = \text{asy}\$
Tunneled BWT - Inverting Example

\( k\)-th occurrence of character \( c \) in \( L \) corresponds to \( k\)-th occurrence of character \( c \) in \( F \)

\[
\begin{align*}
L & \\
y & \leftarrow e \\
e & \leftarrow p \\
p & \leftarrow \$
\end{align*}
\]

\[
\begin{align*}
F & \\
$ & \leftarrow a \\
a & \leftarrow e \\
e & \leftarrow \times
\end{align*}
\]

\[
\begin{align*}
y & \leftarrow y \\
\times & \leftarrow s \\
s & \leftarrow s \\
\times & \leftarrow y
\end{align*}
\]

\[
S = \text{easy}$
\]
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

$L$
$y$ ← $e$
$e$ ← $p$
$p$ ← $s$
$s$ ←

$F$
$y$ ← $a$
$a$ ← $y$
$y$ ←

$S = \text{peasy}$
Tunneled BWT - Inverting Example

\[ k\text{-th occurrence of character c in } L \text{ corresponds to } k\text{-th occurrence of character c in } F \]

\[
\begin{align*}
L & \\
y & \leftarrow \\
e & \leftarrow \\
p & \leftarrow \\
$ & \leftarrow \\
\rightarrow \\
y & \leftarrow \\
a & \leftarrow \\
s & \leftarrow \\
\times & \leftarrow \\
F & \\
$ & \leftarrow \\
a & \leftarrow \\
e & \leftarrow \\
\times & \leftarrow \\
\end{align*}
\]

\[ S = \text{ypeasy$} \]
Tunneled BWT - Inverting Example

\(k\)-th occurrence of character \(c\) in \(L\) corresponds to \(k\)-th occurrence of character \(c\) in \(F\)

\[
L \\
y \rightarrow \cdots \\
e \\
p \\
$ \\
y \\
a \\
s \\
offset = 1
\]

\[
F \\
$ \\
a \\
e \\
$ \\
p \\
s \\
y \\
offset = 1
\]

tunnel start detected \(\Rightarrow\) switch to uppermost row

\[
S = \text{sypeasy}$
\]
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

$L$

$\rightarrow$

$F$

$y \leftarrow \underbrace{\hphantom{a e p}}_{13/25} s y e a p e s y$

$\leftarrow$

$\times$

$\times$

$S = \text{asypeasy}\$
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

L
y ← e ← p ← $ ← y ← a ← s ← s

F
$ ← a ← e ← x ← p ← s ← y ← y

offset = 1

$S = \text{easypeasy}$$
Tunneled BWT - Inverting Example

$k$-th occurrence of character $c$ in $L$ corresponds to $k$-th occurrence of character $c$ in $F$

end →

$tunnel$ end detected ⇒ switch back using offset

offset = 1

\[ S = easypeasy$ \]
Tunneled BWT Inverting - Recap

Normal BWT

Tunneled BWT

▶ uppermost row is used for all rows of a block
▶ offset is stored to "get back" to correct lane

Block Collisions

Compensable

Critical

▶ compensable collisions: cross overlay
▶ offset is stored on a stack
Consider only width-maximal run-based blocks: block height is equal to the height of runs it starts and ends in. 

\[ \text{run} \overset{\Delta}{=} \text{length-maximal repeat of same character} \]

\[ \cdots \underline{aaxx} \underline{xxxx} \underline{xbbc} \underline{aaaaaa} \underline{cc} \cdots \]

Result:

- only compensable collisions
- bitvectors cntL and cntF can be merged to one vector aux with alphabet size 3
- aux can be shortened to work run-based: only 1 symbol per run required
Practice: Block Choice

- choice depends on compression of L and aux
- L and aux come from the same source
  \[ \Rightarrow \text{compress both with same BWT backend encoder} \]
- allows to abstract choice from used backend encoder

