Approximate Sequence Matching Algorithms to Handle Bounded Number of Errors

Neda Tavakoli
Computational Science and Engineering
Georgia Institute of Technology

StringBio Workshop
University of Central Florida
10/27/18

Outline

- Problem of Approximate Sequence Matching
- Type of Solutions:
  - Solution based on Exact Sequence Matching
    - Pigeonhole principle
  - Solutions based on dynamic programming
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Filters
  - Solutions based on Deterministic Automata
  - Solutions based on Bit Parallelism (parallelize another algorithm using bits)
  - Solutions based on Indexing and/or Dynamic programing
- Summary

Problem

Exact Matching:

- Given a text string T of length n and a pattern string P of length m, the exact string matching problem is to find all occurrences of P in T.

Approximate Matching

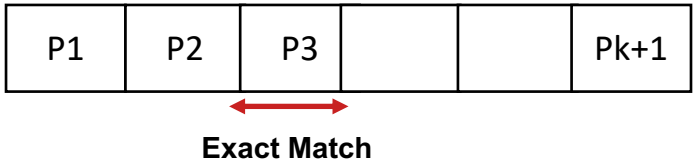
- Given a text string T of length n and a pattern string P of length m, the approximate string matching problem is to find all “almost”- occurrences of P in T.
  - Allow mismatches (substitution): Hamming distance
  - Allow insertion/deletion/substitution: edit distance

Outline

- Problem of Approximate Sequence Matching
- Type of Solutions:
  - Solution based on Exact Sequence Matching
    - Pigeonhole principle
  - Solutions based on dynamic programming
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Filters
  - Solutions based on Deterministic Automata
  - Solutions based on Bit Parallelism (parallelize another algorithm using bits)
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Indexing and/or Dynamic programing
- Summary

Based on Exact Matching

- Pigeonhole principle: If we have “k” locations and “k+1” pigeons, at least one location must have more than one pigeon.
- Find a bridge between exact matching and approximate matching, to handle “k” errors:
  - Divide the pattern “P” to “k+1” parts (**non-overlapping, non-empty**) , so, at least one part must have no error (exactly match).
  - Use an exact matching algorithm to find exact matches for each part, Look for a longer approximate match in the vicinity of the **exact match** up to “k” mismatches.



- New principle: Let p1, p2, ..., pj be a partitioning of P into j (j< k+1) **non-overlapping** non-empty substrings. If P occurs with up to k edits, then at least one of p1, p2, ..., pj must occur with ≤ floor(k / j) edits.

Exact Matching

- Greedy Algorithms: Most of the efficient string matching algorithms in the DNA alphabet are modifications of the **Boyer–Moore** algorithm [1].
  - Bad character heuristic, good suffix rule
  - The pattern is moved forward (shift) after the first character mismatch of an alignment is observed [2,3].
  - Shift can be based on a single character or q-grams (strings of q characters) [4] uses two characters for indexing a two dimensional array.
  - An extension of Boyer-Moore algorithm [5]: shift array is indexed with an integer formed from a q-gram with **shift** and **add** instructions.
  - Kim and Shawe-Taylor[6]: Using alphabet compression by masking the three lowest bits of ASCII characters.
  - KMP, bitap, Robin-Karp, and many more.

[1] Kalsi, Petri, Hannu Peltola, and Jorma Tarhio. "Comparison of exact string matching algorithms for biological sequences." *Bioinformatics Research and Development*. Springer, Berlin, Heidelberg, 2008. 417-426.

