Data Mining: Foundation, Techniques and Applications

Lesson 11: Mining and Search Sequences

Li Cuiping(李翠平)
School of Information
Renmin University of China

Anthony Tung(鄧锦浩)
School of Computing
National University of Singapore

Outline

- Types of sequences
- Foundation
  - Full matching: Building a disk based suffix tree
  - Approximate matching Using vgrams
- Technique & Application
  - Finding global partial order in sequence
  - Finding motif in sequence

Types of sequences

- Symbolic vs Numeric
  - We only touch discrete symbols here. Sequences of number are called time series and is a huge topic by itself!
- Single dimension vs multi-dimensional
  - Example: Yueguo Chen, Shouxu Jiang, Beng Chin Ooi, Anthony K. H. Tung. "Querying Complex Spatial-Temporal Sequences in Human Motion Databases" accepted and to appear in 24th IEEE International Conference on Data Engineering (ICDE) 2008
- Single long sequence vs multiple sequences

Outline

- Types of sequences
- Foundation
  - Full matching: Building a disk based suffix tree
  - Approximate matching Using vgrams
- Technique & Application
  - Finding global partial order in sequence
  - Finding motif in sequence

Suffix

- Suffixes of acacag$:

1. acacag$
2. cacag$
3. acag$
4. cag$
5. ag$
6. g$
7. $

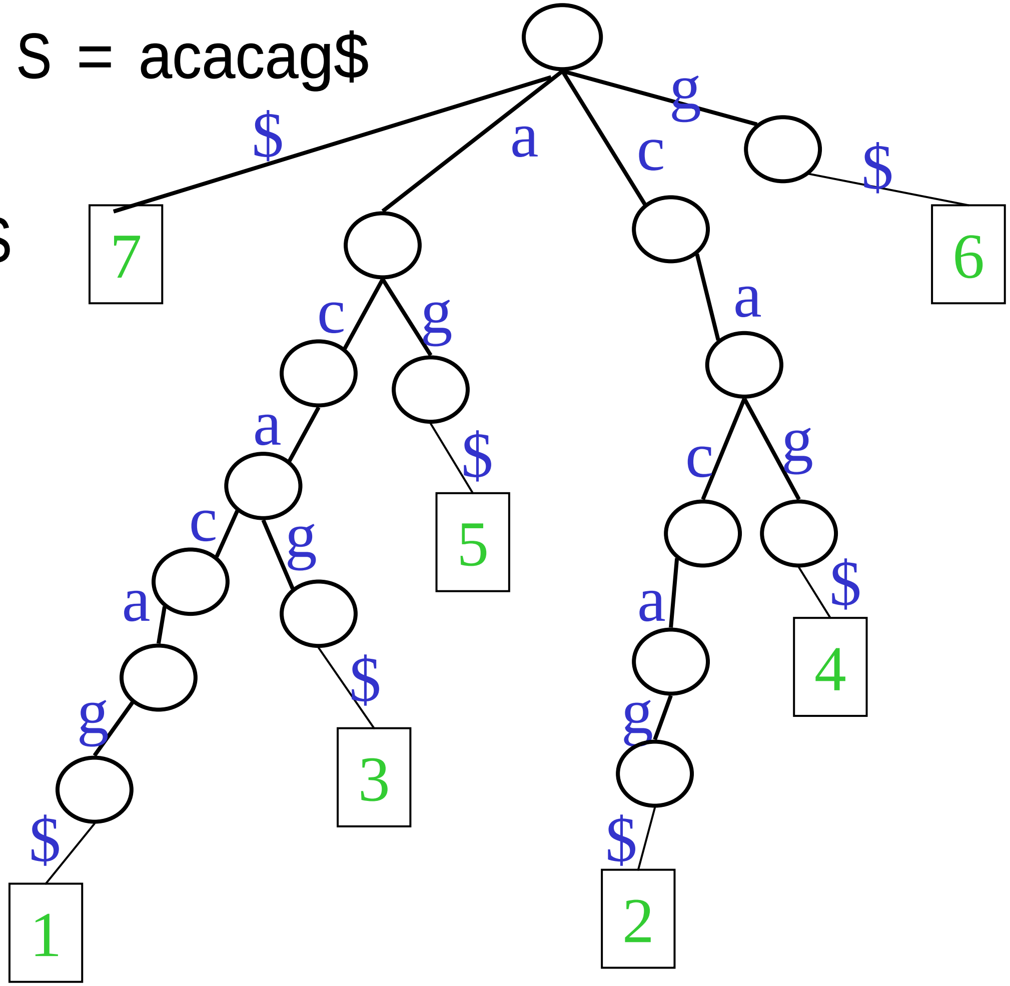
Suffix Trie

E.g. consider the string S = acacag$

Suffix Trie: a ties of

all possible suffices of S

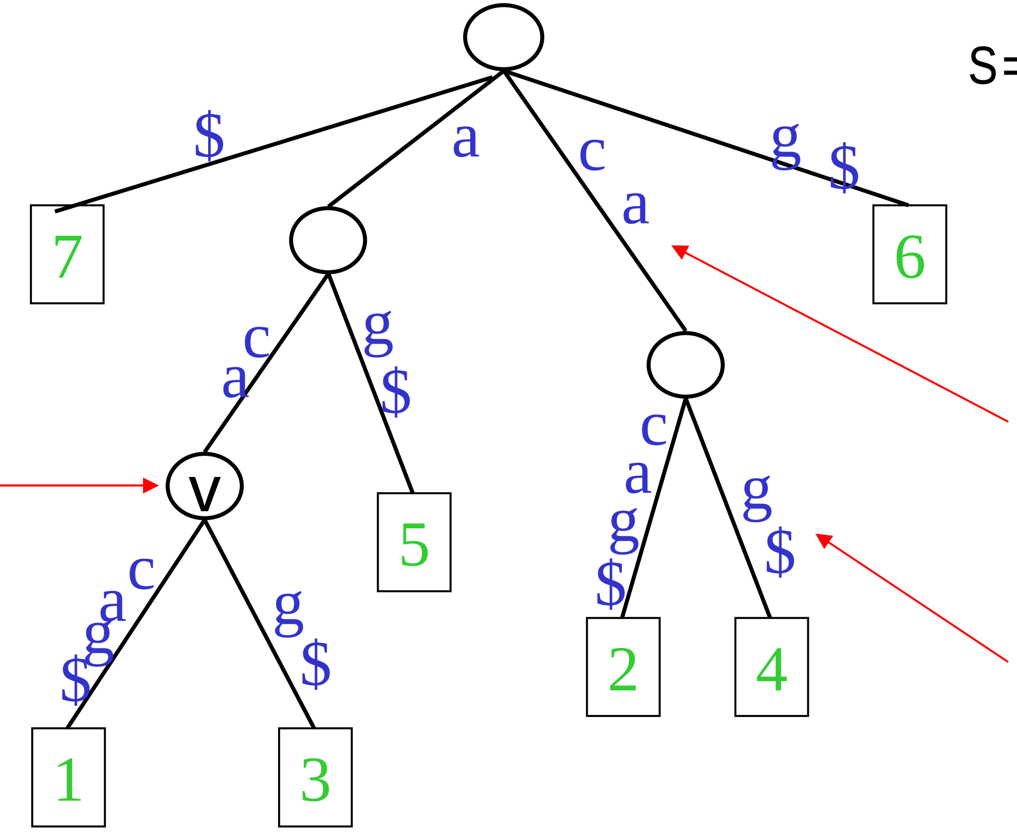
	Suffix
1	acacag$
2	cacag$
3	acag$
4	cag$
5	ag$
6	g$
7	$



Suffix Tree (I)

Suffix tree for S=acacag$: merge nodes with only one child

	1	2	3	4	5	6	7
S=	a	c	a	c	a	g	$



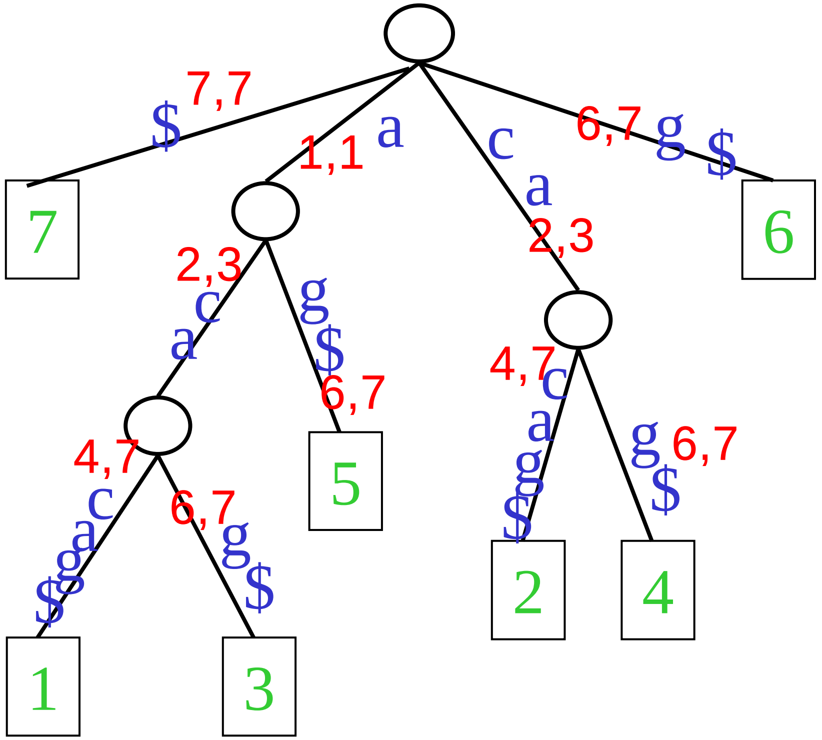
Path-label of node v is “aca” Denoted as α(v)

“ca” is an edge label

This is a leaf edge

Suffix Tree (II)

