technische universität dortmund

SFB 876 Verfügbarkeit von Information durch Analyse unter Ressourcenbeschränkung

# 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 occurences of $P$ in $T$ in $\mathcal{O}(m)$ time.

### Example

$$T = \texttt{GCTATGATAGTCAT\$}$$

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

$$T_{\text{bwt}} = \texttt{TTCGTTGT\$AAACGA}$$

### Auxiliary tables

Two auxiliary tables must be pre-computed.

- `less`: table of size $|\Sigma|$, where `less`$[c]$ is the number of characters in $T$ lexicographically smaller than $c$
- `occ`: $|\Sigma| \times n$ matrix containing the number of $c$s in the BWT up to (and including) index $d$

| $c$ | less[$c$] | occ[$c$] | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 1, | 2, | 3, | 3, | 3, | 4 |
| C | 5 | 0, | 0, | 1, | 1, | 1, | 1, | 1, | 1, | 1, | 1, | 1, | 1, | 2, | 2, | 2 |
| G | 7 | 0, | 0, | 0, | 1, | 1, | 1, | 2, | 2, | 2, | 2, | 2, | 2, | 2, | 3, | 3 |
| T | 10 | 1, | 2, | 2, | 2, | 3, | 4, | 4, | 5, | 5, | 5, | 5, | 5, | 5, | 5, | 5 |
| $T_{\text{bwt}}$ | = | T | T | C | G | T | T | G | T | $ | A | A | A | C | G | A |

### 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

$$L^+(c) = \texttt{less}[c] + \texttt{occ}[c][L-1],$$
$$R^+(c) = \texttt{less}[c] + \texttt{occ}[c][R] - 1.$$

Interval $[L, R]$ determines positions in suffix array where all prefixes equal $P$. This process takes $m$ iteratiorns.

### 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 $n \approx 6.2 \cdot 10^9$ bases, the `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 `occ` table are redundant. Difference of following entries is at most 1. Store information where character appears bitwise in `app` table.

| $c$ | app[$c$] | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 1, | 1, | 1, | 0, | 0, | 1 |
| C | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 1, | 0, | 0 |
| G | 0, | 0, | 0, | 1, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0, | 0, | 1, | 0 |
| T | 1, | 1, | 0, | 0, | 1, | 1, | 0, | 1, | 0, | 0, | 0, | 0, | 0, | 0, | 0 |
| $T_{\text{bwt}}$ = | T | T | C | G | T | T | G | T | $ | A | A | A | C | G | A |

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

| $c$ | occ[$c$] | | |
|---|---|---|---|
| A | 0, | 0, | 3 |
| C | 1, | 1, | 1 |
| G | 1, | 2, | 2 |
| T | 2, | 5, | 5 |
| $T_{\text{bwt}}$ = | T T C G | T T G T | $ A A A C G A |

(Example: $r = 4$)

### Determination of new interval

- Given: interval $L, R$ from previous iteration and current char $c$
- Determine `long` block in `app` table: `block = L >> 6`
- Determine the $i$ significant bits in the block: `i = L & 63`

```
block_l = 0; block_r = R >> 6
if L > 0:
    block_l = (L - 1) >> 6
    appear_l = popcnt(app[c][block_l] << (63 - ((L - 1) & 63)))
appear_r = popcnt(app[c][block_r] << (63 - (R & 63)))
if block_l > 0: occur_l = occ[c][block_l - 1]
if block_r > 0: occur_r = occ[c][block_r - 1]

L = less[c] + appear_l + occur_l
R = less[c] + appear_r + occur_r - 1
```

Sampling `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

| | | |
|---|---|---|
| app: | $\lvert\Sigma\rvert \cdot n \cdot \frac{1}{8}$ bytes | $\approx 3.1$ Gbytes |
| occ: | $\lvert\Sigma\rvert \cdot n \cdot \frac{4}{r \cdot k}$ bytes | $\approx 0.4$ Gbytes |
| | | $\approx 3.5$ Gbytes. |

## References

[1] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, CA, 1994.

[2] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[3] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.