[2] Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6(1), 323–350 (1977)

[3] Boyer, R.S., Moore, J S.: A fast string searching algorithm. Communications of the ACM 20(10), 762–772 (1977)

[4] Zhu, R.F., Takaoka, T.: On improving the average case of the Boyer–Moore string matching algorithm. Journal of Information Processing 10(3), 173–177 (1987)

[5] Baeza-Yates, R.: Improved string searching. Software: Practice and Experi- ence 19(3), 257–271 (1989)

[6] Kim, J.W., Kim, E., Park, K.: Fast matching method for DNA sequences. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 271–281. Springer, Heidelberg (2007)

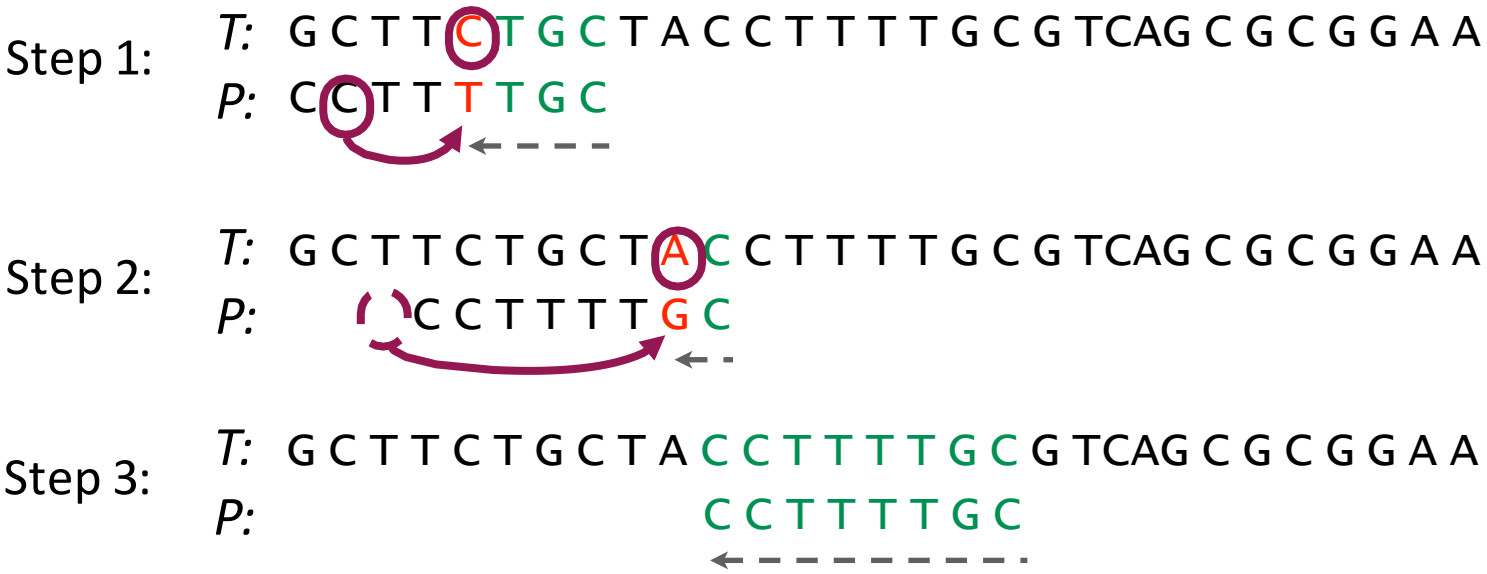
Boyer-Moore

- Better version of naive exact matching by skipping pointless alignments:
  - Learn from character comparisons to skip pointless alignments
- Naive exact matching: Two nested loops, outer loop is used to go over each alignment, and inner loop for looping over characters:
  - loop over alignments
  - loop over characters
  - compare characters
- Boyer-Moore: Alignments in left-to-right order, and try character comparisons in right-to-left order
- Two rules to skip pointless alignments:
  - Bad character rule
  - Good prefix rule

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.