- Suffix tree has exactly n leaves and at most n edges
- The label of each edge can be represented using 2 indices
- Thus, suffix tree can be represented using O(n log n) bits

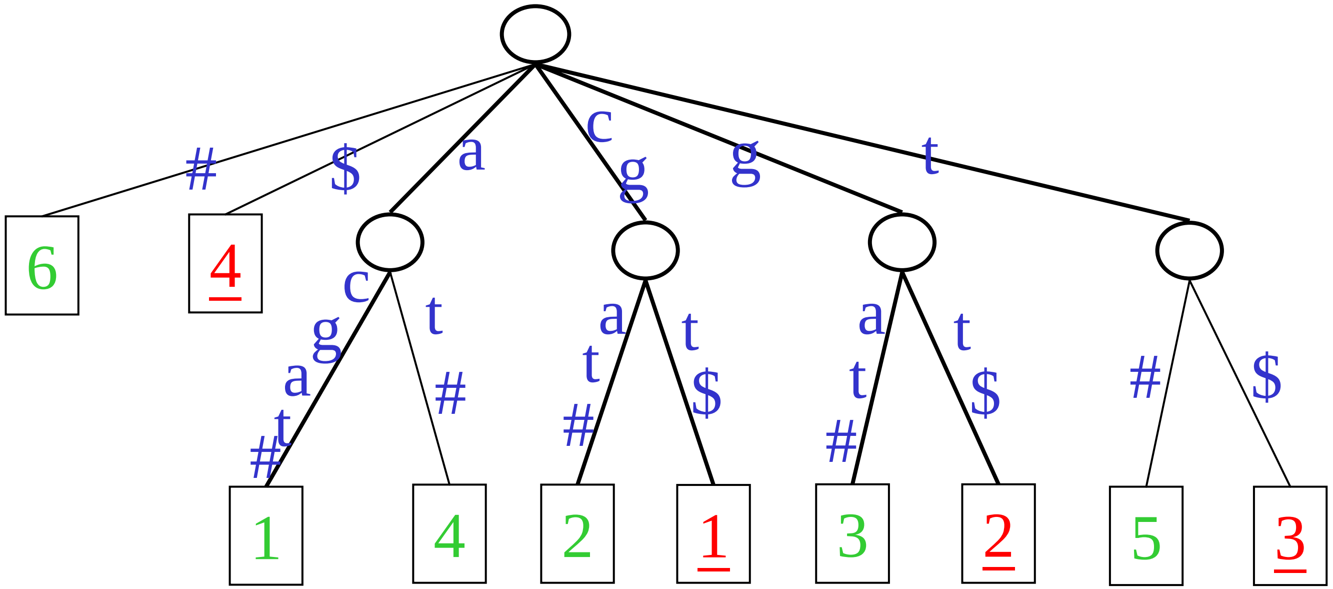


	1	2	3	4	5	6	7
S=	a	c	a	c	a	g	$

Note: The end index of every leaf edge should be 7, the last index of S. Thus, for leaf edges, we only need to store the start index.

Generalized suffix tree

- Build a suffix tree for two or more strings
- E.g. S_1 = acgat#, S_2 = cgt$



Straightforward construction of suffix tree

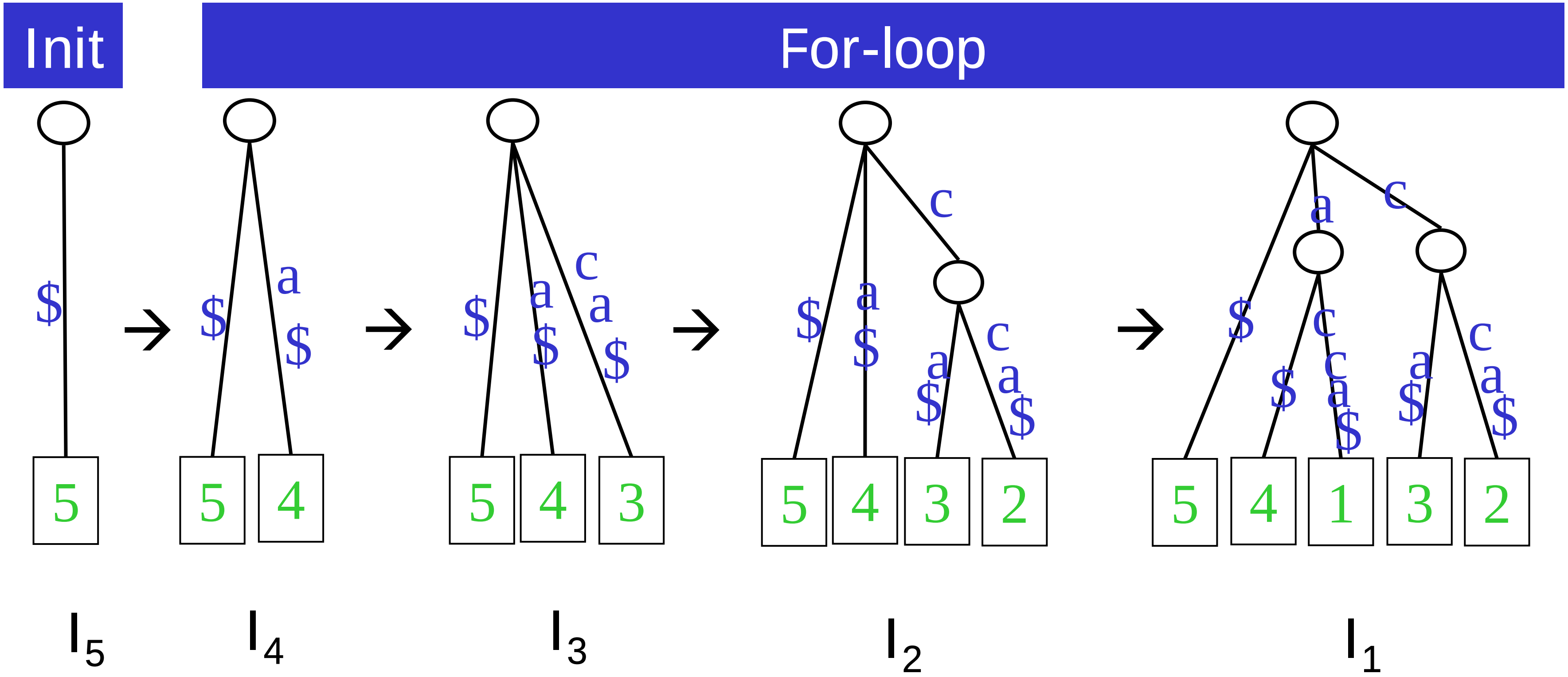
- Consider S = $s_1 s_2 \ldots s_n$ where s_n=$

- Algorithm:
  - Initialize the tree we only a root
  - For i = n to 1
    - Includes S[i..n] into the tree

