A Linear Size Index for Approximate String Matching

CPM 2006

Siu-Lung Tam

Joint work with:
Ho-Leung Chan Tak-Wah Lam
Wing-Kin Sung Swee-Seong Wong

2006-07-05 11:10 +0200

Indexing for Approximate String Matching

Problem

Index a text $S[1..n]$ over a constant-size alphabet Σ w.r.t. an integer constant k.
For each pattern $P[1..m]$, search for all $S[i..n]$ such that $\exists\, j \geq i \;\; dist(P, S[i..j]) \leq k$.

- We focus on Hamming distance in this talk, although edit distance works similarly.

Indexing for Approximate String Matching

Problem

Index a text $S[1..n]$ over a constant-size alphabet Σ w.r.t. an integer constant k.
For each pattern $P[1..m]$, search for all $S[i..n]$ such that $\exists\, j \geq i \;\; dist(P, S[i..j]) \leq k$.

- We focus on Hamming distance in this talk, although edit distance works similarly.
- Performance is measured by both
  (i) index size, and
  (ii) searching time, which can depend on
    - text size n,
    - alphabet size $|\Sigma|$,
    - number of errors k,
    - pattern length m,
    - number of matches occ

Brute Force Searching

- When m, $|\Sigma|$ and k are small ...

Brute Force Searching

- When m, $|\Sigma|$ and k are small ...
- Searching $P[1..m]$ for 1 error = Searching $m(|\Sigma| - 1)$ "1-modified patterns"

Brute Force Searching

- When m, $|\Sigma|$ and k are small ...
- Searching $P[1..m]$ for 1 error = Searching $m(|\Sigma| - 1)$ "1-modified patterns"
- Suppose it takes $O(f(m,n) + occ')$ time to search each 1-modified pattern for exact match.
  - Nowadays $f(m,n)$ is $O(\log n)$ or even $O(\log \log n)$.

Brute Force Searching

- When m, $|\Sigma|$ and k are small ...
- Searching $P[1..m]$ for 1 error = Searching $m(|\Sigma| - 1)$ "1-modified patterns"
- Suppose it takes $O(f(m,n) + occ')$ time to search each 1-modified pattern for exact match.
  - Nowadays $f(m,n)$ is $O(\log n)$ or even $O(\log \log n)$.
- $O(m|\Sigma|\ f(m,n) + occ)$ time to search P for 1 error.
  $O(m^k|\Sigma|^k\ f(m,n) + occ)$ time to search P for k errors.

1-Error Solutions

Space	Time	Authors
$O(n)$	$O(m^2 + occ)$	Cobbs [CPM95]

1-Error Solutions

Space	Time	Authors
$O(n)$	$O(m^2 + occ)$	Cobbs [CPM95]
$O(n \log^2 n)$	$O(m \log n \log \log n + occ)$	Amir et al. [WADS99]
$O(n \log n)$	$O(m \log \log n + occ)$	Bauchsbaum et al. [ESA00]

1-Error Solutions

Space	Time	Authors
$O(n)$	$O(m^2 + occ)$	Cobbs [CPM95]
$O(n \log^2 n)$	$O(m \log n \log \log n + occ)$	Amir et al. [WADS99]
$O(n \log n)$	$O(m \log \log n + occ)$	Bauchsbaum et al. [ESA00]
$O(n \log n)$	$O(m + \log n \log \log n + occ)$	Cole et al. [STOC04]

1-Error Solutions

Space	Time	Authors
$O(n)$	$O(m^2 + occ)$	Cobbs [CPM95]
$O(n \log^2 n)$	$O(m \log n \log \log n + occ)$	Amir et al. [WADS99]
$O(n \log n)$	$O(m \log \log n + occ)$	Bauchsbaum et al. [ESA00]
$O(n \log n)$	$O(m + \log n \log \log n + occ)$	Cole et al. [STOC04]
$O(n)$	$O(m \log n + occ)$	Huynh et al. [CPM04]
$O(n)$ *bits*	$O((m \log n + occ) \log n)$	Huynh et al. [CPM04]

