Approximate String Matching by Position Restricted Alignment

Manish Patil

Xuanting Cai

Sharma V. Thankachan

Rahul Shah

Seung-Jong Park

David Foltz

Louisiana State University, USA

Problem Definition

- Given a non-negative integer τ, a string r and a collection of strings S, an approximate string query finds all pairs (r,s) with $s \in S$ such that $ed(r,s) \leq \tau$

Id	String	$ed(r,s)$
s_1	AAACTGTGC	1
s_2	AACTGTC	1
s_3	CTAATCT	6
s_4	GCGTC	4
s_5	GCGTCGT	5
s_6	TCAACCGTACG	4
s_7	TCCTATAAA	6
s_8	TCCAATAAA	7

$\tau = 2$

r = AACTGTGC

q-gram Based Approach

- If query *r* matches a string *s* with τ errors and if we split the pattern *r* in $\tau + t$ disjoint pieces arbitrarily, then at-least *t* pieces must be present in *s* as its substring with no errors

✔ s_2 = AA A CT GT GC

✗ s_8 = TCC AA TAAA

$q = 2$

$\tau = 2$

r = AA CT GT GC

q-gram Based Approach

- Maintain inverted lists for all q-grams in given collection of strings *S*

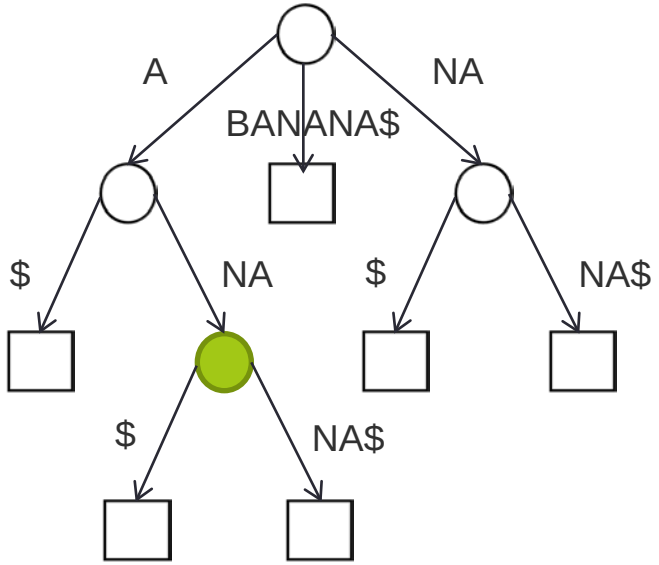
- Given a query string *r*
  - Partition the query string to obtain q-grams
  - Retrieve the inverted lists corresponding to each of the q-gram
  - Apply filtering techniques to prune strings and shrink inverted lists
  - Merge the lists to obtain candidate strings
  - Verify each of the candidate string for edit distance threshold$_T$

Observations

- Dilemma of choosing *q* (fixed at the time of index construction)
  - Maintain inverted lists of grams for all values of *q*
- Applying filters to prune out the candidate can be expensive in terms of computational cost
  - Integrate the inverted lists storage with filtering techniques enabling us to auto-filter the inverted lists

