Binary Jumbled Pattern Matching on Trees and Tree-Like Structures

T. Gagie, D. Hermelin, G.M. Landau, O. Weimann

Regular vs. Jumbled Pattern Matching

- (Regular) Pattern Matching:
  - Large text T (length n): MISSISIPICITSISIS
  - Small pattern P (length m): SISI
  - Report whether P occurs in T (or find all occurrences).
- Indexing variant:
  - Preprocess T to answer query patterns fast.

Regular vs. Jumbled Pattern Matching

- Jumbled Pattern Matching:
  - Large text T (length n): MISSISIPICITSISIS
  - Small pattern P (length m): SISI
  - Report whether some permutation of P occurs in T.
- Indexing variant:
  - Same as regular pattern matching.

Regular vs. Jumbled Pattern Matching

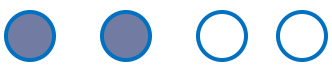
	Pattern Matching	Indexing
Regular	Several classical (non-trivial) O(n + m) solutions: • KnuthMorrisPratt • BoyerMoore • KarpRabin • …	O(n) construction time and space, Õ(m) query time: • Suffix Trees • Suffix arrays • …
Jumbled	Trivial schoolbook "sliding window" solution gives O(n+m).	???

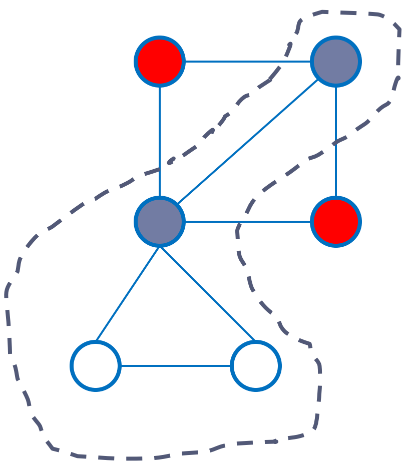
Indices for Binary Jumbled PM

- $O(n^2)$ construction time, $O(n)$ space, $O(1)$ query time [CicaleseFiciLipták '09].
- $O(n^2/\lg n)$ construction time, $O(n)$ space, $O(1)$ query time [BursciCicaleseFiciLipták '10] and [MoosaRahman '10].
- $O(n^2/\lg^2 n)$ construction time, $O(n)$ space, $O(1)$ query time [MoosaRahman '12].
- Better bounds only known for approximate indices ...

Jumbled Pattern Matching on Graphs

- Text: vertex-labeled graph on n vertices.
- Pattern: multiset of labels of size m.
- Question: Is there a connected subgraph whose label multiset matches the pattern ?

P = 

T = 

Jumbled Pattern Matching on Graphs

- Also known as the Graph Motif problem.
- Several work done on this (and variants):
  - [Lacroix, Fernandes, and Sagot ‘06]
  - [Fellows, Fertin, H., Vialette ‘07]
    - An $n^{O(cw)}$ algorithm for graphs with treewidth ≤ w and #labels ≤ c.
  - [Bruckner, Karp, Shamir, and Sharan ‘09]
  - [Dondi, Fertin, Vialette ’07+’09+’11]
  - [Guillemot and Sikora ‘10]
  - Several others . . .
- Trees ??

Our Results

- Index for trees: $O(n^2/\lg^2 n)$ construction time, $O(n)$ bits, $O(1)$ query time.
  - Matches the performance of the best known index for strings.

- Index for grammars: $O(g^{2/3}n^{4/3} /\lg^{4/3} n)$ construction time, $O(n)$ bits, $O(1)$ query time.
  - Time-bound is $O(n^2/\lg^2 n)$ even when the string is incompressible.

- Bounded treewidth graphs: $f(w) \cdot n^{O(c)}$ algorithm.
  - Beats previous $n^{O(cw)}$ algorithm.

