Dictionary Matching With Uneven Gap

Yilin Yang

Joint work with Wing-Kai Hon, Tak-Wah Lam, Rahul Shah,
Sharma Thankachan, Hing-Fung Ting

Outline

- Problem Definition :
  Dictionary Matching With Uneven Gap
- Previous Work
- Main Idea
- Symmetric Problem :
  Dictionary Matching With Missing Substring

Outline

- Problem Definition :
  Dictionary Matching With Uneven Gap
- Previous Work
- Main Idea
- Symmetric Problem :
  Dictionary Matching With Missing Substring

Problem

- Preprocess : a dictionary with d gapped patterns
- Gapped pattern :

- Input : text T
- Output : all the occurrences

Example

- Pattern : ab [1,2] cd
- Text : abcdcdabccccd
- One match :

  abcdcdabccccd

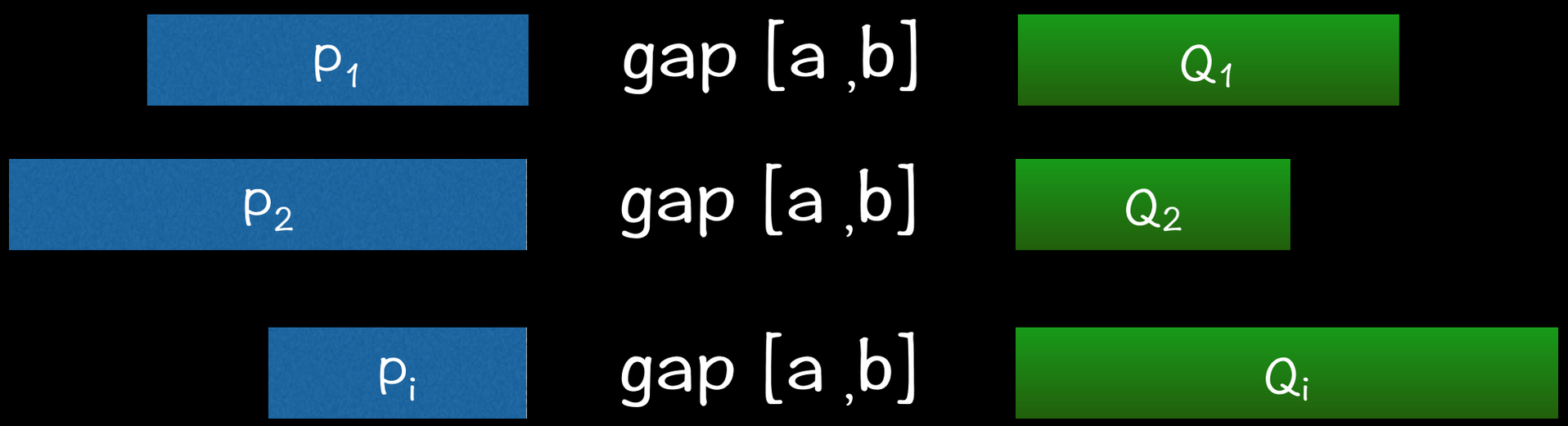
  ab??cd

Outline

- Problem Definition :
  Dictionary Matching With Uneven Gap
- Previous Work
- Main Idea
- Symmetric Problem :
  Dictionary Matching With Missing Substring

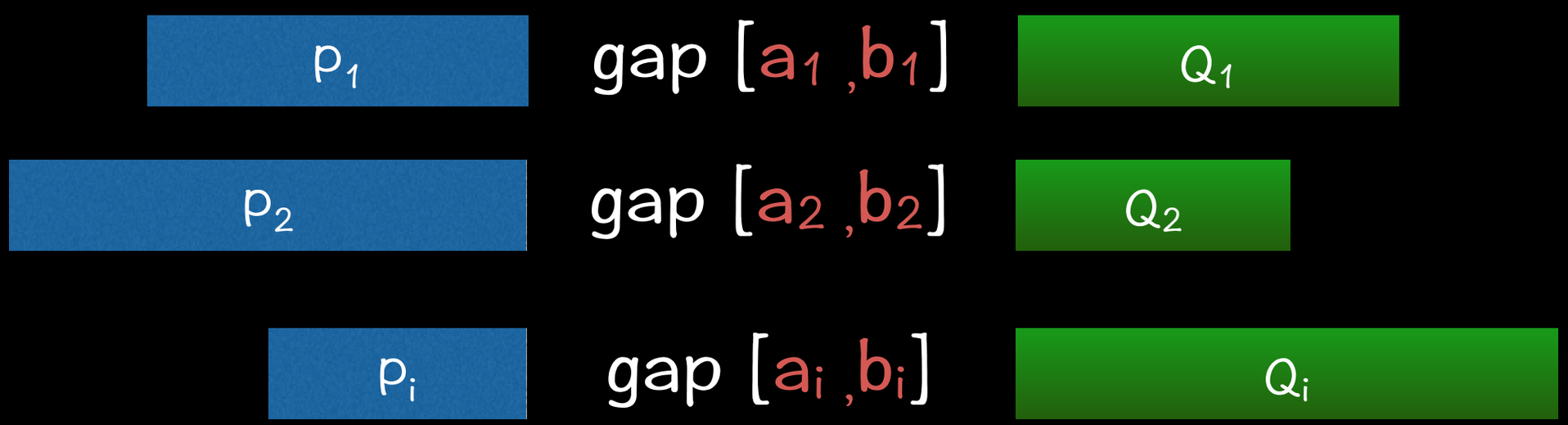
Previous Work

- Dictionary matching with one gap (Amir et al. CPM 2014)
- The same lower bound and upper bound of gap through all patterns