Suffix Tree

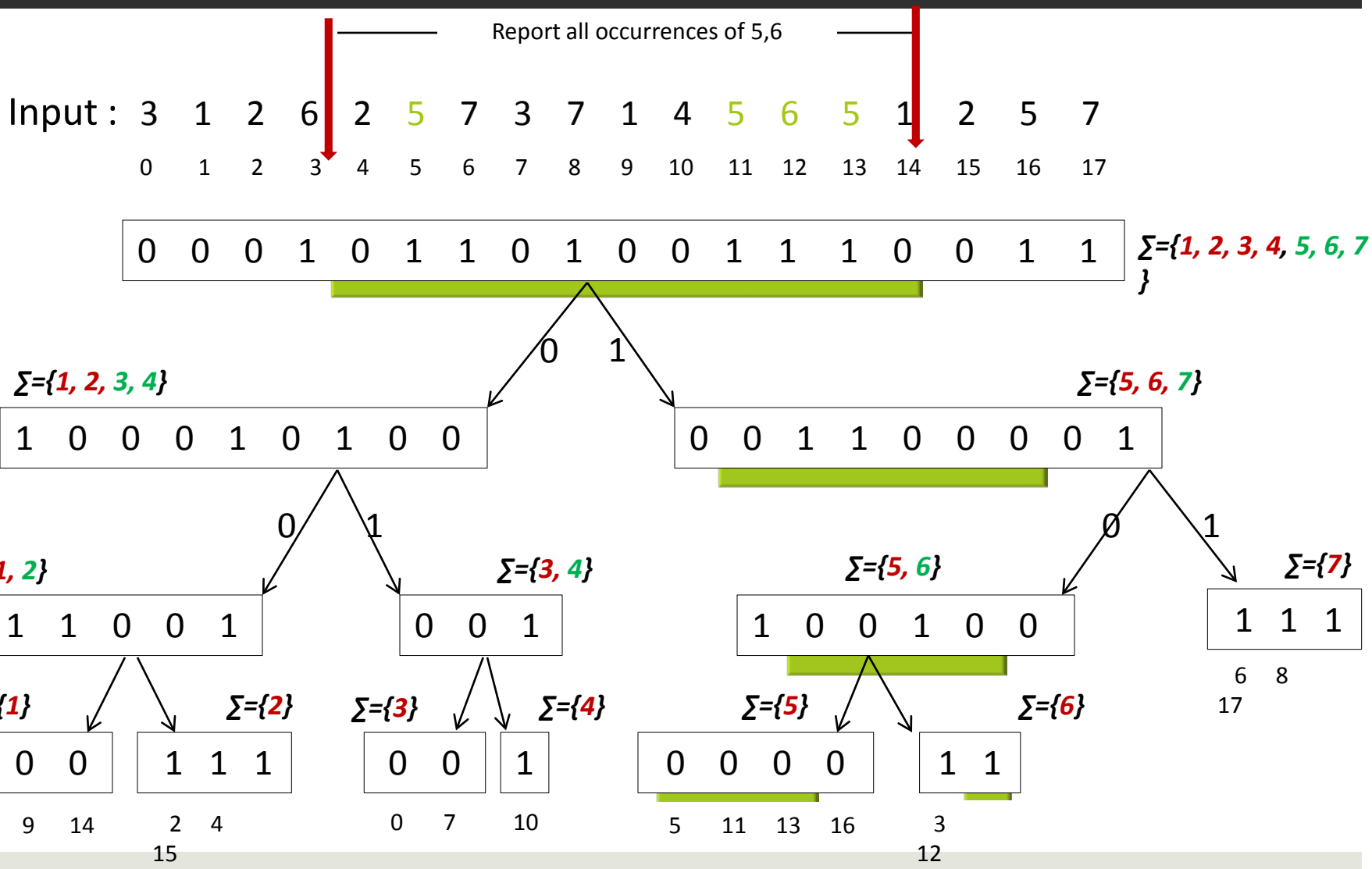
- Suffix tree is a compact trie storing the suffixes of the input string
  - *O(n)* words space
  - Locate inverted list corresponding to the given query *r*, in *O(|r|)* time



Suffix tree for string **BANANA**

- Generalized Suffix Tree (GST) is a compact trie which stores all suffixes of all strings in a given collection *S*

Wavelet-Tree



Wavelet-Tree

- WT is an ordered balanced binary tree where
  - each leaf is labeled with a symbol in Σ
  - leaves are sorted from left to right
  - each internal node represents an alphabet set Σ' , and is associated with a bit-vector (along with its rank-select structure)
  - each node partitions its alphabet set among the two children (almost) equally, such that all symbols represented by the left child are numerically smaller than those represented by the right child
- $O(n)$ words space
- Query time for 2D range searching is $O(\log \Sigma)$ per output

Basic Framework for Query Answering

- Maintain GST for given collection of strings *S*

- Given a query string *r*
  - Divide *r* into τ+ *1* segments each with length *floor(|r|/(τ+ 1))* except the last *|r| mod (τ+ 1)* segments which have length *ceil(|r|/(τ+1))*
  - Search for each partition in the GST to retrieve its inverted list
  - Apply Length and Position filtering to prune strings
  - Merge the lists using scan-count algorithm to obtain candidate strings
  - Verify each of the candidate string for edit distance thresholdτusing linear time algorithm

Length Filtering

- The length of a string *s* that is within edit distance τ from query string *r* is bounded by the equation: $||r| - |s|| \leq \tau$

Id	String	*Length*
s_1	AAACTGTGC	9
s_2	AACTGTC	7
s_3	CTAATCT	7
s_4	GCGTC	5
s_5	GCGTCGT	7
s_6	TCAACCGTACG	11
s_7	TCCTATAAA	9
s_8	TCCAATAAA	9