- Time: $O(n^2)$

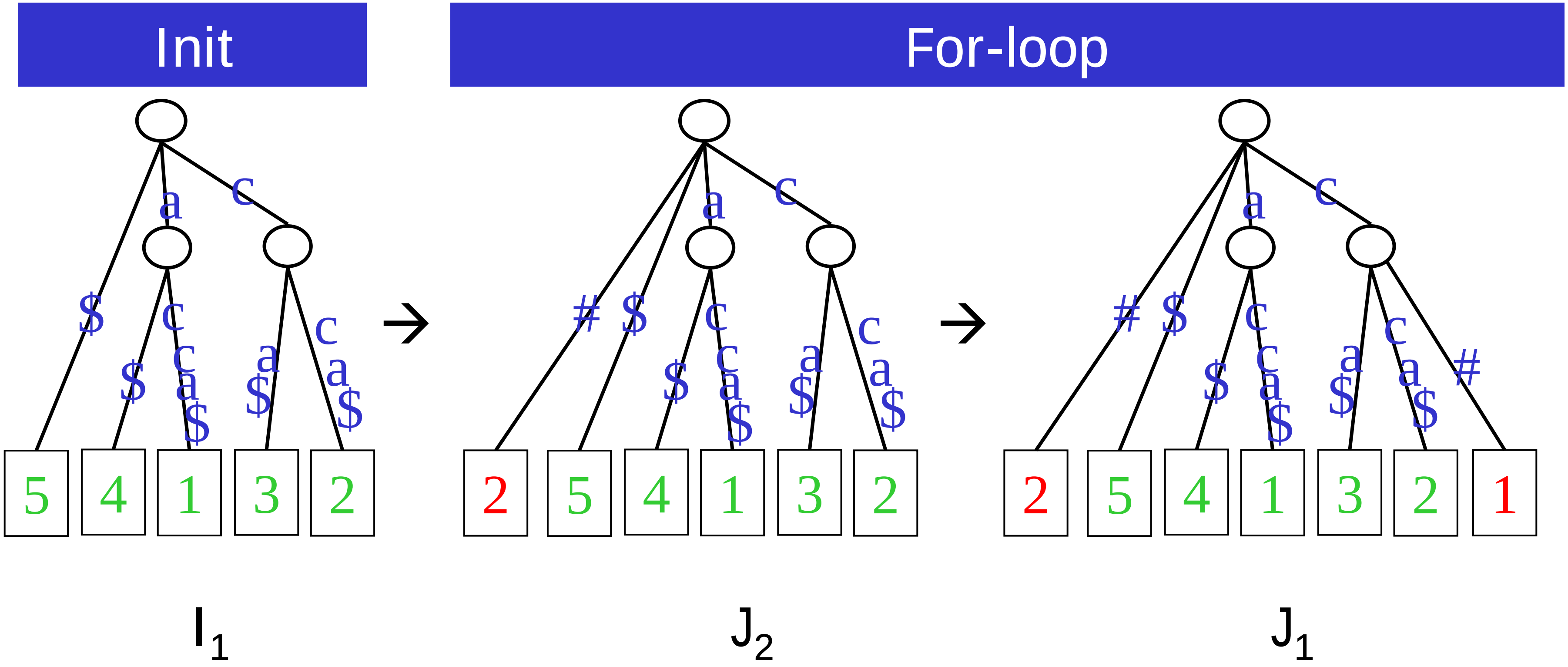
Example of construction

- S=acca$



Construction of generalized suffix tree

- S’= c#

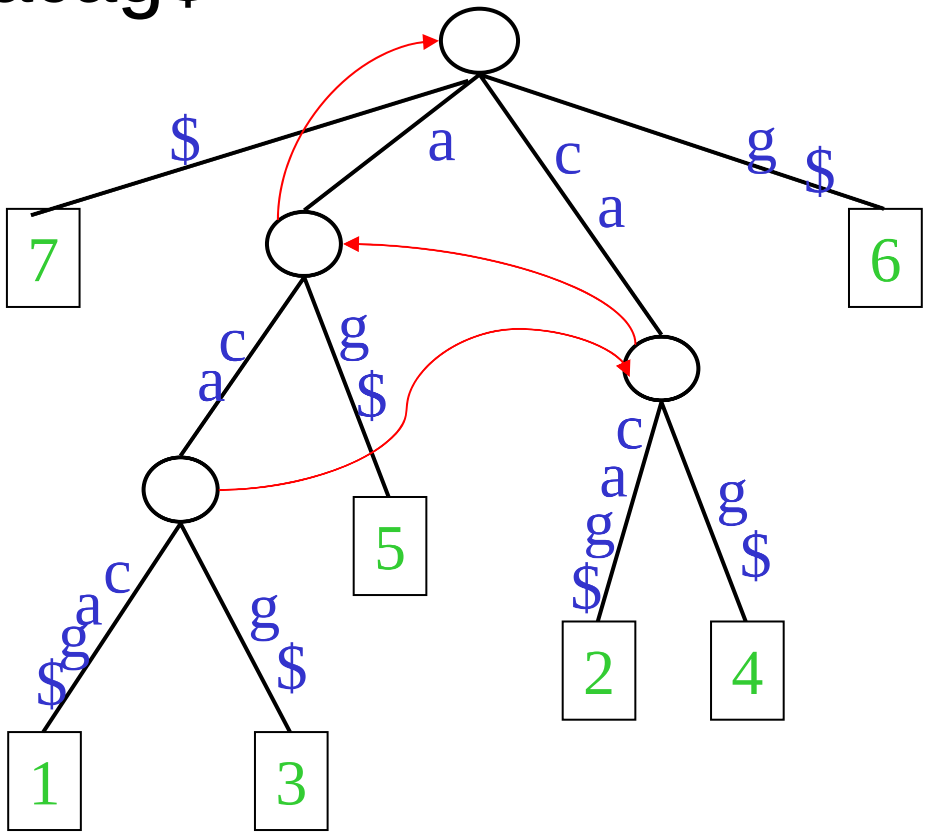


Property of suffix tree

- Fact: For any internal node v in the suffix tree, if the path label of v is $\alpha(v)$=ap, then
  - there exists another node w in the suffix tree such that $\alpha(w)$=p.
- Proof: Skip the proof.
- Definition of Suffix Link:
  - For any internal node v, define its suffix link sl(v) = w.

Suffix Link example

- S=acacag$



Can we construct a suffix tree in O(n) time?

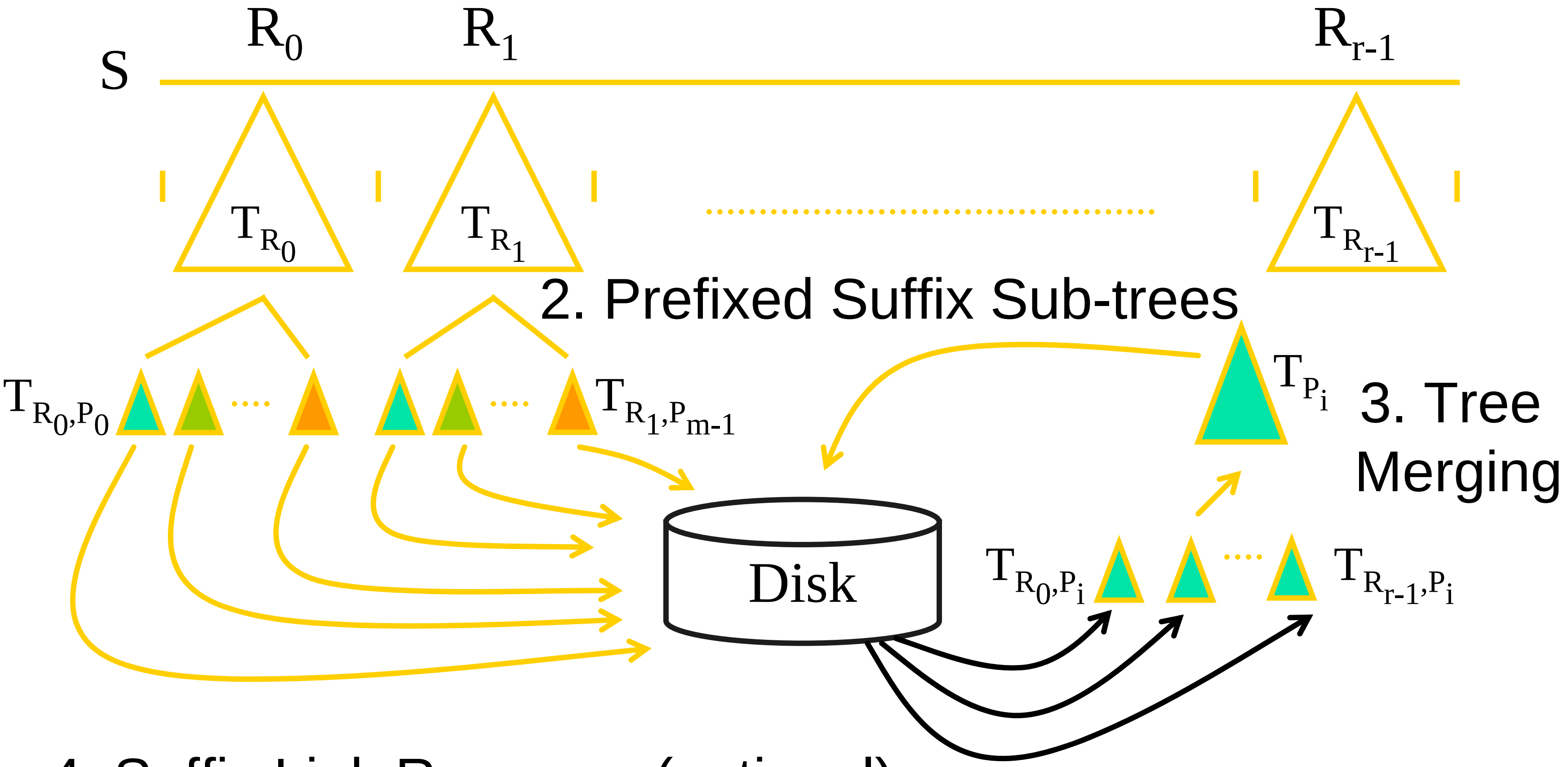
- Yes. We can construct it in O(n) time and O(n) space
  - Weiner's algorithm [1973]
    - Linear time for constant size alphabet, but much space
  - McGreight's algorithm [JACM 1976]
    - Linear time for constant size alphabet, quadratic space
  - Ukkonen's algorithm [Algorithmica, 1995]
    - Online algorithm, linear time for constant size alphabet, less space
  - Farach's algorithm [FOCS 1997]
    - Linear time for general alphabet
  - Hon,Sadakane, and Sung's algorithm [FOCS 2003]
    - O(n) bit space O(n $\log^e$n) time for 0<e<1
    - O(n) bit space O(n) time for suffix array construction
- But they are all in-memory algorithm that does not guarantee locality of processing

Trellis Algorithm

- A novel disk-based suffix tree construction algorithm designed specifically for DNA sequences
- Scales gracefully for very large genome sequences (i.e. human genome)
- Unlike existing algorithms,
  - Trellis exhibits no data skew problem
  - Trellis recovers suffix links quickly
  - Trellis has fast construction and query time
- Trellis is a 4-step algorithm