1-Error Solutions

Space	Time	Authors
$O(n)$	$O(m^2 + occ)$	Cobbs [CPM95]
$O(n \log^2 n)$	$O(m \log n \log \log n + occ)$	Amir et al. [WADS99]
$O(n \log n)$	$O(m \log \log n + occ)$	Bauchsbaum et al. [ESA00]
$O(n \log n)$	$O(m + \log n \log \log n + occ)$	Cole et al. [STOC04]
$O(n)$	$O(m \log n + occ)$	Huynh et al. [CPM04]
$O(n)$ *bits*	$O((m \log n + occ) \log n)$	Huynh et al. [CPM04]
$O(n)$	$O(m \log \log n + occ)$	Lam et al. [ISAAC05]
$O(n)$ *bits*	$O((m \log \log n + occ) \log^{\epsilon} n)$	Lam et al. [ISAAC05]

1-Error Solutions

Space	Time	Authors
$O(n)$	$O(m^2 + occ)$	Cobbs [CPM95]
$O(n \log^2 n)$	$O(m \log n \log\log n + occ)$	Amir et al. [WADS99]
$O(n \log n)$	$O(m \log\log n + occ)$	Bauchsbaum et al. [ESA00]
$O(n \log n)$	$O(m + \log n \log\log n + occ)$	Cole et al. [STOC04]
$O(n)$	$O(m \log n + occ)$	Huynh et al. [CPM04]
$O(n)$ *bits*	$O((m \log n + occ) \log n)$	Huynh et al. [CPM04]
$O(n)$	$O(m \log\log n + occ)$	Lam et al. [ISAAC05]
$O(n)$ *bits*	$O((m \log\log n + occ) \log^{\epsilon} n)$	Lam et al. [ISAAC05]
$O(n)$	$O(m + \log^3 n \log\log n + occ)$	Our Result
$O(n)$ *bits*	$O((m + \log^4 n \log\log n + occ) \log^{\epsilon} n)$	Our Result

k-Error Solutions in Literature

Space	Time	Authors
$O(n)$	$O((cm)^k \log n + occ)$	Huynh et al. [CPM04]
$O(n)$ *bits*	$O(((cm)^k \log n + occ) \log n)$	Huynh et al. [CPM04]
$O(n)$	$O((cm)^k \log \log n + occ)$	Lam et al. [ISAAC05]
$O(n)$ *bits*	$O(((cm)^k \log \log n + occ) \log^{\epsilon} n)$	Lam et al. [ISAAC05]

k-Error Solutions in Literature

Space	Time	Authors
$O(\frac{c^k}{k!} n \log^k n)$	$O(m + \frac{c^k}{k!} \log^k n \log\log n + occ)$	Cole et al. [STOC04]
$O(n)$	$O((cm)^k \log n + occ)$	Huynh et al. [CPM04]
$O(n)$ *bits*	$O(((cm)^k \log n + occ) \log n)$	Huynh et al. [CPM04]
$O(n)$	$O((cm)^k \log\log n + occ)$	Lam et al. [ISAAC05]
$O(n)$ *bits*	$O(((cm)^k \log\log n + occ) \log^\epsilon n)$	Lam et al. [ISAAC05]

k-Error Solutions in Literature

Space	Time	Authors
$O(\frac{c^k}{k!} n \log^k n)$	$O(m + \frac{c^k}{k!} \log^k n \log\log n + occ)$	Cole et al. [STOC04]
$O(n)$	$O((cm)^k \log n + occ)$	Huynh et al. [CPM04]
$O(n)$ *bits*	$O(((cm)^k \log n + occ) \log n)$	Huynh et al. [CPM04]
$O(n)$	$O((cm)^k \log\log n + occ)$	Lam et al. [ISAAC05]
$O(n)$ *bits*	$O(((cm)^k \log\log n + occ) \log^\epsilon n)$	Lam et al. [ISAAC05]
$O(n)$	$O(m + (c \log n)^{k(k+1)} \log\log n + occ)$	Our Result
$O(n)$ *bits*	$O((m + (c \log n)^{k(k+2)} \log\log n + occ) \log^\epsilon n)$	Our Result

High Level Description

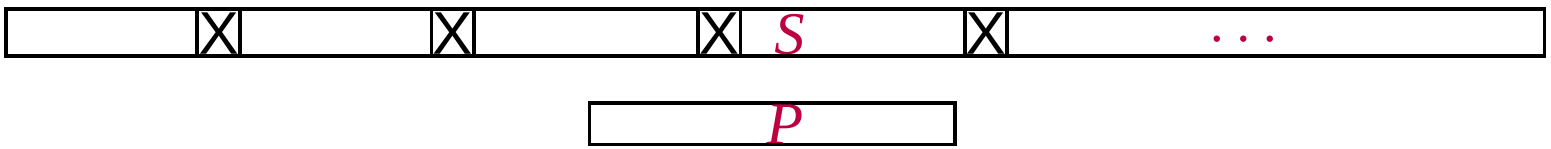
- We combine
  - "k-Errata Trees" by Cole, Gottlieb and Lewenstein [STOC04]
  - "Tree Cross Product" by Buchsbaum, Goodrich and Westbrook [ESA00], and
- If the pattern is "long-enough"
  - this index allows us to locate k-error matches efficiently.
- Otherwise
  - use the best brute-force searching, by Lam, Sung and Wong [ISAAC05].