Boyer-Moore: Bad character rule

- Upon mismatch, skip alignments until
  - Mismatch becomes a match, or
  - *P* moves past mismatched character



Boyer-Moore: Good suffix rule

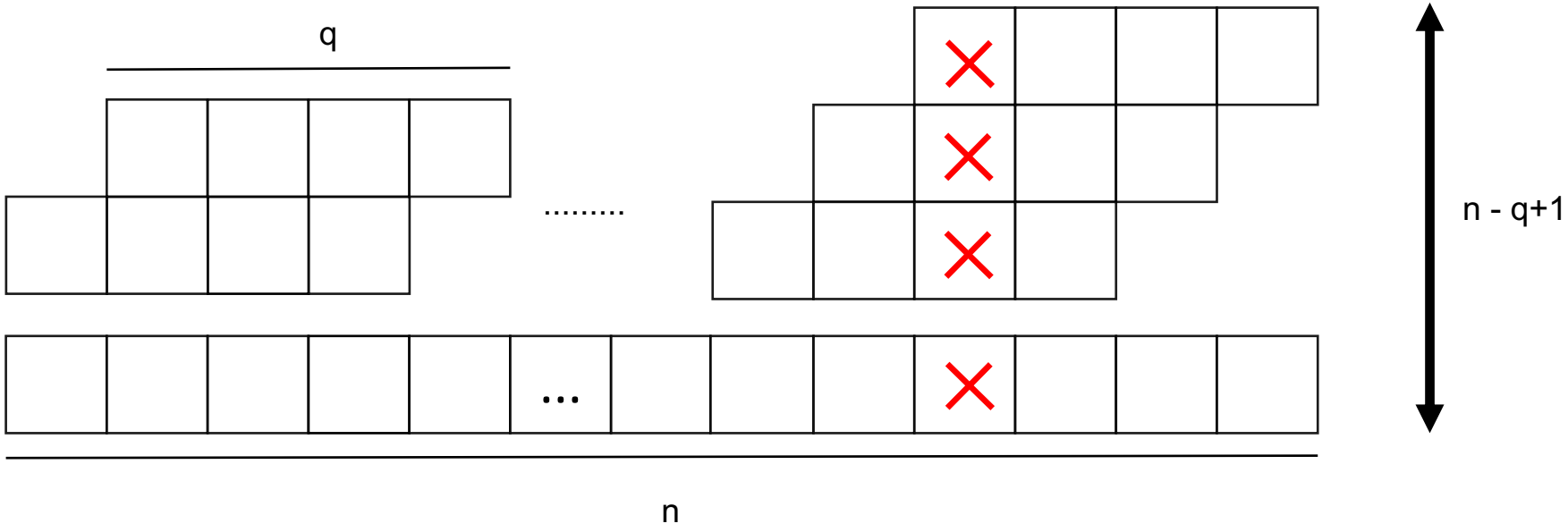
- Let say *t* = substring matched by inner loop; skip until
  - (a) there are no mismatches between *P* and *t or*
  - (b) *P* moves past *t*

Overlapping partitions(q-grams)

Georgia Institute of Technology

What if we have overlapping partitions?

- We have n - q +1 of them
- Worst case: 1 edit to P changes up to q substrings
- Minimum # of length-q substrings unedited after k edits?
  - n - q + 1 - kq
- q-grams lemma: if P occurs in T with up to k edits, alignment must contain t exact matches of length q, where t ≥ n - q + 1 – kq



q-grams principles

- If P occurs in T with up to k edits, alignment contains an exact match of length q, where $q \geq \text{floor}(n / (k + 1))$

- Exact matching filter: find matches of length $\text{floor}(n / (k + 1))$ between T and any substring of P. Check vicinity for full match.

Outline

- Problem of Approximate Sequence Matching
- Type of Solutions:
  - Solution based on Exact Sequence Matching
    - Pigeonhole principle
  - Solutions based on dynamic programming
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Filters
  - Solutions based on Deterministic Automata
  - Solutions based on Bit Parallelism (parallelize another algorithm using bits)
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Indexing and/or Dynamic programing
- Summary

Dynamic Programming

$$X = X_1 \dots \dots \dots X_{i-1} X_i$$

$$Y = Y_1 \dots \dots \dots Y_{j-1} Y_j$$

$C_{i,j}$ = Min number of operations needed to turn X to Y

Note: We assume edit distance of the shorter strings have already been computed. Convert one to another (delete the last char of one and insert the other)

$$\begin{aligned} C_{i,0} &= i \\ C_{0,j} &= j \\ C_{i,j} &= \text{if } (x_i = y_j) \text{ then } C_{i-1,j-1} \\ &\quad \text{else } 1 + \min(C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1}) \end{aligned}$$

Can we do better? Indexing (reducing search space) + Better DP

Figure 7: Taxonomy of algorithms based on the dynamic programming matrix.