Our Problem

- Dictionary matching with uneven gap



Result

	Amir et al.	Amir et al.	Ours (uneven gap)	Ours (uneven gap)
Time	$O(\|T\| r \log^2 n \log\log d + occ)$	$O(\|T\| r + occ)$	$O(\|T\| r \log\lambda \log d + occ)$	$O(\|T\| r + occ)$
Space	$O(n + d\log^{\varepsilon} d)$	$O(n + d^2)$	$O(n)$	$O(n + d^{1+\varepsilon})$

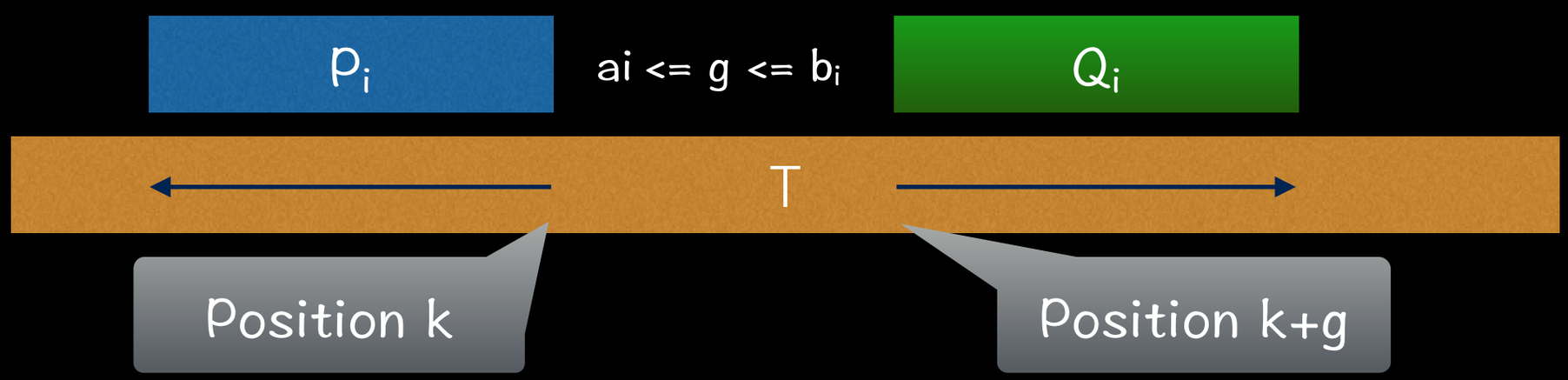
Note : $r = (B-A+1)$, $A = \min(a_i)$, $B = \max(b_i)$, $\lambda \leqq d$

Outline

- Problem Definition :
  Dictionary Matching With Uneven Gap
- Previous Work
- Main Idea
- Symmetric Problem :
  Dictionary Matching With Missing Substring

Search Framework

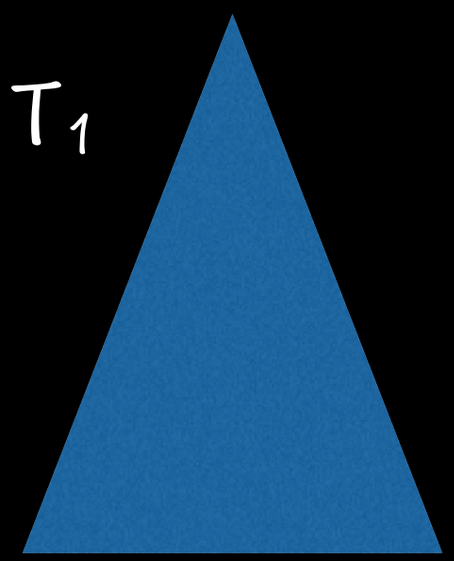
- If a gapped pattern i appears in the text T at position k:
  - We can find rev(P_i) as a prefix of T[k : 1]
  - We can find Q_i as a prefix of T[k+g : |T|]

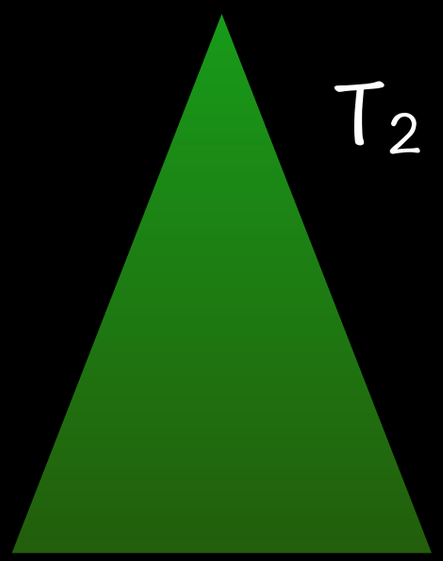


Search Framework

- So we construct two suffix trees :