Luxury of Long Patterns

- Let $\beta \leq |P|$ be an integer, later chosen as $\Theta(\log^{k+1} n)$.
- We mark in text S, a checkpoint at every β characters
  - e.g. $S[\beta]$, $S[2\beta]$, $S[3\beta]$, ... are checkpoints.

Luxury of Long Patterns

- Let $\beta \leq |P|$ be an integer, later chosen as $\Theta(\log^{k+1} n)$.
- We mark in text S, a checkpoint at every β characters
  - e.g. $S[\beta]$, $S[2\beta]$, $S[3\beta]$, ... are checkpoints.



Lemma

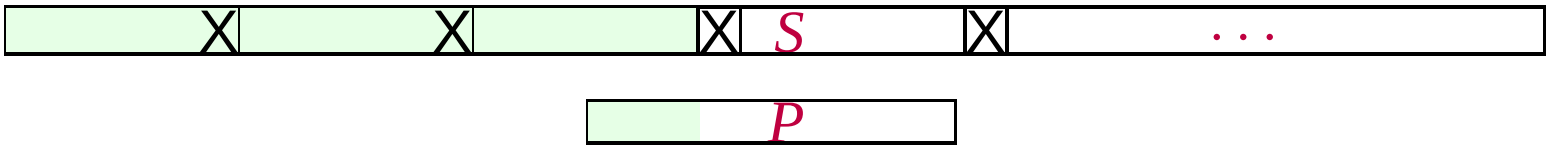
Consider a substring $S[i..j]$ that matches P with exactly k errors. $S[i..j]$ contains at least one checkpoint.
In particular, $S[i..i+\beta-1]$ contains exactly one checkpoint.

Split the Pattern

- Index S for k-error searching only around checkpoints.
- Let TAIL $= \{S[a..n] \mid a = \beta, 2\beta, ...\}$.
  and HEAD $= \{S[1..b] \mid b + 1 = \beta, 2\beta, ...\}$.

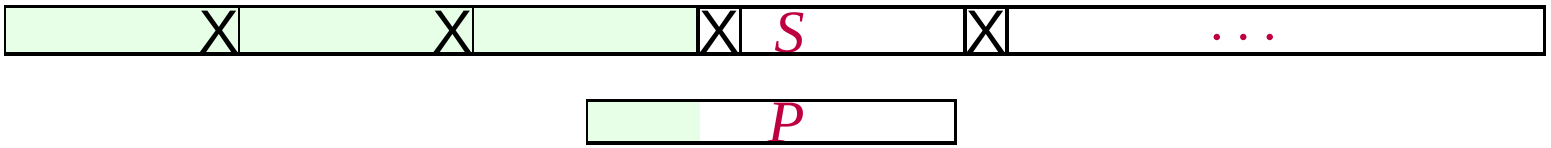
Split the Pattern

- Index S for k-error searching only around checkpoints.
- Let TAIL $= \{S[a..n] \mid a = \beta, 2\beta, ...\}$.
  and HEAD $= \{S[1..b] \mid b + 1 = \beta, 2\beta, ...\}$.



Split the Pattern

- Index S for k-error searching only around checkpoints.
- Let TAIL $= \{S[a..n] \mid a = \beta, 2\beta, ...\}$.
  and HEAD $= \{S[1..b] \mid b + 1 = \beta, 2\beta, ...\}$.



Lemma

Consider a substring $S[i..j]$ that matches P with k errors. There exist non-negative integers $k_1 + k_2 = k$, such that some string in TAIL has a prefix matching P with k_1 errors, and some string in HEAD has a suffix matching P with k_2 errors.