$\tau = 2$
r = AACTGTGC

$|r| = 8$
$6 <= |s| <= 10$

Position Filtering

- Let *s* contains a substring *s'* that matches substring *r'* of *r*
- Let *s'* have starting position s'_{sp} in *s* and substring *r'* has starting position r'_{sp} in *r*
- If alignment of *r* and *s* produced by matching *r'* and *s'* gives edit distance less than or equal to threshold T then $|r'_{sp} - s'_{sp}| \leq T$

Id	String
s_1	AAACTGTGC
s_2	AACTGTC
s_3	CTAATCT
s_5	GCGTCGT
s_7	TCCTATAAA
s_8	TCCAATAAA

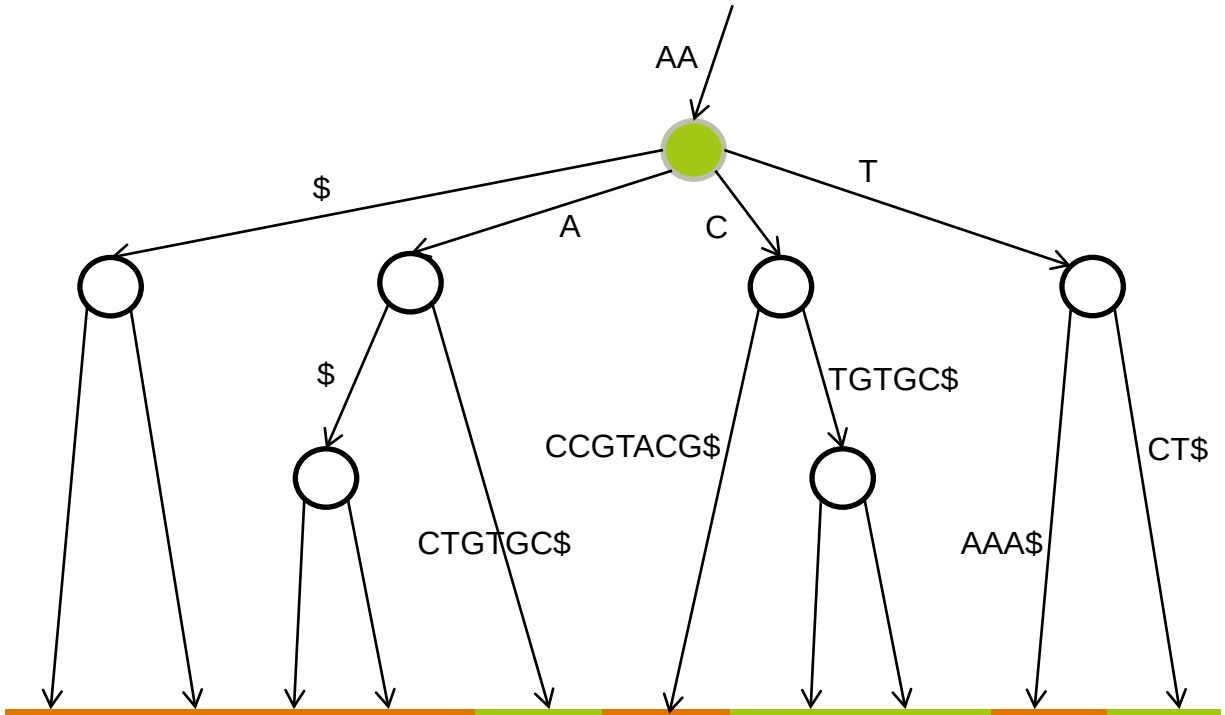
r = AA CTGTGC
s_7 = TCCTAT AA A

r = AA CTGTGC
s_3 = CT AA TCT

Basic Framework for Query Answering

T= 2

r = AA | CTG | TGC



Id	s_7	s_8	s_7	s_8	s_1	s_6	s_1	s_2	s_8	s_3
Length	9	9	9	9	9	11	9	7	9	7
Starting position	8	8	7	7	1	3	2	1	4	3