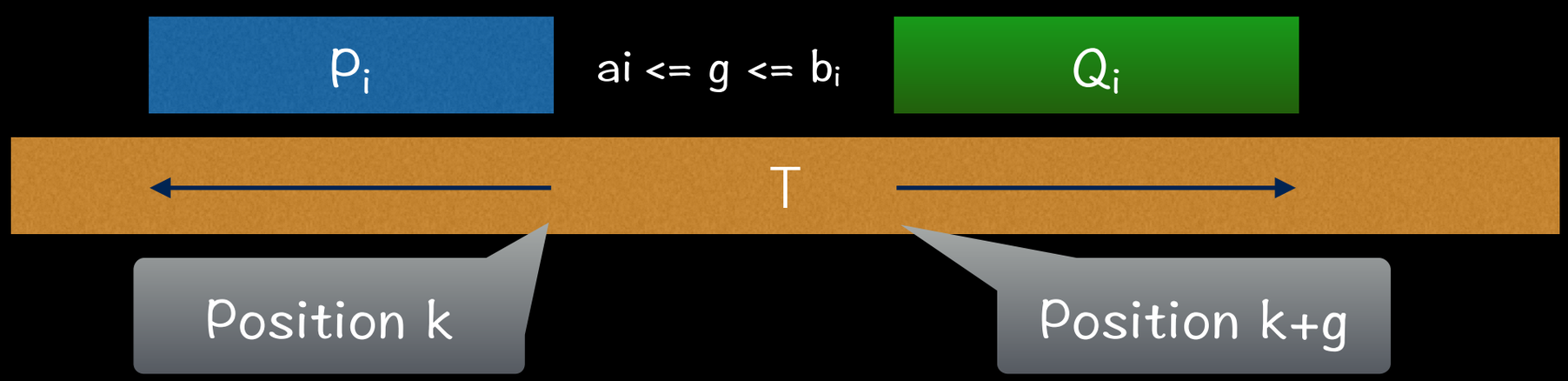
- T_1 : for $rev(P_i)$
- T_2 : for Q_i





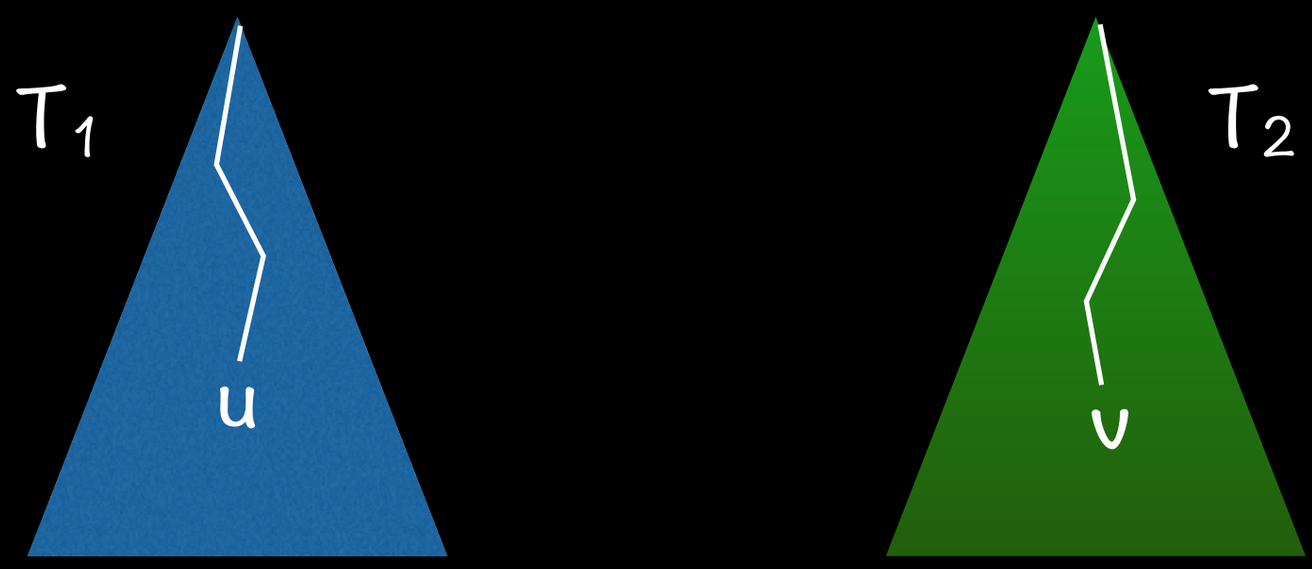
Search Framework

- When searching, we iterate position k and the gap size g



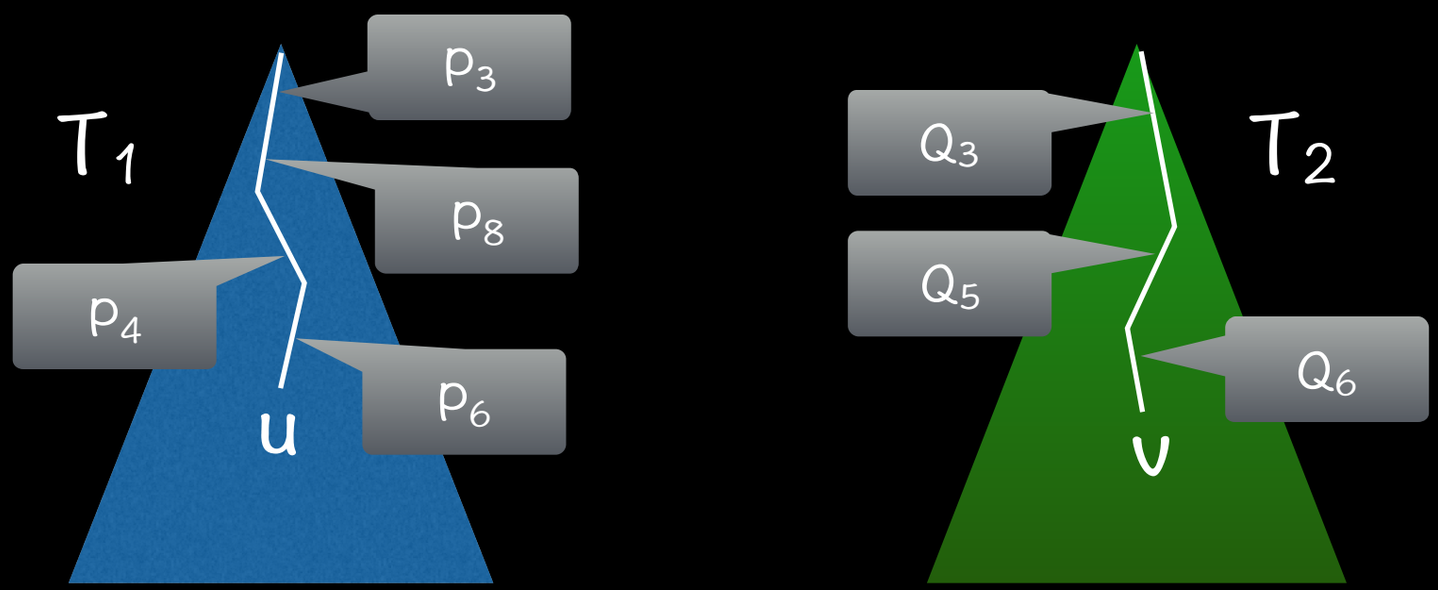
Search Framework

- When searching, we iterate position k and the gap size g