Note that $|\text{TAIL}| = |\text{HEAD}| = O(n/\beta)$

Dictionary Index for *TAIL*

Problem

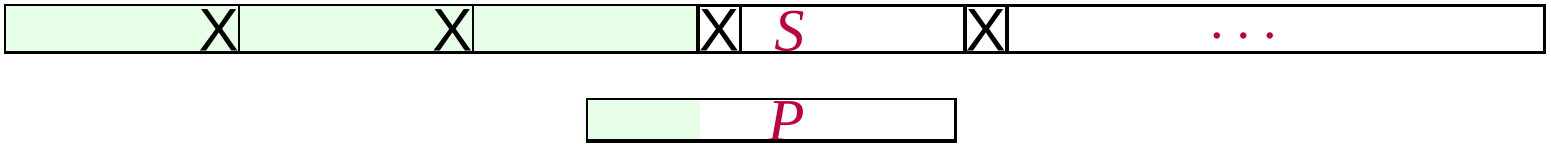
Index all strings in TAIL w.r.t. an integer ℓ. Given a query Q, search for all strings with Q' as a prefix, where $dist(Q, Q') = \ell$.

Lemma

The STOC04 index

- *is a forest of compact tries where each leaf is a suffix of S;*
- *supports searching in $O(\log^{\ell} n \log\log n)$ time, if Q is preprocessed using $O(|Q|)$ time on suffix tree of S;*
- *represents the strings found as the disjoint union over descendant leaves of $O(\log^{\ell} n)$ nodes in the compact tries;*
- *contains each suffix at most $O(\log^{\ell} n)$ times;*
- *takes $O(|TAIL| \log^{\ell} n)$ space.*

Searching Algorithm

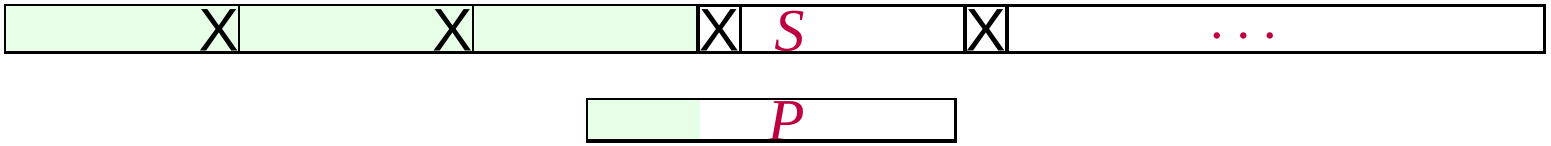


Algorithm 1 Find all k-error matches of P in S, for $|P| \geq \beta$

1: **for** i in $[1, \beta]$ {Conceptually cut P into $P[1..i–1]$ and $P[i..m]$} **do**

7: **end for**

Searching Algorithm

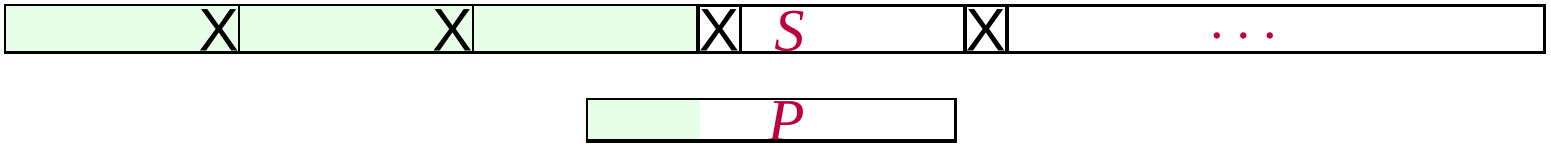


Algorithm 2 Find all k-error matches of P in S, for $|P| \geq \beta$

1: **for** i in $[1, \beta]$ {Conceptually cut P into $P[1..i–1]$ and $P[i..m]$} **do**
2: **for** k_2 in $[0, k]$ {Search for k_2 and $k_1 = k - k_2$ errors respectively} **do**