Trellis: Algorithm Overview

1. Variable-length prefixes: e.g. AA, ACA, ACC, …



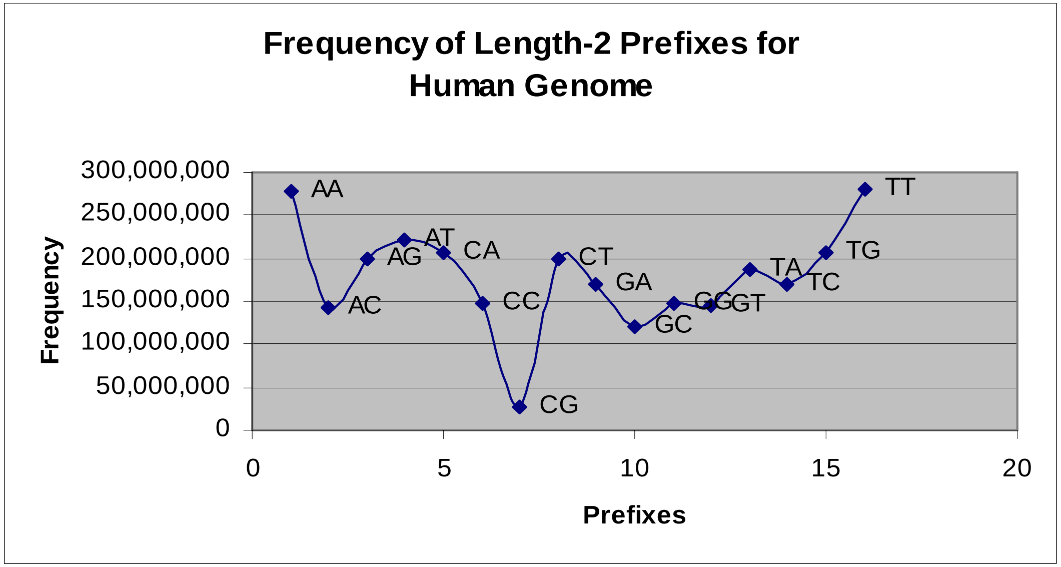
2. Prefixed Suffix Sub-trees

3. Tree Merging

4. Suffix Link Recovery (optional)

1. Variable-length Prefix Creation

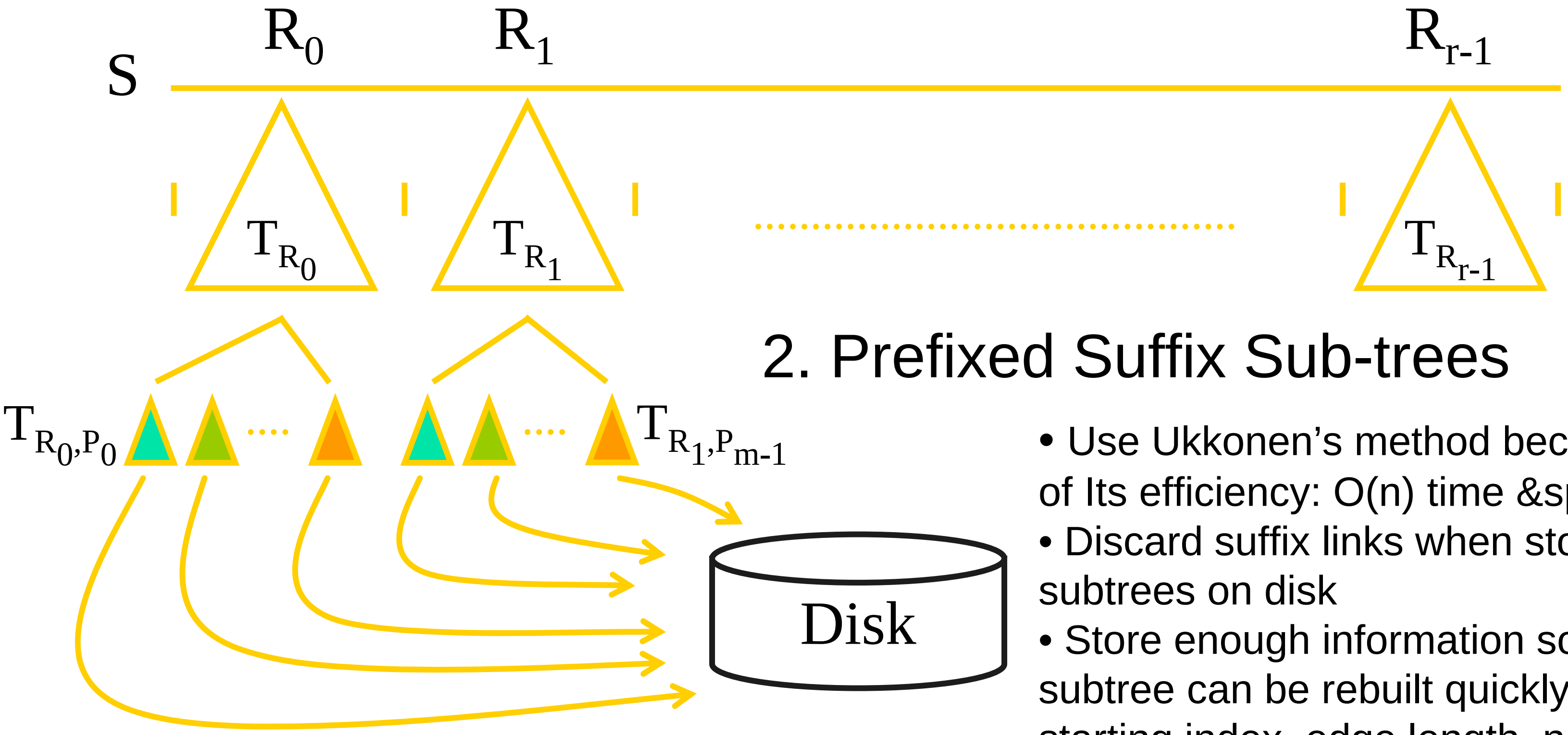
- *Goal:* Separate the complete suffix tree by prefixes of suffixes, such that each subtree can reside entirely in the available memory



Main Idea: Expand prefixes only as needed

2. Suffix Tree Partitioning

1. Variable-length prefixes: e.g. AA, ACA, ACC, …

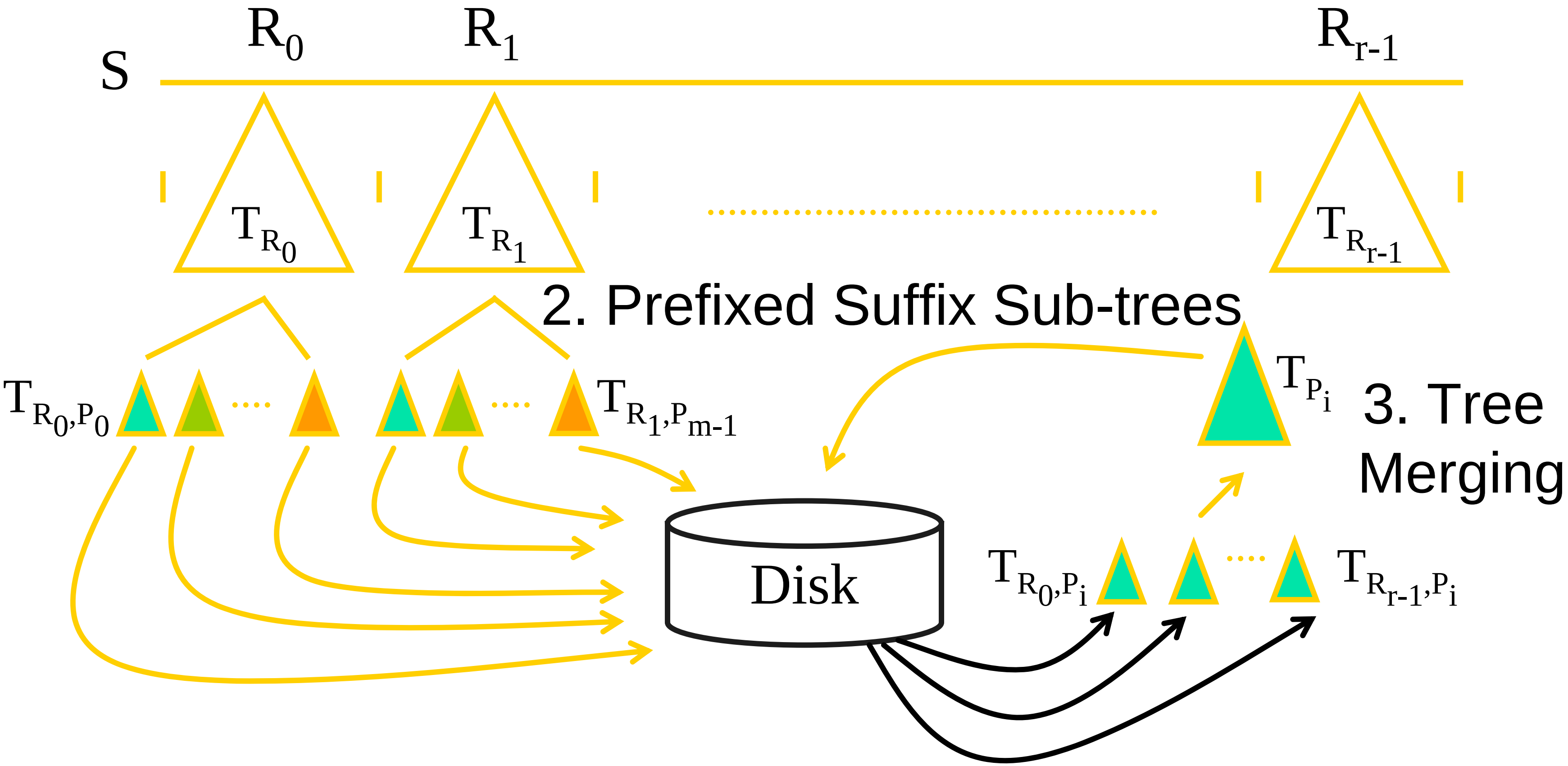


2. Prefixed Suffix Sub-trees

- Use Ukkonen’s method because of Its efficiency: O(n) time &space
- Discard suffix links when store the subtrees on disk
- Store enough information so that a subtree can be rebuilt quickly, e.g. edge starting index, edge length, node parent, etc.