O(n)-space index for trees in $O(n^2)$ time

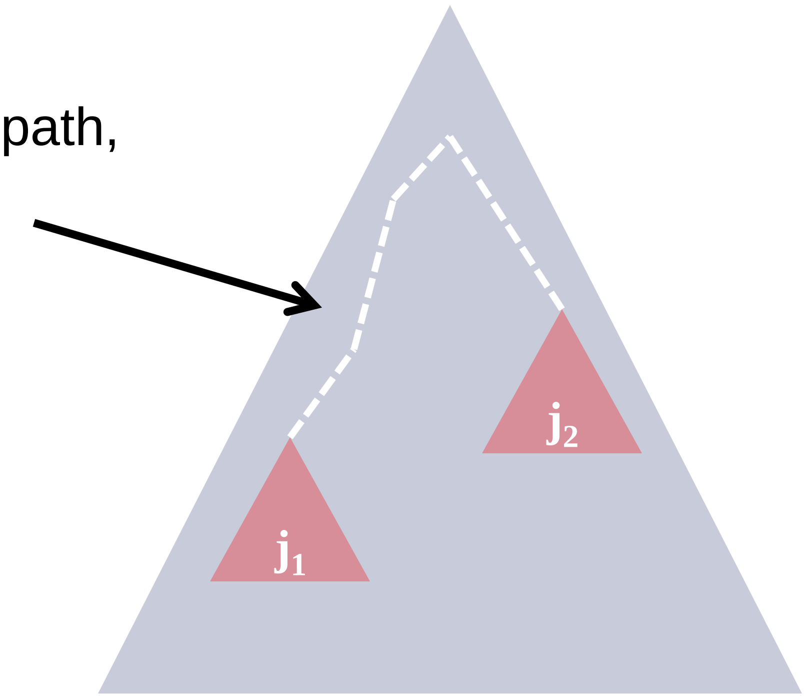
- Some conventions:
  - The tree T is rooted and complete binary (for ease of presentation).
  - Binary alphabet = tree nodes are colored either white or black.
  - Query (i,j) = A connected subgraph on exactly i nodes with exactly j black nodes.

Observation: *If (i,j_1) and (i,j_2) both occur in T, then (i,j) also occurs in T for all $j_1 \leq j \leq j_2$.*

O(n)-space index for trees in O(n^2) time

Observation: *If (i,j_1) and (i,j_2) both occur in T, then (i,j) also occurs in T for all $j_1 \leq j \leq j_2$.*

Each time we move on this path, the number of black nodes changes by at most 1.



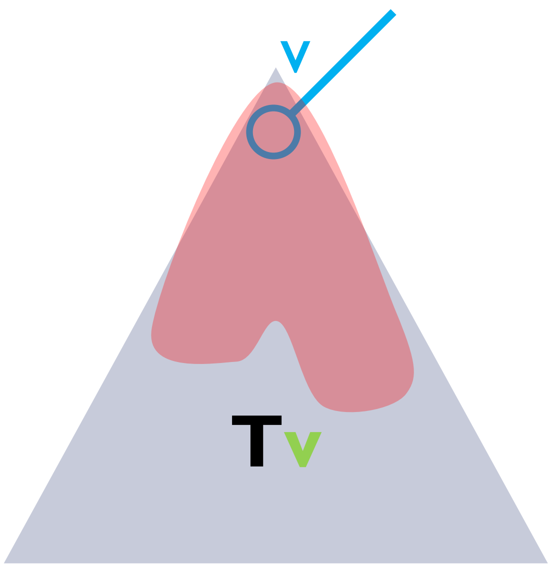
O(n)-space index for trees in $O(n^2)$ time

Observation: *If (i,j_1) and (i,j_2) both occur in T, then (i,j) also occurs in T for all* $j_1 \le j \le j_2$.

- For each i, store only two values:
  - i_{min}= minimum j such that (i,j) occurs in T.
  - i_{max}= maximum j such that (i,j) occurs in T.
- On query (i,j) report yes iff $i_{min} \le j \le i_{max}$.
- O(n) space.
- We show how all i_{max} values $O(n^2)$ time.
  - i_{min} analogous.