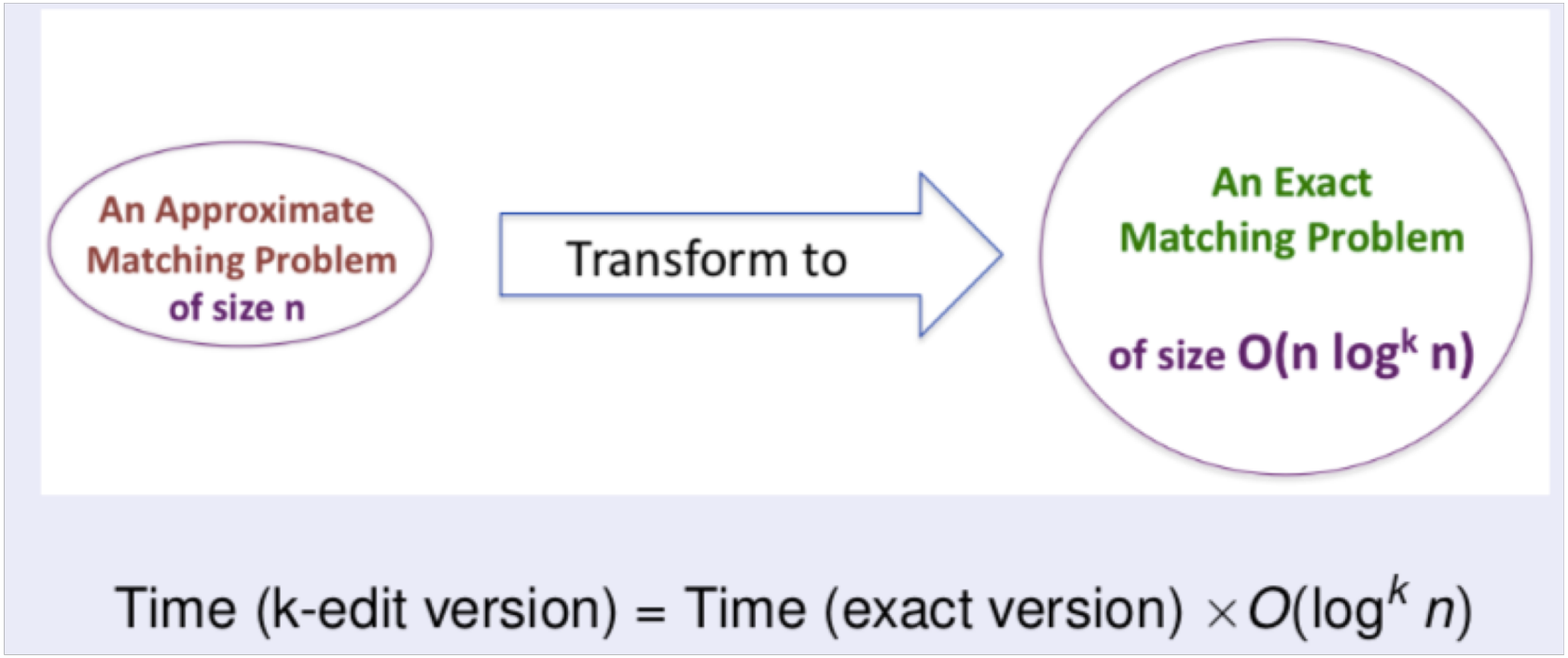
Navarro, Gonzalo. "A guided tour to approximate string matching." *ACM computing surveys (CSUR)* 33.1 (2001): 31-88.