3. Suffix Tree Merging

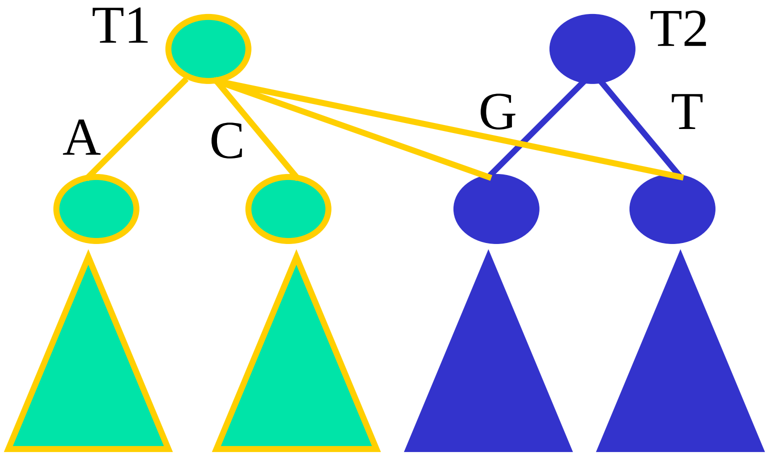
1. Variable-length prefixes: e.g. AA, ACA, ACC, …



2. Prefixed Suffix Sub-trees

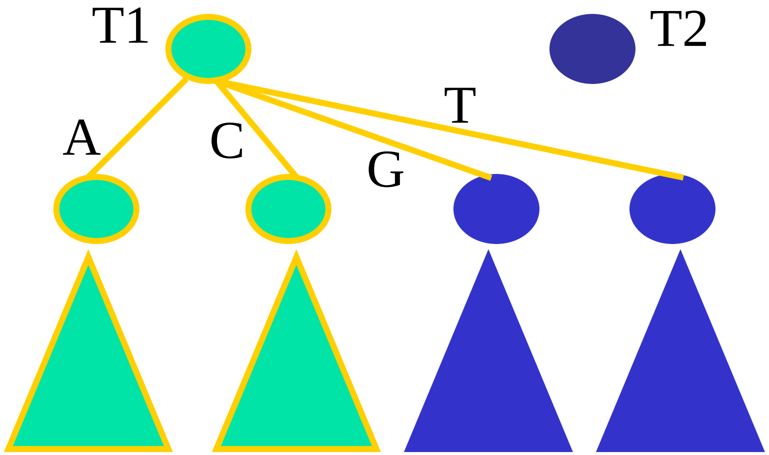
3. Tree Merging

Merge Algorithm



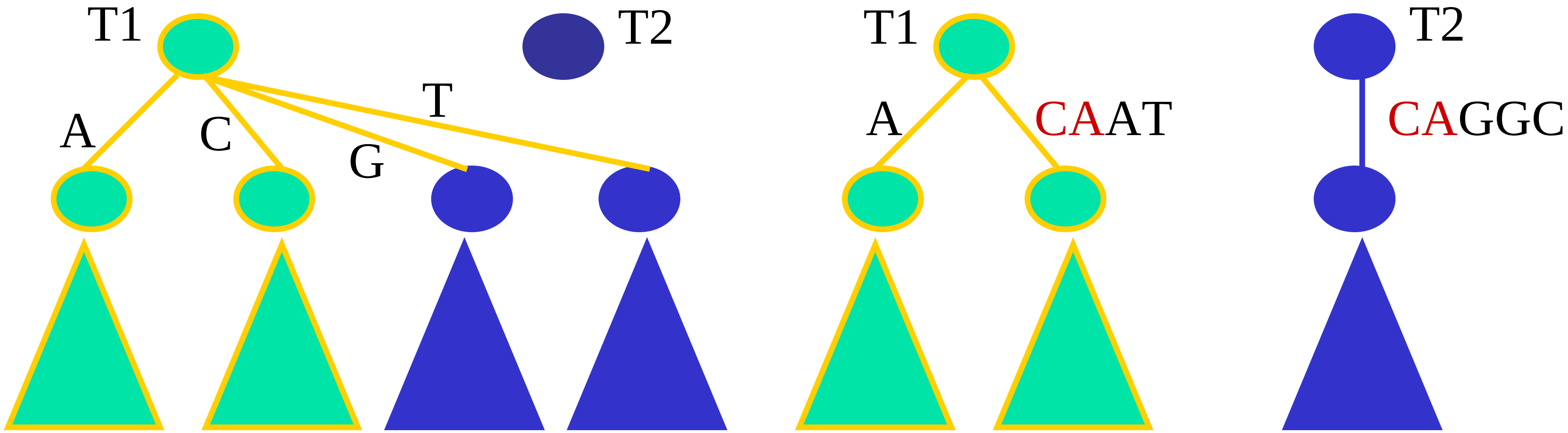
Case 1: No common prefix

Merge Algorithm



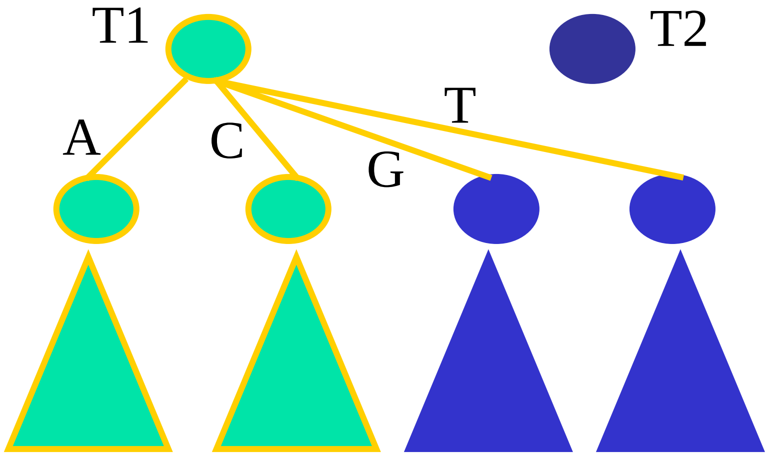
Case 1: No common prefix

Merge Algorithm

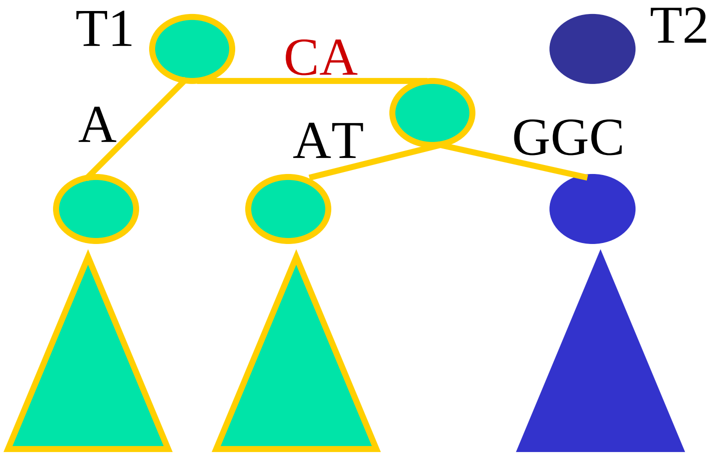


Case 1: No common prefix Case 2: Has common prefix

Merge Algorithm



Case 1: No common prefix



Case 2: Has common prefix

4. Suffix Link Recovery

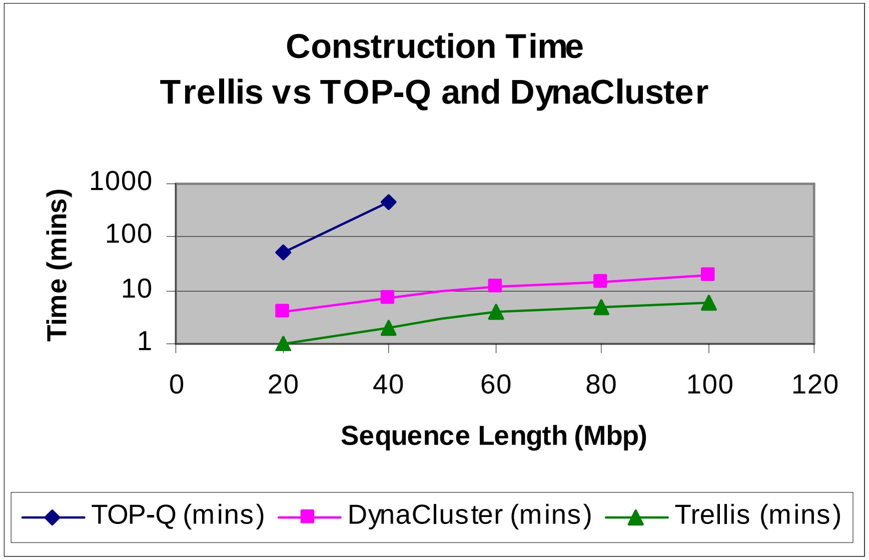
- Some internal nodes *have suffix links* from the Ukkonen’s algorithm in Step #1
- Some internal nodes are created in the merging step and do *not have suffix links*
- Discard all suffix link information from step #1 and stored suffix trees on disk (does not help speed this step up, so discard to simplify)
- Should suffix links are required, use the suffix link recovery algorithm to rebuild them

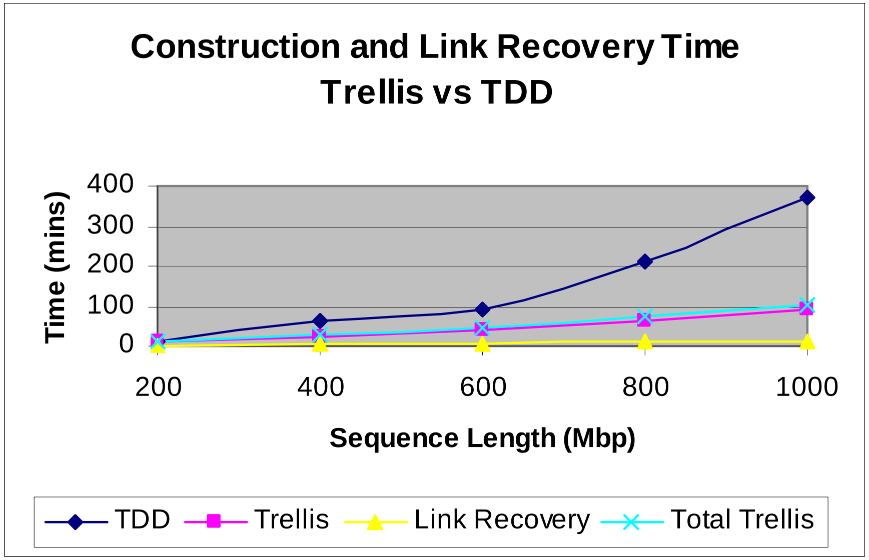
4. Suffix Link Recovery (cont.