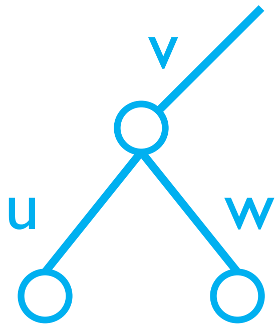
O(n)-space index for trees in O(n²) time

- For each node v and each i define
  - Av[i] = Maximum number of black nodes in a connected subgraph of size i in Tv which includes v.



O(n)-space index for trees in $O(n^2)$ time

- Simple top-down recursion:
  - $Av[i] = \max_k Au[k] + Aw[i-1-k] + col(v)$.



- Looks like $O(n^3)$ time, but its actually $O(n^2)$ time.

Succinct O(n)-bits index

Observation: *Either* $A_v[i] = A_v[i-1]$ *or* $A_v[i] = A_v[i-1]+1$.

- Thus, we can store the binary difference vectors instead:
  - $B_v[i] = A_v[i] - A_v[i-1]$.
- Since $A_v[i] = \sum_{1 \leq k \leq i} B_v[k]$, we can retrieve $A_v[i]$ from B_v in O(1) time using rank queries.
  - rank[i] = #1's in $B_v[1 \ldots i]$.

From trees to strings

Bu = **1 1 0 0 0 0**

#1 = Au[4]

Bw = **0 0 1 1 1 0**

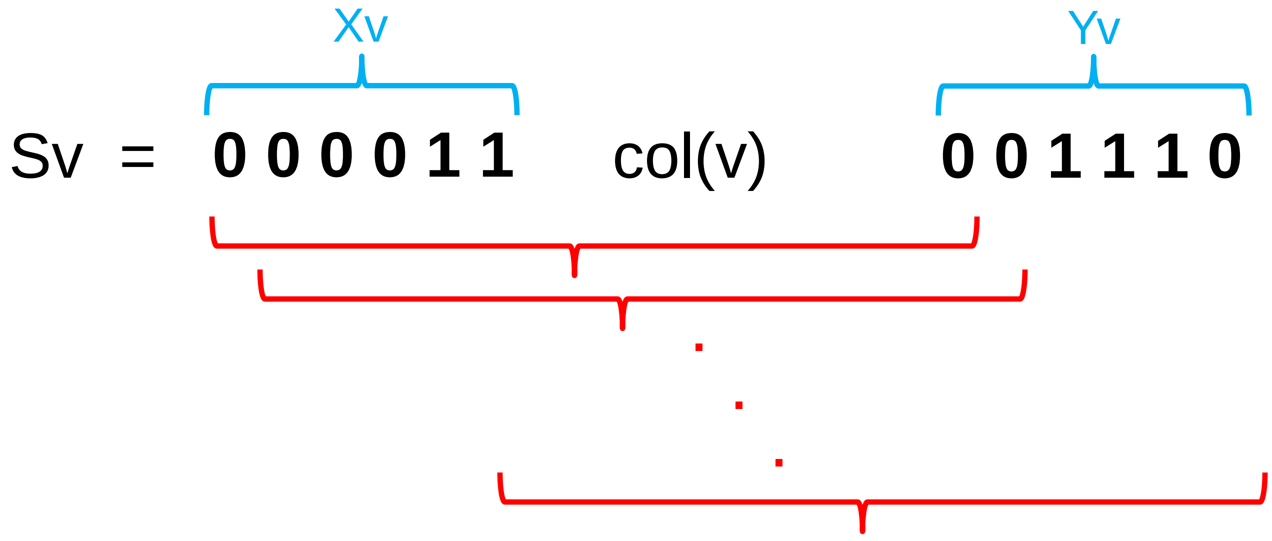
#1 = Aw[3]

Sv = **0 0 0 0 1 1** col(v) **0 0 1 1 1 0**

#1 = Au[4] + Aw[3] + col(v)

From trees to strings

- Recall: $Av[i] = \max_k Au[k] + Aw[i-1-k] + col(v)$.
- Hence, $Av[i]$ = max #1's in a window of size i in Sv containing the col(v) position.