Position Restricted Alignment

- Consider alignment of *r* and *s* produced by matching *r'* and *s'*
  - *s'* is a substring of *s* and *r'* is substring of *r*
  - *s'* has starting position s'_{sp} in *s* and r'_{sp} be the same for *r'*
- Partition r in to r_{left}, *r'*, r_{right} around *r'* and similarly partition *s* as well

$$r = r_{left}\ r'\ r_{right}$$
$$s = s_{left}\ s'\ s_{right}$$

$$ed(r, s) \leq T$$

$$\Downarrow$$

$$ed(r_{left}, s_{left}) + ed(r_{right}, s_{right}) \leq T$$

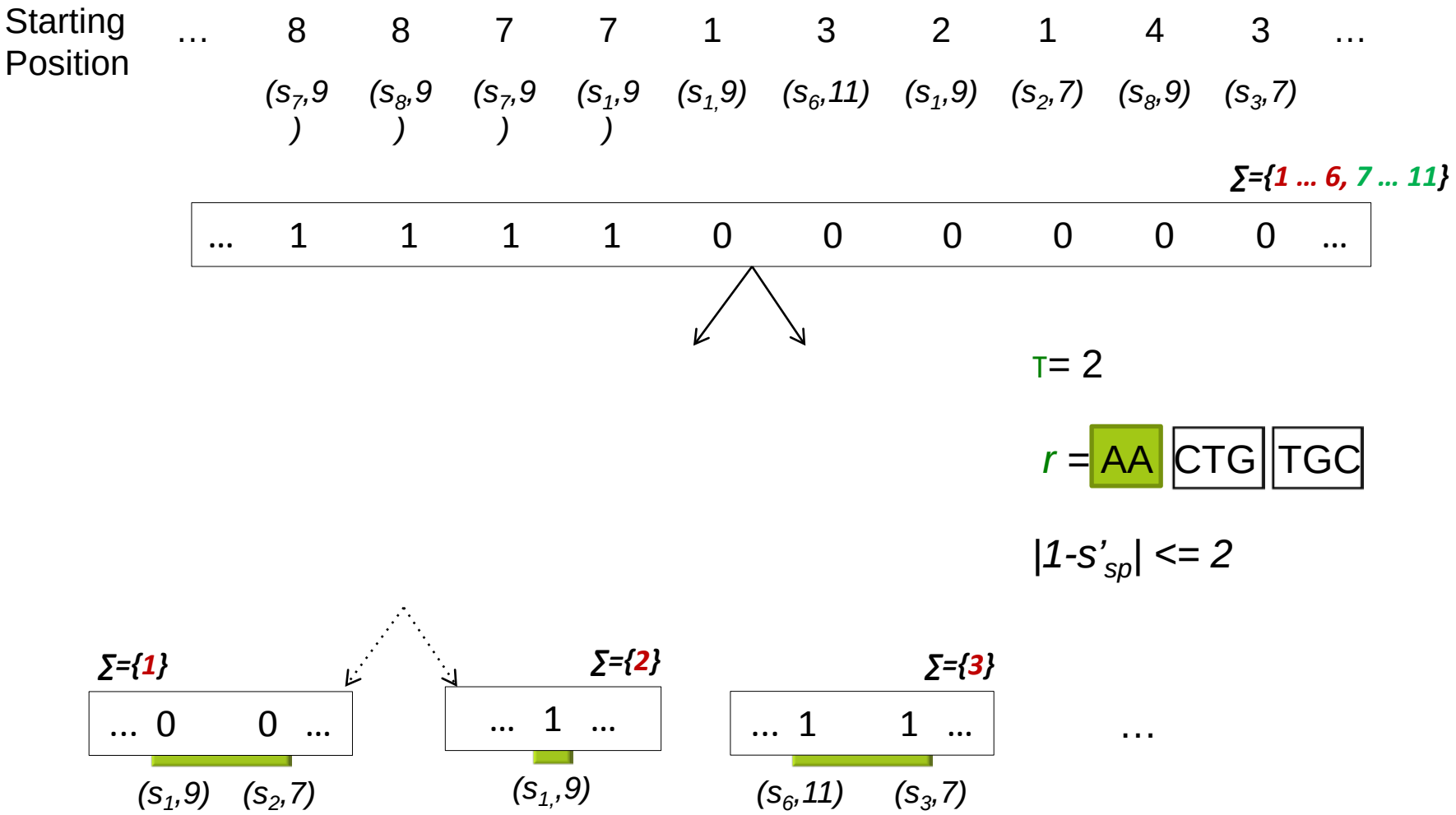
$$\Downarrow$$

$$|r'_{sp} - s'_{sp}| + |(|r| - r'_{sp}) - (|s| - s'_{sp})| \leq T$$

r = AA CTGTGC

✗ s_3 = CT AA TCT

Level 1: Wavelet-Tree



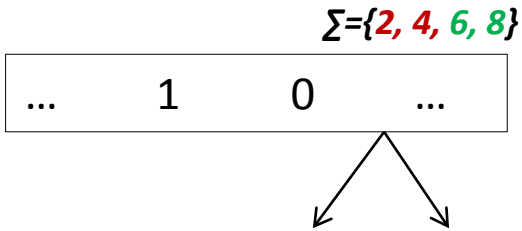
Level 2: Wavelet-Tree

(Length – Starting Position)