- Simulate "Insertion" of T[k : 1] into T_1 and T[k+g : |T|] into T_2 to get two locus nodes : u and v



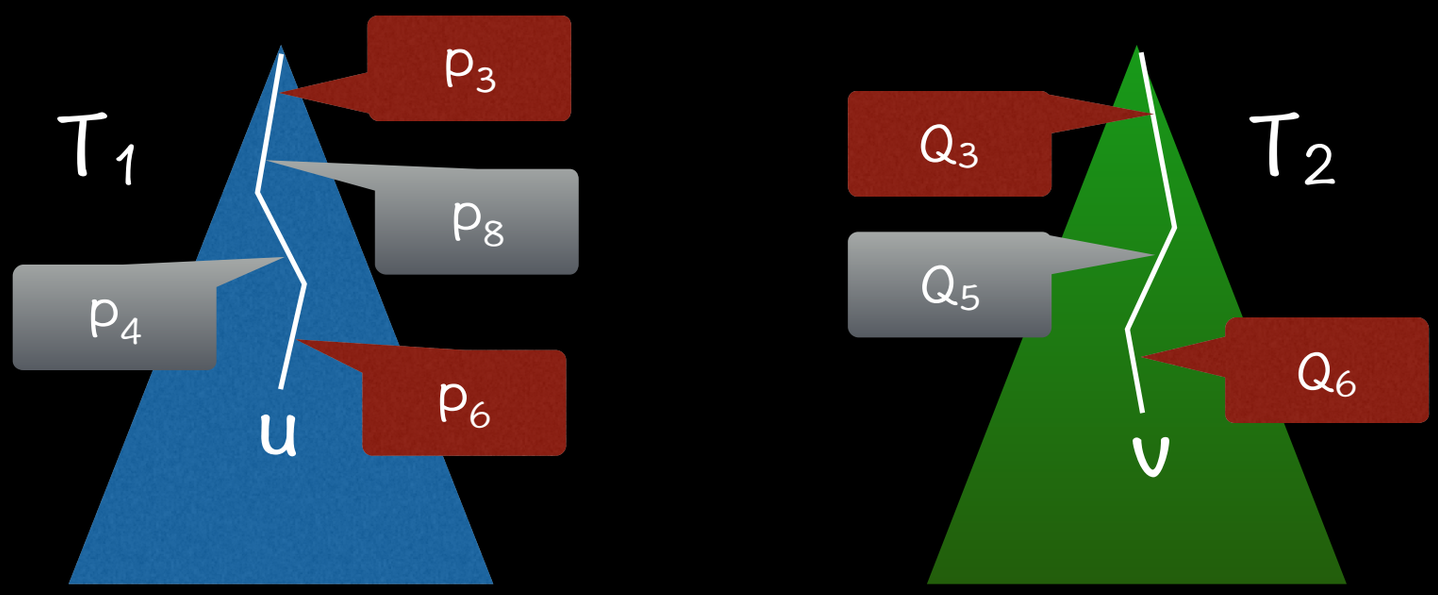
Search Framework

- We may find some $rev(P_i)$ appear as a prefix of $T[k : 1]$, and some Q_i appear as a prefix of $T[k+g : |T|]$



