Improved Approximate String Matching and Regular Expression Matching on Ziv-Lempel Compressed Texts

Philip Bille
IT University of Copenhagen

Rolf Fagerberg
University of Southern Denmark

Inge Li Gørtz
Technical University of Denmark
Approximate String Matching

• The *edit distance* between two strings is the minimum number of insertions, deletions, and substitutions needed to convert one string to the other. E.g., edit-distance(“cocoa”, “cola”) = 2.

• Let $P$ and $Q$ be strings and let $k$ (integer $> 0$) be an *error threshold*.

• The approximate string matching problem is to find all ending positions of substrings in $Q$ whose edit distance to $P$ is at most $k$. 
## Results

| Time                        | Space         | Reference      |
|-----------------------------|---------------|----------------|
| $O(um)$                     | $O(m)$        | [Sellers1980]  |
| $O(uk)$                     | $O(m)$        | [LV1989]       |
| $O\left(\frac{uk^4}{m} + u\right)$ | $O(m)$        | [CH2002]       |

$|P| = m$ and $|Q| = u$
Ziv-Lempel 1978 compression

\[ Q = \text{ananas} \]

\[ Z = (0, a)(0, n)(1, n)(1, s) \]

\[ Z = Z_0 Z_1 Z_2 Z_3 Z_4 \]
Approximate String Matching on ZL78 compressed texts

- Let \( P \) be a string and \( Z \) be a ZL78 compressed representation of a string \( Q \).

- Given \( P \) and \( Z \), the \textit{compressed approximate string matching problem} is to solve the approximate string matching for \( P \) and \( Q \) without decompressing \( Z \).

- Goal: Do it more efficiently than decompressing \( Z \) and using the best (uncompressed) approximate string matching algorithm.
Applications

• Textual data bases (e.g. DNA sequence collections) issues:

  • Save space = keep data in compressed form.

  • Search efficiently.

• Solution: Compressed string matching algorithms.
Results

• Let $|P| = m$ and $|Z| = n$.

• Kärkkäinen, Navarro, and Ukkonen [KNU2003]:
  
  • $O(nmk + \text{occ})$ time and $O(nmk)$ space.

• Our result (Theorem 1): For any parameter $\tau \geq 1$:
  
  • $O(n(\tau + m + t(m, 2m + 2k, k)) + \text{occ})$ expected time and
  
  • $O(n/\tau + m + s(m, 2m + 2k, k)) + \text{occ}$ space.
Example Results

| Time                      | Space                               | Reference       |
|---------------------------|-------------------------------------|-----------------|
| \(O(nmk + \text{occ})\)  | \(O(nmk)\)                         | [KNU2003]       |
| \(O(nmk + \text{occ})\)  | \(O\left(\frac{n}{mk} + m + \text{occ}\right)\) | LV + \(\tau = mk\) |
| \(O(nk^4 + nm + \text{occ})\) | \(O\left(\frac{n}{k^4 + m} + m + \text{occ}\right)\) | CH + \(\tau = k^4 + m\) |

\(|P| = m\) and \(|Z| = n\)
Selecting Compression Elements

For parameter $\tau \geq 1$, select a subset $C$ of the compression elements of $Z$ such that:

- $|C| = O(n/\tau)$.
- From any compression element $z_i$, the distance (minimum number of references) to any compression element in $C$ is at most $2\tau$. 
Selecting Compression Elements

- Maintain $C$ using dynamic perfect hashing while scanning $Z$ from left-to-right.
- Initially, set $C = \{z_0\}$.
- To process element $z_{i+1}$ follow references until we encounter $y \in C$:
  - If the distance $l$ from $z_{i+1}$ to $y$ is less than $2\tau$ we are done.
  - Otherwise ($l = 2\tau$), insert element the element at distance $\tau$ into $C$. 
Selecting Compression Elements

\[ Z = \]

- Lemma: For any parameter \( \tau \geq 1 \), \( C \) is constructed in
  
  - \( O(n\tau) \) expected time and
  
  - \( O(n/\tau) \) space.
Computing Matches

\[ Q = \cdots \begin{array}{|c|c|} \hline \text{phrase}(z_{i-1}) & \text{phrase}(z_i) \ \hline \end{array} \cdots \]

- Strategy:
  - Process \( Z \) from left-to-right.
  - At \( z_i \) we compute all matches ending in the substring encoded by \( z_i \).
Computing Overlapping Matches

Let \([u_i, u_i + l_i - 1]\) be the positions in \(Q\) of \(\text{phrase}(z_i)\).

