A Memory-Efficient Data Structure for Pattern Matching in DNA with Backward Search

Dominik Kopczynski *  
Sven Rahmann*†

*Collaborative Research Center SFB 876, TU Dortmund, Germany  
Dominik.Kopczynski@tu-dortmund.de
†Genome Informatics, Institute of Human Genetics, Faculty of Medicine, University of Duisburg-Essen, Germany  
Sven.Rahmann@uni-due.de

Since backward search was introduced by Ferragina and Manzini, it became a standard index-based linear-time exact pattern search technique [2]. Due to the inherently high memory usage of its auxiliary tables, we developed a data structure that provides at least a 20-fold data reduction of memory usage without increasing computation time significantly.

Backward search

Given: A text \( T \) with \( n = |T| \) and pattern \( P \) with \( m = |P| \) over a finite alphabet \( \Sigma \). Task: Find all occurrences of \( P \) in \( T \) in \( O(m) \) time.

Example

\[ T = \text{GCTATGATAGTCAT}$ \]

Backward search uses trick by exploiting sorted order of characters in Burrows-Wheeler transform [1] of \( T \).

\[ T_{\text{bwt}} = \text{TTCGTTGTAACGA} \]

Auxiliary tables

Two auxiliary tables must be pre-computed.

• \( \text{less} \): table of size \( |\Sigma| \), where \( \text{less}[c] \) is the number of characters in \( T \) lexicographically smaller than \( c \).

• \( \text{occ} \): \( |\Sigma| \times n \) matrix containing the number of \( c \)'s in the BWT up to (and including) index \( d \).

Pattern matching

Process the pattern backwards character by character and update the suffix array interval \([L, R]\) that points to the occurrences of the processed pattern suffix at each step. The update step is

\[
L'(c) = \text{less}[c] + \text{occ}[c][L - 1], \\
R'(c) = \text{less}[c] + \text{occ}[c][R] - 1.
\]

Interval \([L, R]\) determines positions in suffix array where all prefixes equal \( P \). This process takes \( m \) iterations.

Bioinformatics context

Backward search is widely used in read mappers like BWA [3]. Either searching for exact seeds or error-tolerant alignment is possible. Since double-stranded DNA contains about \( n \approx 6.2 \times 10^9 \) bases, the \( \text{occ} \) table (int-typed entries of 4 byte each) reaches about

\[ |\Sigma| \cdot n \cdot 4 \text{ bytes} \approx 100 \text{ Gbytes}. \]

Challenge: Reduce memory usage without increasing runtime.

Improved backward search

Idea: Lots of information in \( \text{occ} \) table are redundant. Difference of following entries is at most 1. Store information where character appears bitwise in \( \text{app} \) table.

\[
c \quad \text{app}[c] \\
A 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1 \\
C 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 \\
G 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 \\
T 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
\]

\[ T_{\text{bwt}} = T \ T \ C \ G \ T \ T \ G \ T \ S \ A \ A \ A \ C \ G \ A \]

Bits are stored in blocks of \( \text{long int} \). Trick: using hardware implemented command \( \text{popcnt} \) to count bits in block and additionally store summed up occurrences in \( \text{occ} \) table for every \( r \)-th entry. Typically the register size (\( \text{long int} \)) \( r = 64 \) is suitable.

**Determination of new interval**

• Given: Interval \( L, R \) from previous iteration and current char \( c \)
• Determine \( \text{long int} \) block: \( \text{block} = L \gg 6 \)
• Determine the \( i \) significant bits in the block: \( i = L \& 63 \)

\[
1. \text{block} \cdot L; \text{block} \cdot R \gg 6 \\
2. \text{if} \ L > 0: \\
3. \text{block} \cdot L = (L - 1) \gg 6 \\
4. \text{appear} \cdot L = \text{popcnt}([\text{app}[c][\text{block} \cdot L] \ll (63 - ((L - 1) \& 63))) \\
5. \text{appear} \cdot r = \text{popcnt}([\text{app}[c][\text{block} \cdot L] \ll (63 - (R \& 63))) \\
6. \text{if} \text{block} \cdot L > 0: \text{occur} \cdot L = \text{occ}[c][\text{block} \cdot L - 1] \\
7. \text{if} \text{block} \cdot R > 0: \text{occur} \cdot R = \text{occ}[c][\text{block} \cdot R - 1] \\
8. L = \text{less}[c] + \text{appear} \cdot L + \text{occur} \cdot L \\
9. R = \text{less}[c] + \text{appear} \cdot R + \text{occur} \cdot R - 1
\]

Sampling \( \text{occ} \) table by storing only every \( k \)-th entry provides additional data reduction. Typically \( k = 4 \) is suitable. Code does not get more difficult.

Memory

Using the double-stranded DNA again we get a memory usage of

\[
\text{app}: \ |\Sigma| \cdot n \cdot \frac{1}{k} \text{ bytes} \approx 3.1 \text{ Gbytes} \\
\text{occ}: \ |\Sigma| \cdot n \cdot \frac{4}{k^2} \text{ bytes} \approx 0.4 \text{ Gbytes} \approx 3.5 \text{ Gbytes}.
\]

References