Search Framework

- Now, we need to output the intersection of them
- Ex : we find pattern 3 and pattern 6 at position k with gap size g



Problem translation

- With the framework above, the problem becomes to “Tree Path Intersection” problem :

- Given two trees and two paths on each tree

- Output the intersection

Previous solution

- Amir et al. uses heavy path decomposition and relabels the heavy path with contiguous number
  - Each pattern forms a 2D point
- Then the query path would be cut into at most log n heavy paths
  - Each query forms $\log^2 n$ rectangles
- Time complexity : $O(|T| (B-A+1) \log^2 n \log\log d + occ)$

Orthogonal range query

Our idea

	Amir et al.	Ours
Pattern	2D point	???
Text	$\log^2 n$ rectangles	???

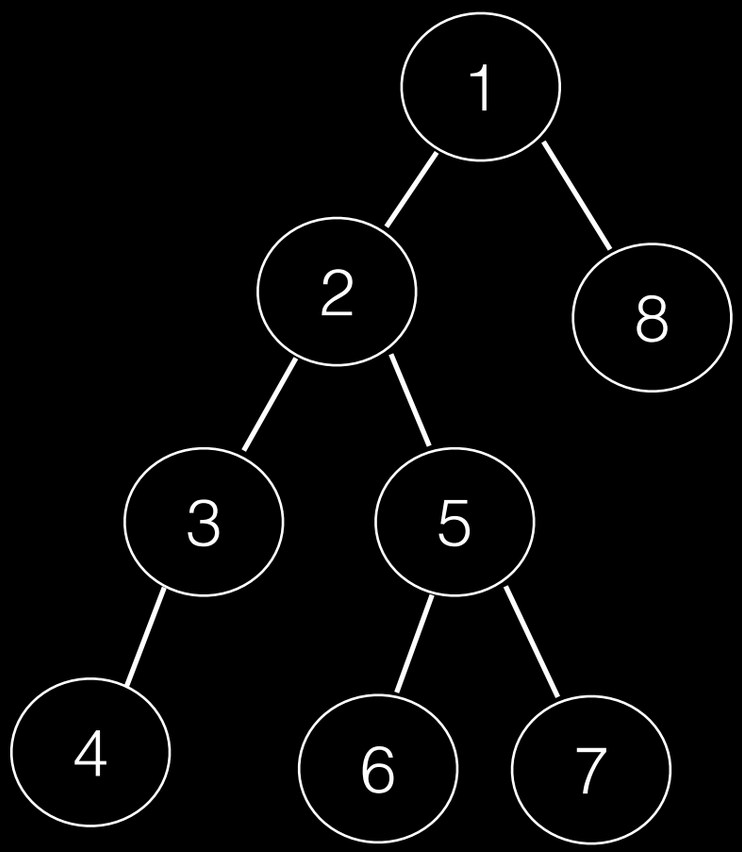
Our idea

- With different view …

	Amir et al.	Ours
Pattern	2D point	3D rectangle
Text	$\log^2 n$ rectangles	“only one” point !