Goal: Find all overlapping matches for \(z_i\), i.e., the matches starting before \(u_i\) and ending in \([u_i, u_i + l_i - 1]\).

Decompress substrings \(\text{rpre}(z_i)\) and \(\text{rsuf}(z_{i-1})\) of length \(m + k\) around \(u_i\).

Run favorite (uncompressed) approximate string matching algorithm to find matches of \(P\) in \(\text{rsuf}(z_{i-1}) \cdot \text{rpre}(z_i)\). Add offset to these to get the overlapping matches for \(z_i\).
Computing the Relevant Prefix and Suffix

• For parameter $\tau \geq 1$, select a subset $C$ of the compression elements of $Z$ according to Lemma 1.

• For each element in $C$ at distance more than $m+k$ from $z_0$ add “shortcut” to element at distance $m+k$. 
Computing the Relevant Prefix

- Follow references to nearest element in $C$.
- Follow shortcut if present.
- Compute the relevant prefix by decompressing length $m + k$ substring.
Computing the Relevant Suffix

- Follow references to decompress substring of length $m + k$.
- If the phrase is shorter than $m + k$, recursively apply to $z_{i-1}$ until we have $m + k$ characters.
Analysis

- Time = preprocess + n(find nearest element + decompress + match) =
  
  \( O(n\tau + n(\tau + m + t(m, 2m + 2k, k))) \)

- Space = preprocess + decompress + match =
  
  \( O(n/\tau + m + s(m, 2m + 2k, k)) \)
Computing Internal Matches

- **Goal:** Find all *internal matches* for $z_i$, i.e., all matches starting and ending within $[u_i, u_i + l_i - 1]$.

- Compute and store all the internal match sets indexed by compression elements using dynamic perfect hashing.

- Decompress substring $rsuf'(z_i)$ of length $\min(l_i, m + k)$ ending at $u_i + l_i - 1$.

- Internal matches for $z_i =$

  \[
  (\text{internal matches for } reference(z_i)) \cup (\text{matches of } P \text{ in } rsuf'(z_i))
  \]
Analysis

• Time = n (decompress + match + internal matches) =

\[ O(n(m + t(m, m + k, k)) + \text{occ}) \]

• Space = decompress + match + total number of internal matches =

\[ O(m + s(m, m + k, k) + \text{occ}) \]
Putting the Pieces Together

• Merging overlapping and internal matches we get all matches for $z_i$ ending within $[u_i, u_i + l_i - 1]$.

• Implies Theorem 1: For any parameter $\tau \geq 1$:
  
  • $O(n(\tau + m + t(m, 2m + 2k, k)) + \text{occ})$ expected time and
  
  • $O(n/\tau + m + s(m, 2m + 2k, k)) + \text{occ})$ space.

• Does not hold for ZLW compressed texts, unless $\Omega(n)$ space is used.

• For $\Omega(n)$ space the bounds hold in the worst-case and work for both ZL78 and ZLW.
Regular Expression Matching

• A regular expression is a generalized pattern composed from simple characters using union, concatenation, and Kleene star.

• Given a regular expression $R$ and a string $Q$ the regular expression matching problem is to find all ending positions of substrings in $Q$ that matches a string in the language generated by $R$. 
Regular Expression Matching

• Let $|R| = m$ and $|Q| = u$.

• Classic solution [Thompson1968]: $O(um)$ time and $O(m)$ space.

• Several improvements based on the Four-Russian technique or word-level parallelism [Myers1992, NR2004, BFC2005, Bille2006].
Compressed Regular Expression Matching

• Let $|R| = m$ and $|Z| = n$.

• Navarro [Navarro2003] simplified and without word-level parallel techniques:
  • $O(nm^2 + \text{occ} \cdot m \log m)$ time and $O(nm^2)$ space.

• Our result (Theorem 2): For any parameter $\tau \geq 1$:
  • $O(nm(m + \tau) + \text{occ} \cdot m \log m)$ time and
  • $O(nm^2 / \tau + nm)$ space.

• E.g. $\tau = m$ gives $O(nm^2 + \text{occ} \cdot m \log m)$ time and $O(nm)$ space.
Remarks

• Compressed strings are large and therefore $\Omega(n)$ space may not be feasible for large texts.

• Our result for compressed approximate string matching is one of the very few algorithms for compressed matching that uses $o(n)$ space.

• More sublinear space compressed string matching algorithms are needed!