- For each prefixed suffix tree, recursively call this function from the tree's root.
- x: an internal node
- L: be edge label between x and parent(x)

```
RECOVER(x, L)
    if (x == root)  sl(x) ← x;
    else {
        1. p = parent(x);
        2. q = sl(p); //get suffix link of p, and load the prefix tree
                       for q from disk if not in memory
        3. Skip/count using L to locate sl(x) under q; }
    for (each internal child y of x)
        RECOVER(y, edge-label(x,y));
```

Experimental Results





- Memory: 512 MB
- TOP-Q and DynaCluster parameters were set as recommended in their papers

- Memory: 512MB

Human genome suffix tree

(size ~3Gbp, using 2GB of memory)

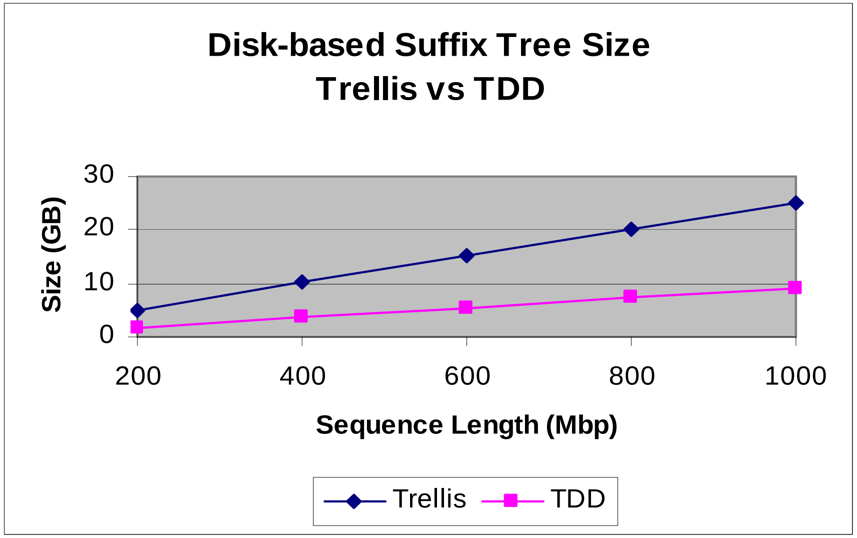
Trellis

- Without links: 4.2hr
- With links: 5.9hr

TDD: 12.8hr

Experimental Results (cont.)

- Disk Space Usage



Human Genome

Trellis	TDD
72GB	54GB

On average, Trellis uses about 27 bytes per character indexed while TDD uses about 9.7 bytes.

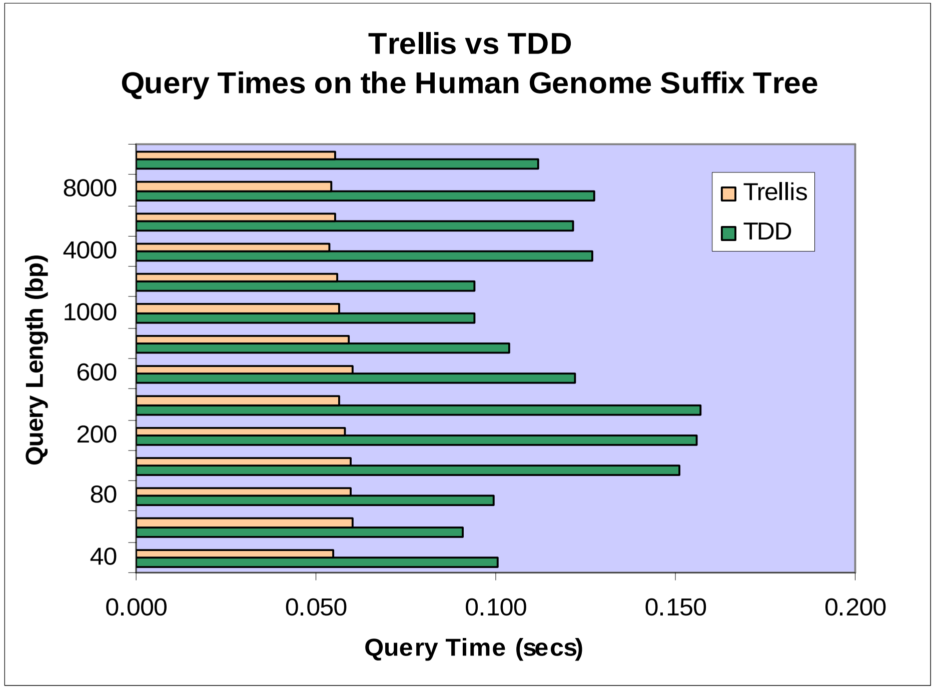
For the human genome, TDD uses about 19.3 bytes/char because it requires 64-bit environment to index larger sequences.

Trellis remains at 27 bytes/char for the human genome.

Disk-space vs query time tradeoff

Experimental Results (cont.)

- Query time (without suffix links)



Hence, faster query time!

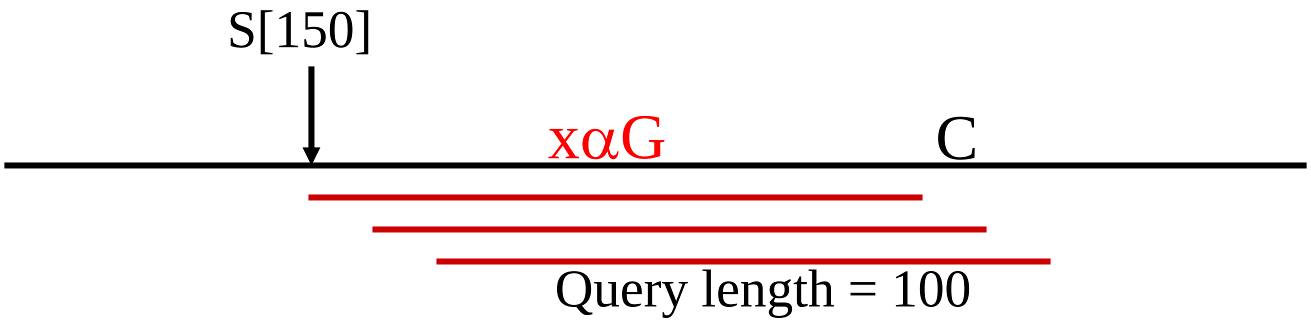
TDD

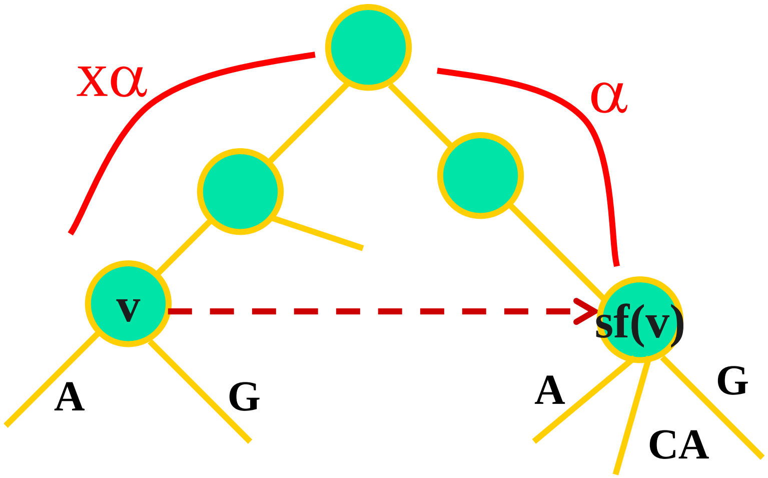
- *smaller* suffix trees
- edge length must be determined by examining *all* children nodes
- each internal node only has a pointer to its *first* child, i.e. children must be *linearly* scanned during a query search

Trellis

- *larger* suffix trees
- edge length stored *locally* with its respective node
- all children locations stored *locally*, so each child can be accessed in a constant time, i.e. no linear scan needed

Experimental Results (cont.)

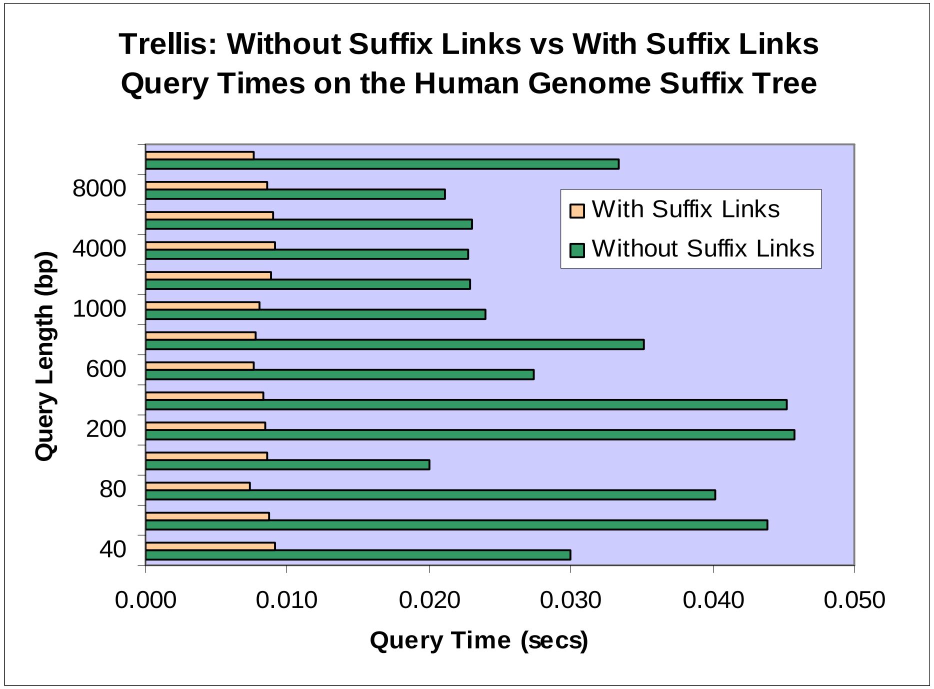




- Uses suffix links to move across the tree to search for the next query
- Mimics the behavior of exact match anchor search during a genome alignment

Experiment Results (cont.)