Greedy run-length-encoding strategy

- encoding size of run-length-encoded L and aux can be estimated
- greedy strategy: assign each block a score (\( \hat{=} \) number of bits removed from L-encoding)
  - choose block with highest score
  - decrease score of colliding blocks with lower score
- result: “sorted list” of blocks
- tunnel score-highest blocks which give best tradeoff between benefit and aux encoding size
- works good as long as backend encoders also use run-length-encoding (or something similar)
Experiments: Overview

BWT compressors enhanced with tunneling

- **bwz**: original scheme by Burrows & Wheeler ($\approx$ bzip2)
- **bcm**: one of the best open-source BWT compressors
- **wt**: wavelet tree using hybrid bitvectors

Test Data

- Silesia Corpus: contains 12 files (6 - 49 MB)
- Pizza & Chili Corpus: contains 6 files (54 - 1130 MB)
- Repetitive Corpus: contains 9 files (45 - 446 MB)
Comparison: normal vs. tunneled BWT

- average encoding size decrease about 8 – 16 %
- peak encoding size decrease about 33 – 58 %
Comparison to other Compressors

- **xz**: uses LZMA, similar to **7-zip**
- **zpaq**: uses context mixing
- all values are measured in bits per symbol

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Silesia Corpus</th>
<th>Pizza &amp; Chili Corpus</th>
<th>Repetitive Corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nci (32 MB)</td>
<td>samba (21 MB)</td>
<td>webster (40 MB)</td>
</tr>
<tr>
<td>bwz</td>
<td>0.34</td>
<td>1.81</td>
<td>1.48</td>
</tr>
<tr>
<td>bwz-tunneled</td>
<td>0.33</td>
<td>1.75</td>
<td>1.48</td>
</tr>
<tr>
<td>bcm</td>
<td>0.29</td>
<td>1.49</td>
<td>1.24</td>
</tr>
<tr>
<td>bcm-tunneled</td>
<td><strong>0.28</strong></td>
<td>1.42</td>
<td>1.24</td>
</tr>
<tr>
<td>wt</td>
<td>0.61</td>
<td>2.70</td>
<td>2.08</td>
</tr>
<tr>
<td>wt-tunneled</td>
<td>0.54</td>
<td>2.45</td>
<td>2.07</td>
</tr>
<tr>
<td>xz</td>
<td>0.35</td>
<td>1.38</td>
<td>1.61</td>
</tr>
<tr>
<td>zpaq</td>
<td>0.36</td>
<td><strong>1.20</strong></td>
<td><strong>1.21</strong></td>
</tr>
</tbody>
</table>
Conclusion

Tunneling works nice...

- natural way to extend context-based compression to longer strings
- significant BWT compression improvement
- same or less resource requirements for decoding BWT

... but has some problems:

- block choice under collisions is not always optimal
- current block choice strategy is too complicated
- heavy resource requirements for encoding (memory peak and time double)

Future research goals

- try simpler block choice strategies
- examine hardness of optimal block choice
- prepare tunneling for text indexing
Questions
References

Uwe Baier.  
Tunneled BWT Implementation and Benchmark.  
https://github.com/waYne1337/tbwt.  
last visited January 2018.

Uwe Baier.  

Michael Burrows and David J Wheeler.  
A block-sorting lossless data compression algorithm.  
References II

Sebastian Deorowicz.
Silesia Corpus.
last visited January 2018.

Paolo Ferragina and Gonzalo Navarro.
Pizza & Chili Corpus.
last visited January 2018.

Paolo Ferragina and Gonzalo Navarro.
Repetitive Corpus.
last visited January 2018.
Luca Foschini, Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter.
When Indexing Equals Compression: Experiments with Compressing Suffix Arrays and Applications.

Simon Gog.
sdsl-lite *Library*.
last visited January 2018.

David A. Huffman.
A Method for the Construction of Minimum-Redundancy Codes.


B. Ya Ryabko.
Data compression by means of a “book stack”.

Tukaani.
xz File Compressor.
https://tukaani.org/xz/.
last visited January 2018.