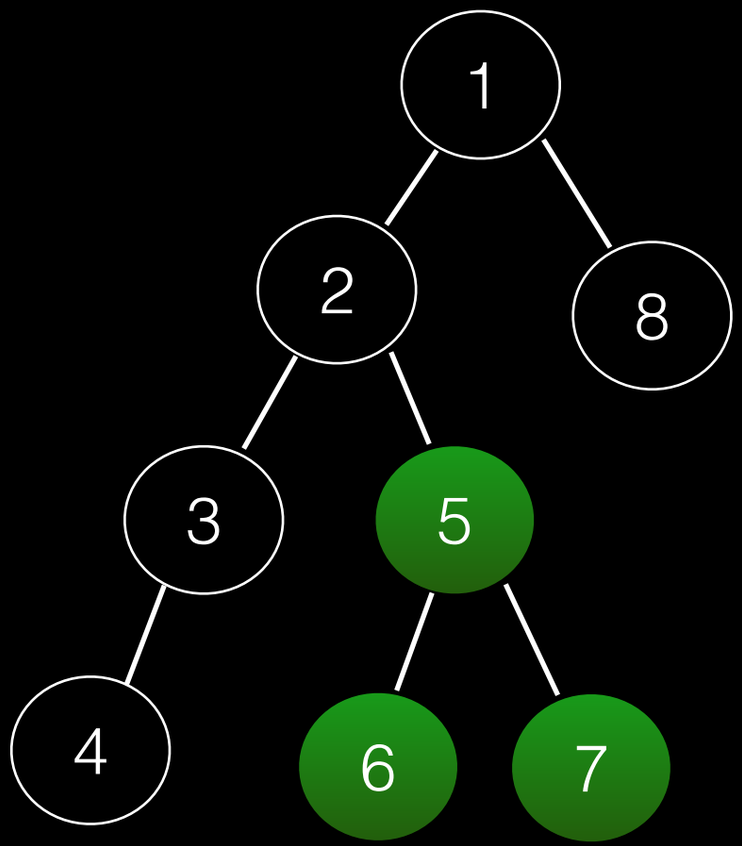
Index method

- The method is very simple : we relabel the tree using preorder traversal rank



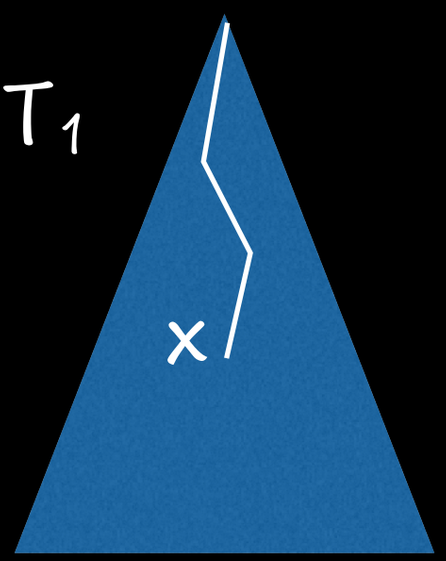
Index method

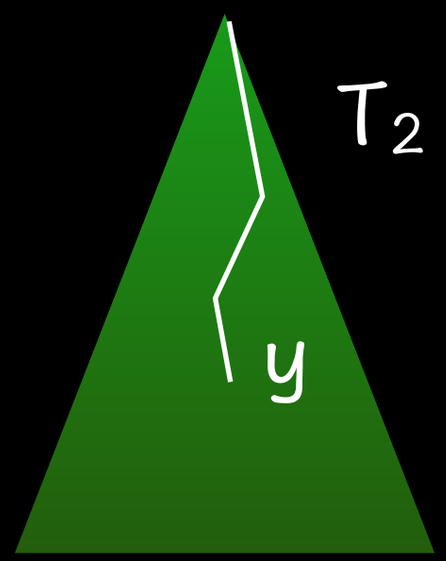
- Property : any subtree would contain contiguous relabeled number



Index method

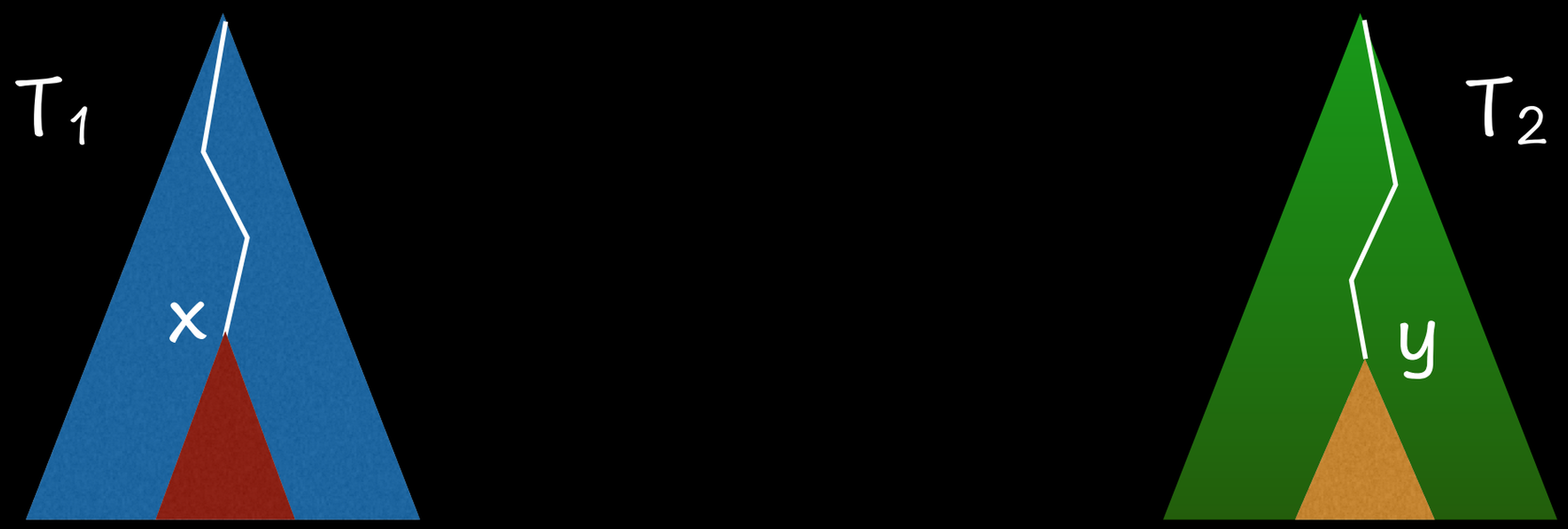
- For each gapped pattern i, we can find
  - locus node x of $rev(P_i)$
  - locus node y of Q_i