- Query time (with suffix links)



Summary

- Trellis builds a disk-based suffix tree based on
  - A partitioning method via variable-length prefixes
  - A suffix subtree merging algorithm
- Trellis is both time and space efficient
- Trellis quickly recovers suffix links
- Faster than existing leading methods in both construction and query time

Future Work

- Input sequence larger than the human genome (more than 3Gbp)
- Wider range of alphabets, e.g. protein alphabet and English text
- Parallelize Trellis

Question?

Outline

- Types of sequences
- Foundation
  - Full matching: Building a disk based suffix tree
  - Approximate matching Using vgrams
- Technique & Application
  - Finding global partial order in sequence
  - Finding motif in sequence

Example 1: a movie database

Tom

Find movies starred Samuel Jackson

Star	Title	Year	Genre
Keanu Reeves	The Matrix	1999	Sci-Fi
Samuel Jackson	Star Wars: Episode III - Revenge of the Sith	2005	Sci-Fi
Schwarzenegger	The Terminator	1984	Sci-Fi
Samuel Jackson	Goodfellas	1990	Drama
...	...	...	...

How about Schwarrzenger?

The user doesn’t know the exact spelling!

Star	Title	Year	Genre
Keanu Reeves	The Matrix	1999	Sci-Fi
Samuel Jackson	Star Wars: Episode III - Revenge of the Sith	2005	Sci-Fi
Schwarzenegger	The Terminator	1984	Sci-Fi
Samuel Jackson	Goodfellas	1990	Drama
…	…	…	…

Relax Condition

Find movies with a star “similar to” Schwarrzenger.

Star	Title	Year	Genre
Keanu Reeves	The Matrix	1999	Sci-Fi
Samuel Jackson	Star Wars: Episode III - Revenge of the Sith	2005	Sci-Fi
Schwarzenegger	The Terminator	1984	Sci-Fi
Samuel Jackson	Goodfellas	1990	Drama
...	...	...	...

Edit Distance

- Given two strings A and B, edit A to B with the minimum number of edit operations:
  - Replace a letter with another letter
  - Insert a letter
  - Delete a letter
- E.g.

  -
    ```
    A = interestings      _i__nterestings
    B = bioinformatics    bioinformatic_s
                          101101101100110
    ```
  - Edit distance = 9

Edit Distance Computation

- Instead of minimizing the number of edge operations, we can associate a cost function to the operations and minimize the total cost. Such cost is called edit distance.
- For the previous example, the cost function is as follows:

  -
    ```
    A= _i__nterestings
    B= bioinformatic_s
       1011011011100110
    ```
  - Edit distance = 9

	_	A	C	G	T
_		1	1	1	1
A	1	0	1	1	1
C	1	1	0	1	1
G	1	1	1	0	1
T	1	1	1	1	0

Needleman-Wunsch algorithm (I)

- Consider two strings S[1..n] and T[1..m].
- Define V(i, j) be the score of the optimal alignment between S[1..i] and T[1..j]
- Basis:
  - $V(0, 0) = 0$
  - $V(0, j) = V(0, j-1) + \delta(\_, T[j])$
    - Insert j times
  - $V(i, 0) = V(i-1, 0) + \delta(S[i], \_)$
    - Delete i times

Needleman-Wunsch algorithm (II)

- Recurrence: For i>0, j>0
  - $$V(i,j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{Match/mismatch} \\ V(i-1, j) + \delta(S[i], \_) & \text{Delete} \\ V(i, j-1) + \delta(\_, T[j]) & \text{Insert} \end{cases}$$
- In the alignment, the last pair must be either match/mismatch, delete, insert.

```
xxx…xx          xxx…xx          xxx…x_
      |               |               |
xxx…yy          yyy…y_          yyy…yy
match/mismatch  delete          insert
```

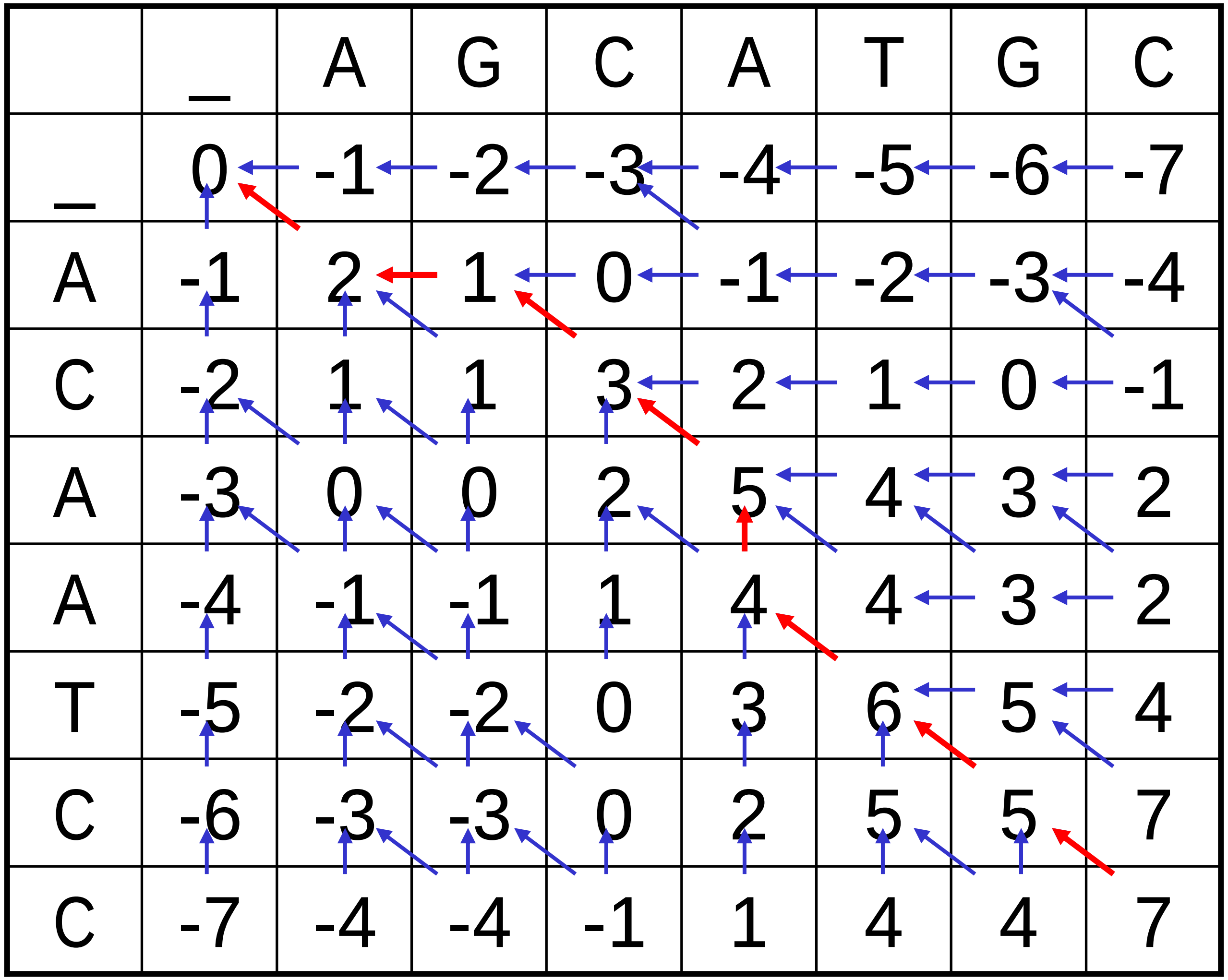
Example (I)

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2							
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

Example (II)

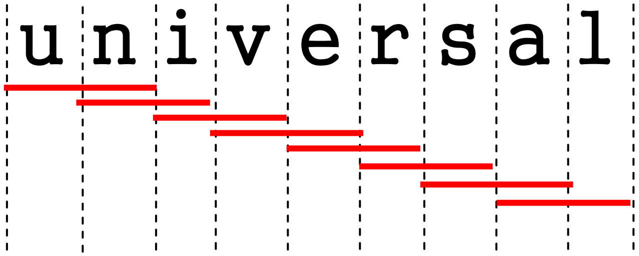
	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2			
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

Example (III)



	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

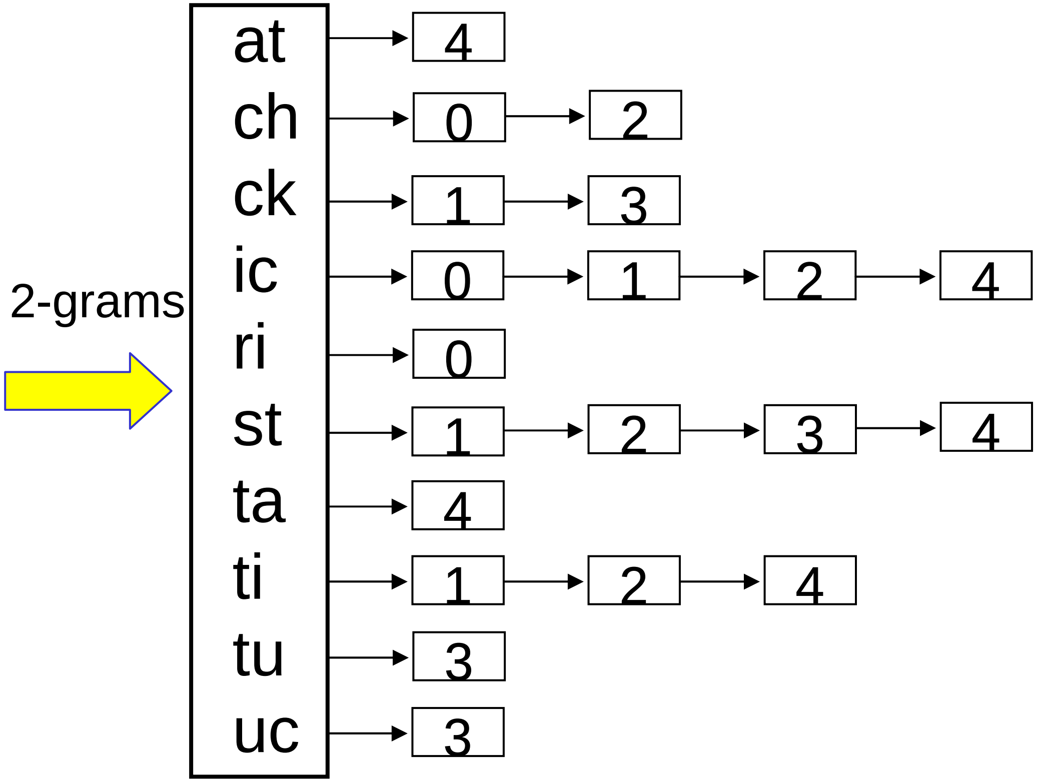
“q-grams” of strings



2-grams

q-gram inverted lists

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static



2-grams

at → 4
ch → 0 → 2
ck → 1 → 3
ic → 0 → 1 → 2 → 4
ri → 0
st → 1 → 2 → 3 → 4
ta → 4
ti → 1 → 2 → 4
tu → 3
uc → 3