6: **end for**
7: **end for**

Searching Algorithm

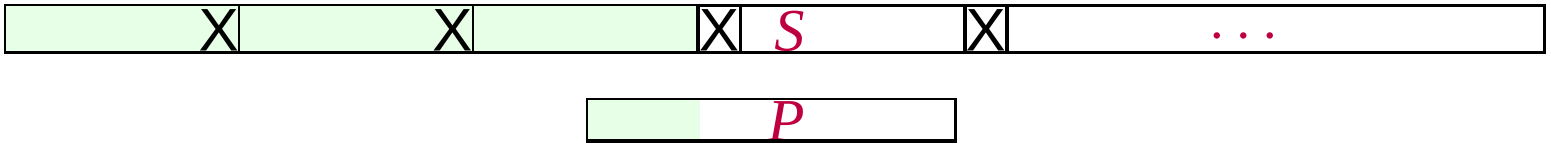


Algorithm 3 Find all k-error matches of P in S, for $|P| \geq \beta$

1: **for** i in $[1, \beta]$ {Conceptually cut P into $P[1..i–1]$ and $P[i..m]$} **do**
2: **for** k_2 in $[0, k]$ {Search for k_2 and $k_1 = k - k_2$ errors respectively} **do**
3: Find all $S[a..n] \in$ TAIL with a prefix matching $P[i..m]$ with k_1 errors.
4: Find all $S[1..b] \in$ HEAD with a suffix matching $P[1..i–1]$ with k_2 errors.

6: **end for**
7: **end for**

Searching Algorithm



Algorithm 4 Find all k-error matches of P in S, for $|P| \geq \beta$

1: **for** i in $[1, \beta]$ {Conceptually cut P into $P[1..i–1]$ and $P[i..m]$} **do**
2: **for** k_2 in $[0, k]$ {Search for k_2 and $k_1 = k - k_2$ errors respectively} **do**
3: Find all $S[a..n] \in$ TAIL with a prefix matching $P[i..m]$ with k_1 errors.
4: Find all $S[1..b] \in$ HEAD with a suffix matching $P[1..i–1]$ with k_2 errors.
5: Find all "connecting pairs": $S[a..n]$ and $S[1..b]$ where $a = b + 1$.
6: **end for**
7: **end for**

Tree Cross Product

Problem

Index a set $I \subseteq V(T_1) \times V(T_2)$. Given a query (u, w), search for all $(x, y) \in I$ where $x \in descendants(u)$ and $y \in descendants(w)$.

Lemma

The ESA00 index

- *supports searching in $O(\log \log n + occ)$ time;*
- *takes $O(|I| \log |V|)$ space, where $V = V(T_1) \cup V(T_2)$.*

Tree Cross Product

Problem

Index a set $I \subseteq V(T_1) \times V(T_2)$. Given a query (u, w), search for all $(x, y) \in I$ where $x \in descendants(u)$ and $y \in descendants(w)$.

Lemma

The ESA00 index

- *supports searching in $O(\log \log n + occ)$ time;*
- *takes $O(|I| \log |V|)$ space, where $V = V(T_1) \cup V(T_2)$.*

- A pair of leaves is in I if they represent $S[a..n]$ and $S[1..b]$ respectively where $a = b + 1$.
- Each (a, b) has $O(\log^{k_1} n)$ and $O(\log^{k_2} n)$ leaves respectively. $|I| = O(\frac{n}{\beta} \log^{k_1} n \log^{k_2} n) = O(\frac{n}{\beta} \log^{k} n)$.

Searching Complexity

- By setting $\beta = \Theta(\log^{k+1} n)$, we obtained an $O(n)$ space index.
- $|P| \geq \beta$:
  - It takes $O(\log^k n \log\log n + occ')$ time to search STOC04 and ESA00 indexes.
  - It takes $O(m)$ time to preprocess P to enable the above searching.
  - Searching time is $O(m+\beta \log^k n \log\log n+occ) = O(m+\log^{2k+1} n \log\log n+occ)$.
- $|P| < \beta$:
  - Searching time is $O((cm)^k \log\log n + occ) = O((c\log n)^{k(k+1)} \log\log n + occ)$.

Corollaries

- If we pick $\beta = \Theta(\log^{k+2} n)$ and use a compressed suffix tree to replace the suffix tree in [STOC04], we obtain an index using $O(n)$ bits.
- Space-time tradeoff is possible by choosing a smaller β and using the brute-force searching technique for some of the k errors.
- Extention to edit distance is possible but each k-error match maybe reported multiple times.

Future Directions

- Make a better trade-off — the gap in time complexity between long and short patterns
- Improve "output complexity" of [STOC04]
- Improve space complexity of [ESA00]
- Improve searching complexity for short patterns
  - Our recent ESA 2006 paper uses $O(n)$ space to allow k-error searching in $O((cm)^{k-1} \log n \log\log n + occ)$ time.
  - Using this result, searching time is reduced to $O(m + \log^{k^2} n \log\log n + occ)$.