Index method

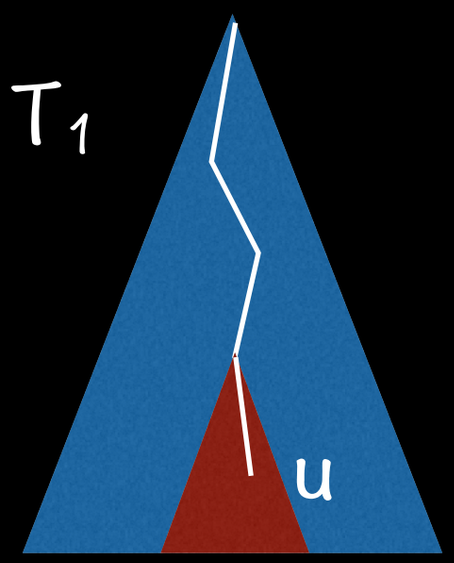
- And construct a 3D rectangle :
  - x coordinate : [pre(x), pre(x)+size(x)]
  - y coordinate : [pre(y), pre(y)+size(y)]
  - z coordinate : gap size $[a_i, b_i]$

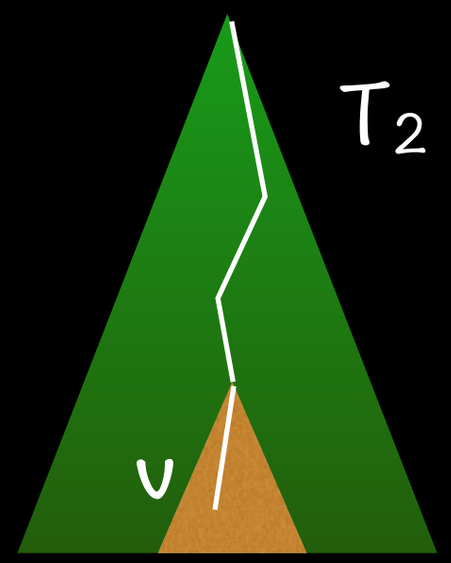


Query

- Then, each query can be changed into only one point : (pre(u), pre(v), g)

- rectangle stabbing query





Index method

- Space : O(n+dlogd)

- Time : O(|T| (B-A+1) $\log^2 d$ + occ)

3D rectangle stabbing query

Our idea

- With different view …

	Amir et al.	Ours
Pattern	2D point	rectangle
Text	$\log^2 n$ rectangles	“only one” point !

Result

	Amir et al.	Amir et al.	Ours (uneven gap)	Ours (uneven gap)
Time	$O(\|T\| r \log^2 n \log\log d + occ)$	$O(\|T\| r + occ)$	$O(\|T\| r \log\lambda \log d + occ)$	$O(\|T\| r + occ)$
Space	$O(n + d\log^{\varepsilon} d)$	$O(n + d^2)$	$O(n)$	$O(n + d^{1+\varepsilon})$

Note : $r = (B-A+1)$, $A = \min(a_i)$, $B = \max(b_i)$, $\lambda \leqq d$

Linear space

- Idea : we categorize the patterns into long and short patterns

- Then we can achieve the same time complexity, but linear space

More trade-offs

- Using more space : $O(n + d^{1+\varepsilon})$

- Faster query time : $O(|T| (B-A+1) + occ)$

More trade-offs

- Succinct space : $n \log\sigma + O(d \log n) + o(n \log\sigma)$ bits

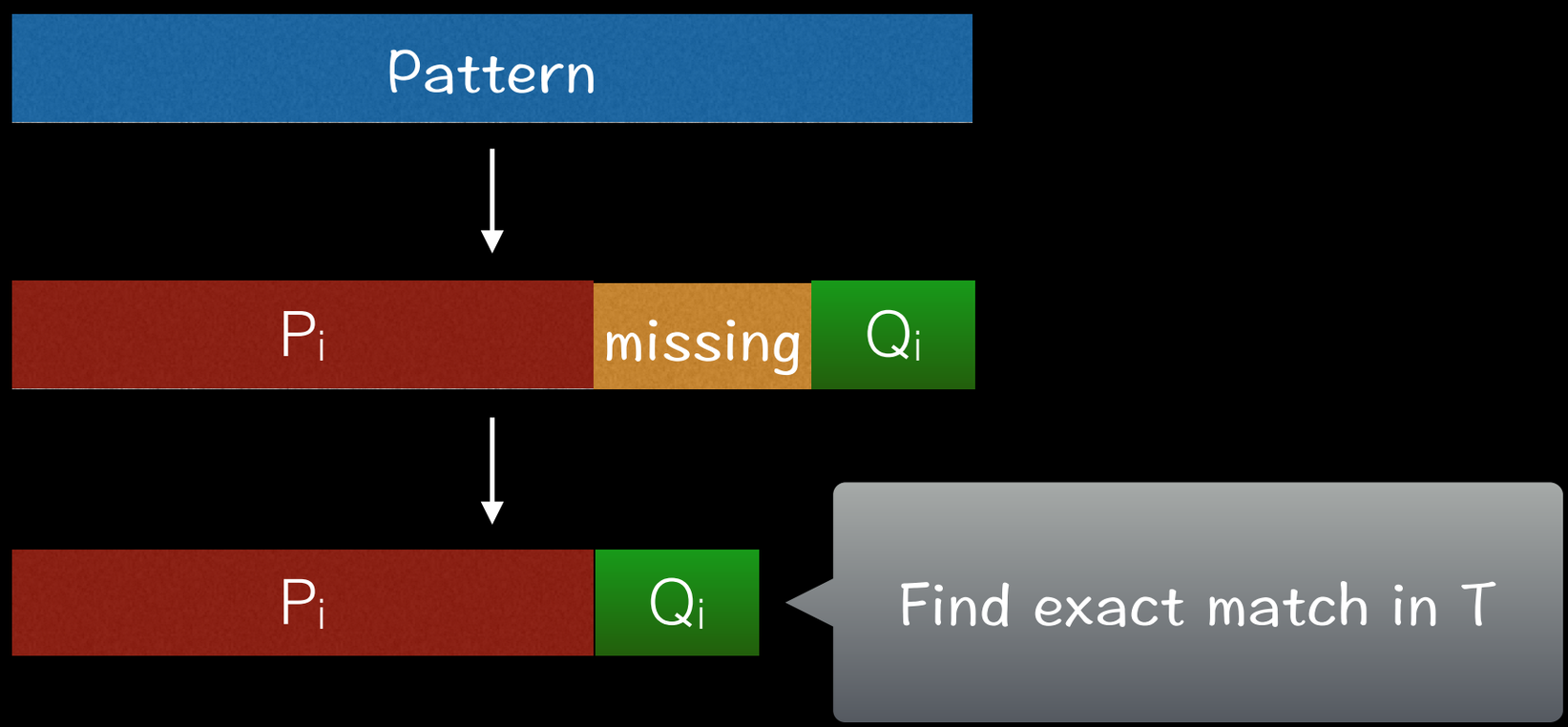
- Query time : $O(|T| (B-A+1) \log \lambda \log^{2+\varepsilon} n + occ)$