$\Sigma=\{3\}$

… 8 4 …

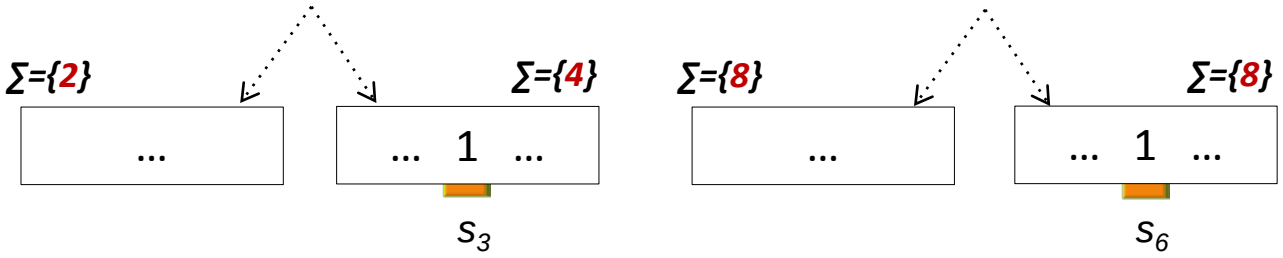
$(s_6,11)$ $(s_3,7)$



$|r'_{sp}-s'_{sp}| + |(|r|-r'_{sp})-(|s|-s'_{sp})| \leq \tau$

$2 + |7-(|s|-s'_{sp})| \leq 2$

$|s|-s'_{sp} <= 0$



Performance Bottleneck

- Particular string *s* can have multiple possible alignments with *r* based on partition *r'*
- Use of scan-count of merging inverted lists
- Results in artificial candidate strings increasing verification cost
- Special care has to be taken to increment count for a string *s* only once for each of the partition of *r*
  - easy to be taken care of theoretically
  - practically it is an additional cost to ensure uniqueness

Selectively Enforcing Uniqueness

- Goal: To balance cost of ensuring uniquess vs increased cost of verification due to artificial candidate strings

- Maintain a dictionary of GST nodes (strings) such that
  - *dist(u)/size(u)* < UQ_{min} ,where *dist(u)* is the number of distinct leaves (string ids) in the subtree of node *u* and *size(u)* is the total number of leaves in subtree of node u

- If *r'* is in the dictionary filter out duplicate string ids

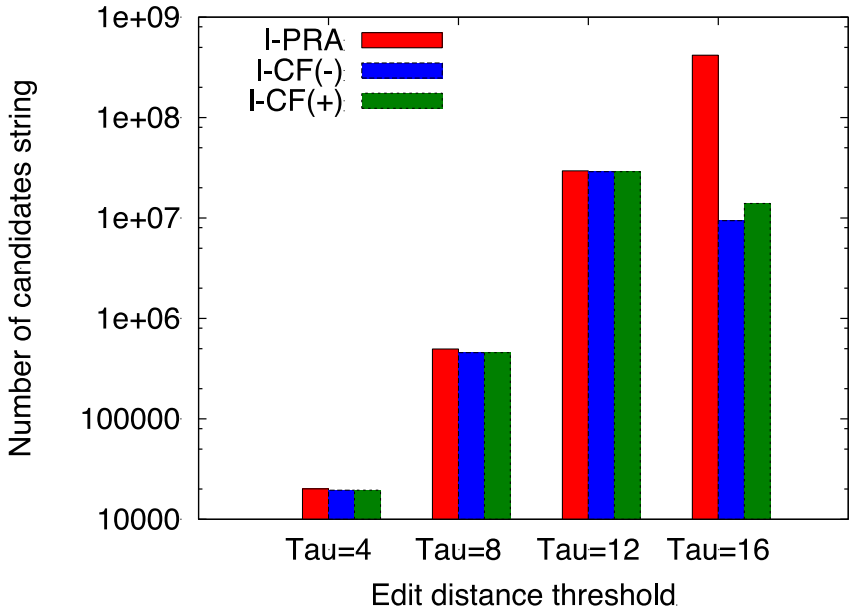
Other Practical Improvements

- Incorporating count filtering
  - Partition query *r* intoτ+ t disjoint pieces where t depends on query string length
- Variable length partitioning
  - Greedy partitioning of query string *r* guided by the dictionary of GST nodes
- Filtering based on frequency distance
  - If two strings are similar, then the frequency of the alphabet symbols in two strings should also be similar