Shaving off log-factors

- Using the algorithm for strings, we can compute Av in $O(|Sv|^2/\lg^2 n)$ time.
- But in order to get $O(n^2/\lg^2 n)$ overall we need:
  - $O(|Xv| \cdot |Yv| / \lg^2 n)$, and not …
  - $O((|Xv| + |Yv|)^2 / \lg^2 n)$.
- To get $O(|Xv| \cdot |Yv| / \lg n)$ is relatively easy.
  - Use lookup tables of size $s \approx \lg n$.
  - Run sliding windows of sizes which are multiples of s.
  - In total, $O(n/\lg n)$ sliding windows.

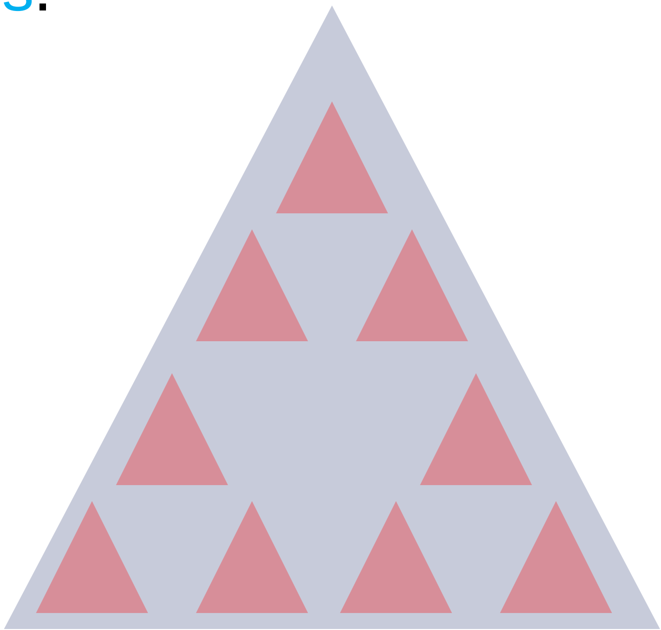
as is done in MoosaRahaman

Shaving off log-factors

- To get $O(|Xv| \cdot |Yv| / \lg^2 n)$ is problematic:
  - In strings, Moosa and Rahaman use additional lookup tables to slide the window in jumps of size $s \approx \lg n$.
  - Here we can also do this in most cases.
  - But what happens when, e.g., $|Xv| < s$?
    - Since we only consider sliding windows which include the col(v) position, we cannot jump !
    - In this case, we get a running-time of $O(|Yv| / \lg n) = O(n / \lg n)$.
  - **Solution:** Use micro-macro decomposition to ensure that the computations above happen only $O(n / \lg n)$ times.

Micro-macro decomposition

- Decompose the tree into a macro-tree of disjoint connected subgraphs, aka micro-trees.
  - Each micro-tree has size ≤ lg n.
  - # micro-trees = O(n/ lg n).



1. Compute (essentially) Av arrays for each micro-tree. Requires $O(\lg^2 n \cdot n/ \lg n) = O(n \lg n)$ time.

2. In bottom-up fashion, merge micro arrays to macro arrays using string speedups. Requires $O(n / \lg n \cdot n / \lg n) = O(n^2 / \lg^2 n)$ time.

Closing Remarks

- We can also find a node of an occurrence in O(lg n) time, assuming (i,j) occurs in T.
- Our algorithm can be made into a pattern matching algorithm running in $O(nm/\lg^2 n)$ time, when the pattern is of size m.
  - Can this be improved to near-linear (recall the string case) ?
- Bigger alphabets ?
- **A reasonable index for strings ?**

Thank you

for

your attention