Outline

- Problem Definition :
  Dictionary Matching With Uneven Gap
- Previous Work
- Main Idea
- Symmetric Problem :
  Dictionary Matching With Missing Substring

Symmetric Problem

- Dictionary Matching With Missing Substring
- Pattern :



Symmetric Problem

- Pattern : abcd, b = 2
- Text T : acdabdad
- Three occurrences :

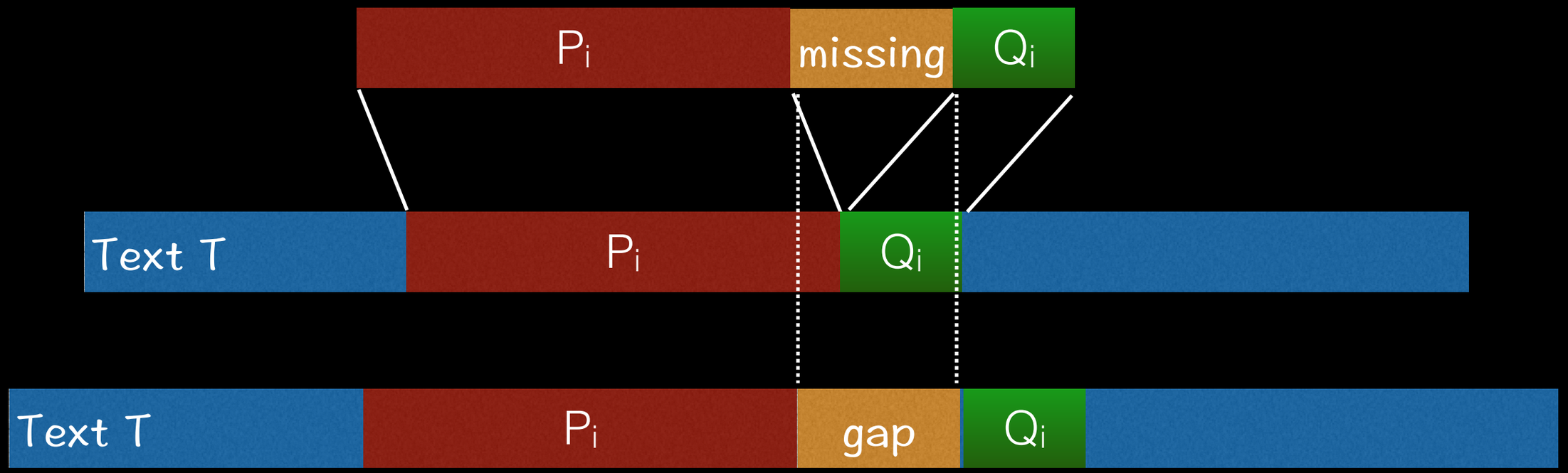
acdabdad
acd

acdabdad
abd

acdabdad
ad

Symmetric Problem

- It's like that we insert a gap into text T



- Dictionary matching with uneven gap :

Symmetric Problem

- For each pattern, if we know the starting position (only one) of missing substring :

- Space : O(n)

- Time : O(|T| logn + occ)

Symmetric Problem

- For each pattern, if we don't know the starting position of missing substring : (means it would happen anywhere in the pattern)
- Space : O(n logn)
- Time : O(k|T| logn + occ), k = max(|Pattern|)

Open Problems

- In dictionary matching with uneven gap problem, can we solve it with more than one gap ?

- In dictionary matching with missing substring problem, can we obtain succinct solution ?