Experimental Setup

- Public code libraries used
  - http://www.un-iulm.de/in/theo/research/sdsl.html
  - http://pizzachili.dcc.uchile.cl/indexes.html
- C++ Implementation (gcc 4.4 and above)
- Ubuntu machine with an Intel core i5 (quad core) 1.6GHz processor and 8GB RAM

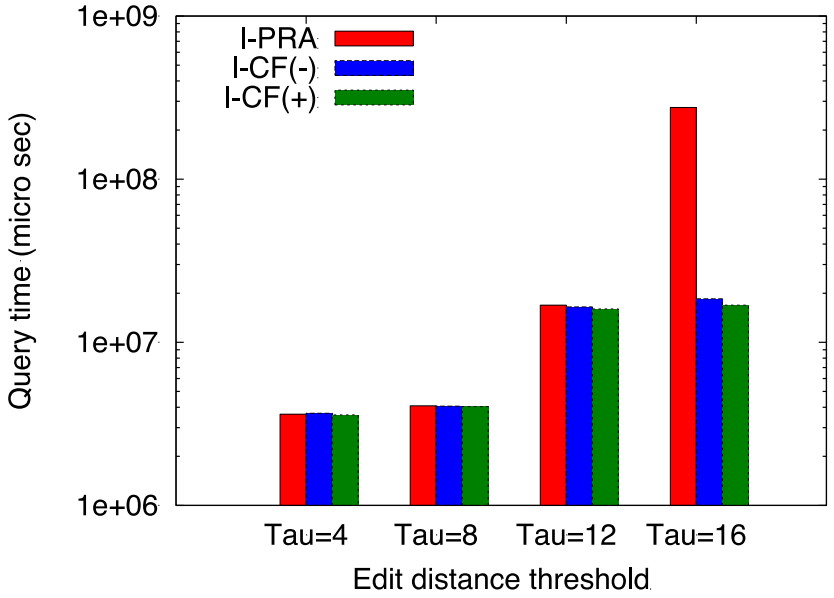
Results



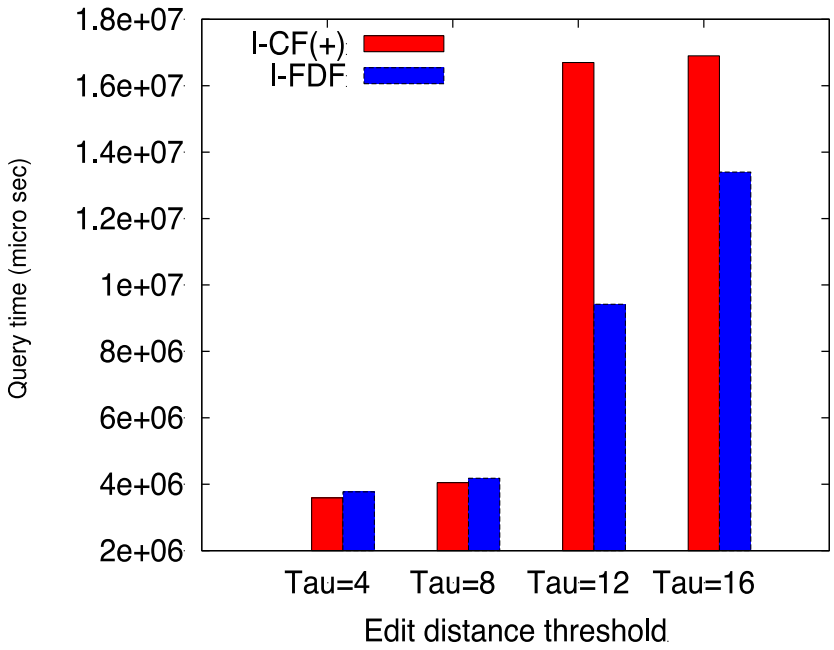
I-PRA: τ+ 1 partitioning
(No uniqueness check required)

I-CF(-): τ+ t partitioning with mandatory uniqueness check

I-CF(+): τ+ t partitioning with selective uniqueness check



Results



I-CF(+): τ+ t partitioning with selective uniqueness check

I-FDF: I-CF(+) with frequency distance filtering

Questions?

THANK YOU!