An Automaton-Based View on Error-Tolerant Pattern Matching with Backward Search

Dominik Kopczynski\textsuperscript{*} Sven Rahmann\textsuperscript{†}

\textsuperscript{*}Collaborative Research Center SFB 876, Computer Science XI, TU Dortmund, Germany dominik.kopczynski@tu-dortmund.de
\textsuperscript{†}Genome Informatics, Institute of Human Genetics, Faculty of Medicine, University of Duisburg-Essen, Germany sven.rahmann@uni-due.de

Introduction: Backward search is used as a computational core in many read mapping applications in the context of next generation sequencing data analysis. Here we introduce an automaton-based view on error-tolerant backward search by combining the non-deterministic finite automaton from the error-tolerant NFA with exact backward search. This leads to a conceptually simple, efficient and easily implementable version of error-tolerant backward search.

Input: text \(T\), pattern \(P, n = |T|, m = |P|, k\) errors at most
Output: all occurrences of \(P\) in \(T\) with \(0 \leq i \leq k\) errors

Error-tolerant NFA in \(O(k \cdot n)\):

<table>
<thead>
<tr>
<th>k=0</th>
<th>k=1</th>
<th>k=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ</td>
<td>Σ</td>
<td>Σ</td>
</tr>
<tr>
<td>ΣC</td>
<td>ΣC</td>
<td>ΣC</td>
</tr>
<tr>
<td>ΣG</td>
<td>ΣG</td>
<td>ΣG</td>
</tr>
<tr>
<td>ΣT</td>
<td>ΣT</td>
<td>ΣT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exact Backward Search \cite{1} in \(O(m)\):
• Uses suffix array \(pos\) of \(T\) and Burrows-Wheeler transform (BWT)
• Needs auxiliary tables:
  – \(less[c]\): number of characters in \(T\) lexicographically smaller than \(c\)
  – \(occ[c][r]\): number of \(c\)'s in BWT up to index \(r\)
• Updates an interval containing possible suffixes in \(pos\)
• Starts with whole interval \(L = 0, R = n - 1\) for empty pattern
• Updates interval processing reversed pattern, using:
  \(L^*(c) = less[c] + occ[c][|L - 1|]\)
  \(R^*(c) = less[c] + occ[c][|R - 1|]\)

Automaton-based error-tolerant Backward Search:
• Initialize empty matrix \(M\) with \((k + 1) \times (m + 1)\) nodes
• Use reversed pattern \(P'\)
• Store full interval \([0, n - 1]\) in node \(M[0][0]\)
• For every interval in every node:
  – If after BS update new interval is valid, perform:
    • A match with \(c = P'[j]\) and store in \(M[i][j + 1]\)
    • An insertion with \(c \in \Sigma\) and store in \(M[i + 1][j]\)
    • A substitution with \(c \in \Sigma\setminus P'[j]\) and store in \(M[i + 1][j + 1]\)
  – Perform a deletion, store current interval in \(M[i + 1][j + 1]\)
• Example:
  – Text: AAAACGTCCT, pattern: ACTGT, \(k = 2\)
  – No exact match, one match with single error, four matches with two errors

Memory saving:
• Only two columns needed, current and subsequent column
• After processing current column all important data stored in subsequent column

Traceback:
• Needs complete matrix \(M\)
• Is applicable without considering \(pos\) and BWT after processing
• Auxiliary data must be stored per interval:
  – Its ancestor interval
  – Operation it was computed (mat, ins, del, sub)
  – Character involved in operation

Reasonable improvements for read mapping:
• Omit computation of first column, exponential growing, insertions at the left and right of a read not reasonable
• Restrict error bound for the first \(j\) matches
• Precompute lower bound errors for every suffix in \(P'\) (consider \(D()\) array in \cite{2})

Conclusion: We presented a novel view on error-tolerant pattern matching using backward search, combining error-tolerant NFA with backward search. Certain improvements lead to a dramatically acceleration of computation time. This method is additionally well suited e.g. for teaching in class.

References