Outline

- Problem of Approximate Sequence Matching
- Type of Solutions:
  - Solution based on Exact Sequence Matching
    - Pigeonhole principle
  - Solutions based on dynamic programming
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Filters
  - Solutions based on Deterministic Automata
  - Solutions based on Bit Parallelism (parallelize another algorithm using bits)
  - Solutions based on String Data Structures (SA, lcp), LCA, SA intervals,...
  - Solutions based on Indexing and/or Dynamic programing
- Summary

Based on ST, SA, lcp...

- Matching under bounded “k” edits:[1]



[1]Thankachan, Sharma V., et al. "Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2018.

Bounded number of errors

Georgia Institute of Technology

An Approximate Matching Problem

Given $S[1...n]$, compute the array L, where

$$L[i] = \max_{j \neq i} \text{lcp}_k(S_i, S_j)$$

Note: Longest k-edit repeat is of length $= \max_i L[i]$

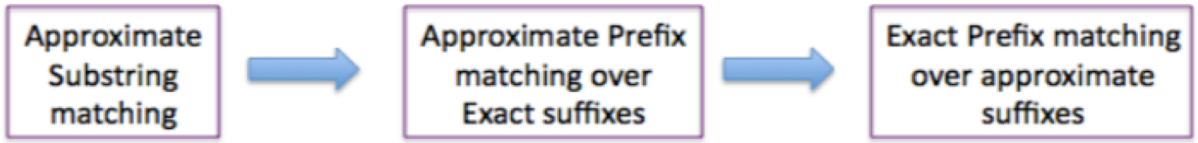
For any fixed (i, j), compute $\text{lcp}(S_i, S_j) = z$ in constant time (via suffix tree) and in $O(3^k)$ time, compute

$$\text{lcp}_k(S_i, S_j) = z + \max \begin{cases} \text{lcp}_{k-1}(S_{i+z+1}, S_{j+z+1}) + 1 & \text{(substitution)} \\ \text{lcp}_{k-1}(S_{i+z+1}, S_{j+z}) & \text{(deletion in } S_i\text{)} \\ \text{lcp}_{k-1}(S_{i+z}, S_{j+z+1}) & \text{(deletion in } S_j\text{)} \end{cases}$$

Total time: $O(n^2 3^k)$ is easy —-> $O(n^2 k)$ via DP —-> $O(n \log^k n)$ [new !]

Thankachan, Sharma V., et al. "Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2018.

Bounded number of errors



- A k-edited suffix S'_i is the string after "at most k" edits on the suffix S_i

.

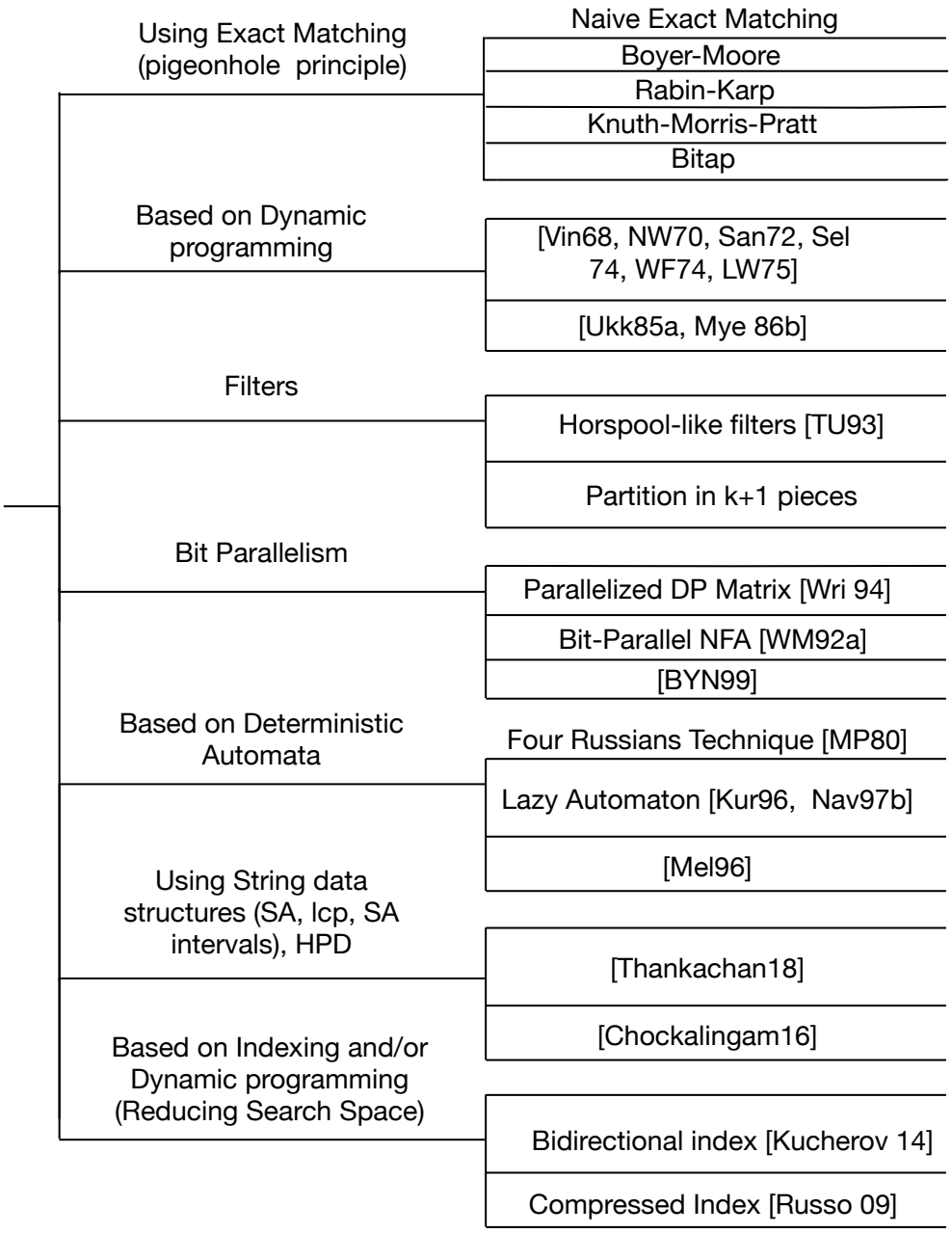
- We generate buckets (sets of k-edited suffixes) s.t.

1. total bucket size $= O(n \log^k n)$
2. max bucket size $= O(n)$
3. and

$$\text{lcp}_k(S_i, S_j) = \max_{S'_i, S'_j \in B} \text{lcp}(S'_i, S'_j)$$

Thankachan, Sharma V., et al. "Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2018.

Summary


Using Exact Matching (pigeonhole principle)
Naive Exact Matching
Boyer-Moore
Rabin-Karp
Knuth-Morris-Pratt
Bitap
Based on Dynamic programming
[Vin68, NW70, San72, Sel 74, WF74, LW75]
[Ukk85a, Mye 86b]
Filters
Horspool-like filters [TU93]
Partition in k+1 pieces
Bit Parallelism
Parallelized DP Matrix [Wri 94]
Bit-Parallel NFA [WM92a]
[BYN99]
Based on Deterministic Automata
Four Russians Technique [MP80]
Lazy Automaton [Kur96, Nav97b]
[Mel96]
Using String data structures (SA, lcp, SA intervals), HPD
[Thankachan18]
[Chockalingam16]
Based on Indexing and/or Dynamic programming (Reducing Search Space)
Bidirectional index [Kucherov 14]
Compressed Index [Russo 09]