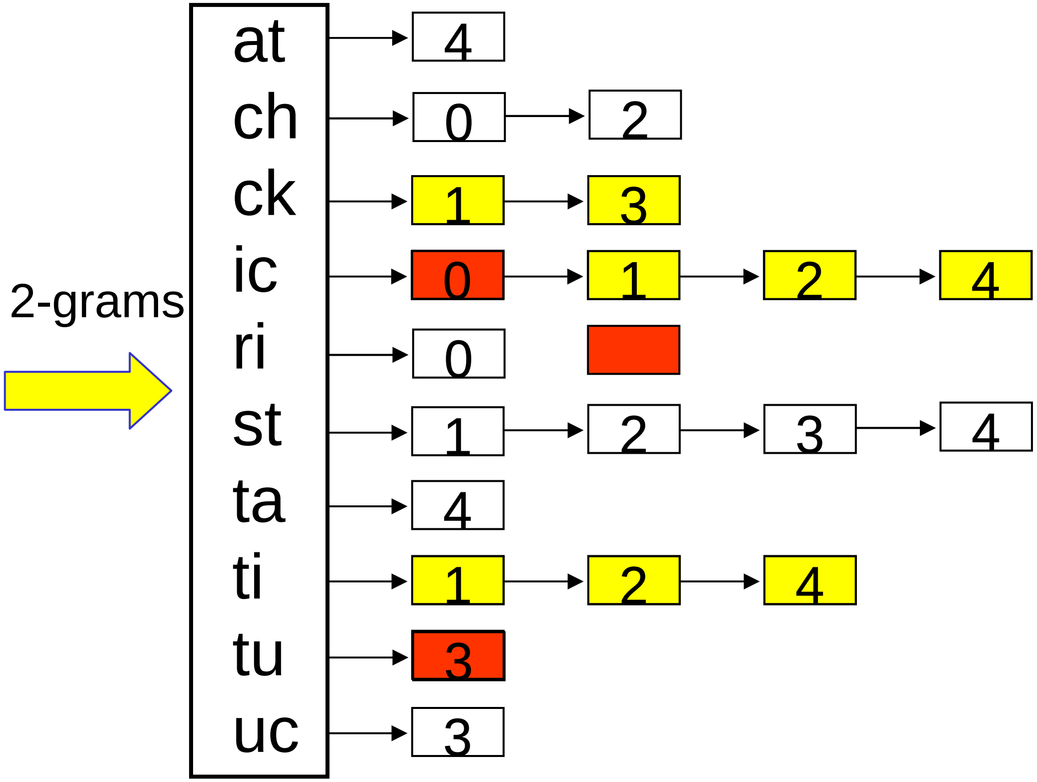
Searching using inverted lists

- Query: “shtick”, ED(shtick, ?)≤1

sh ht ti ic ck

# of common grams >= 3

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static



2-grams

at → 4

ch → 0 → 2

ck → 1 → 3

ic → 0 → 1 → 2 → 4

ri → 0

st → 1 → 2 → 3 → 4

ta → 4

ti → 1 → 2 → 4

tu → 3

uc → 3

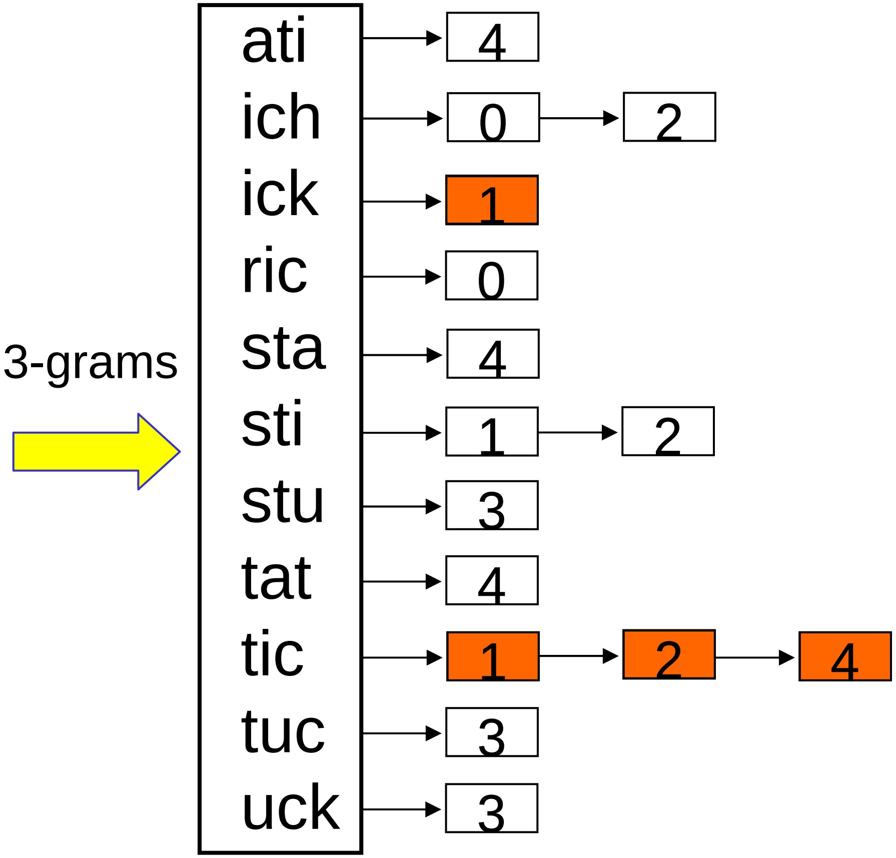
2-grams -> 3-grams?

- Query: “shtick”, ED(shtick, ?)⩽1

sht hti tic ick

# of common grams >= 1

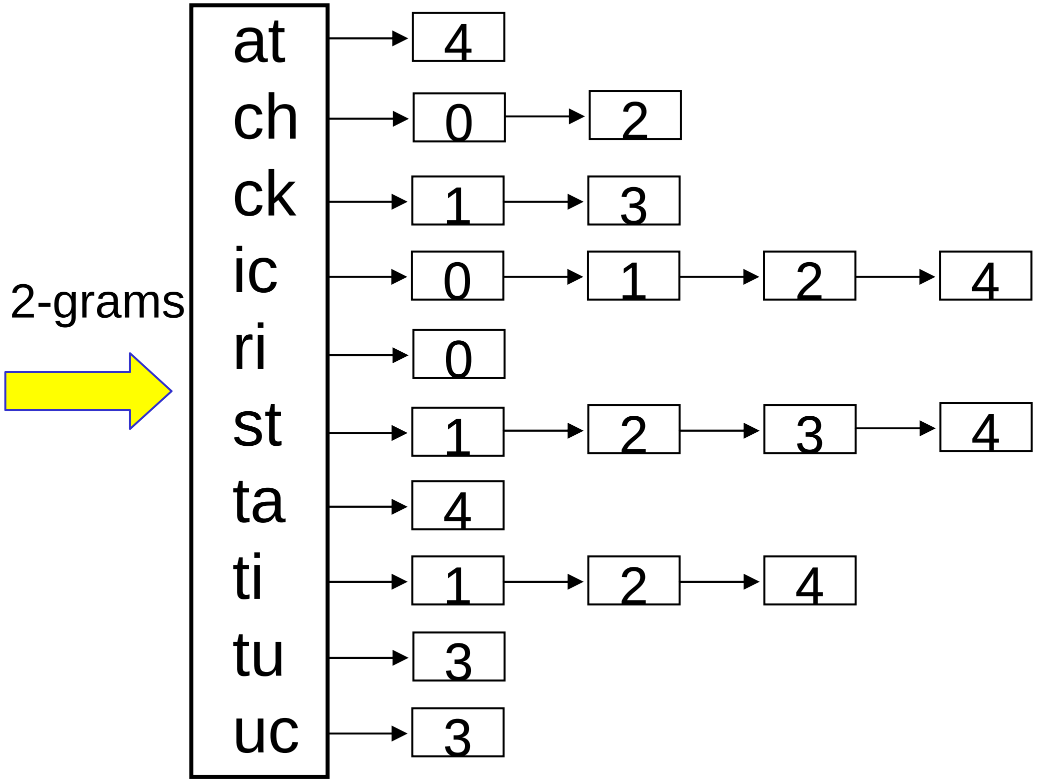
id	strings
0	rich
1	stick
2	stich
3	stuck
4	static



Observation 1: dilemma of choosing “q”

- Increasing “q” causing:
  - Longer grams → Shorter lists
  - Smaller # of common grams of similar strings

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

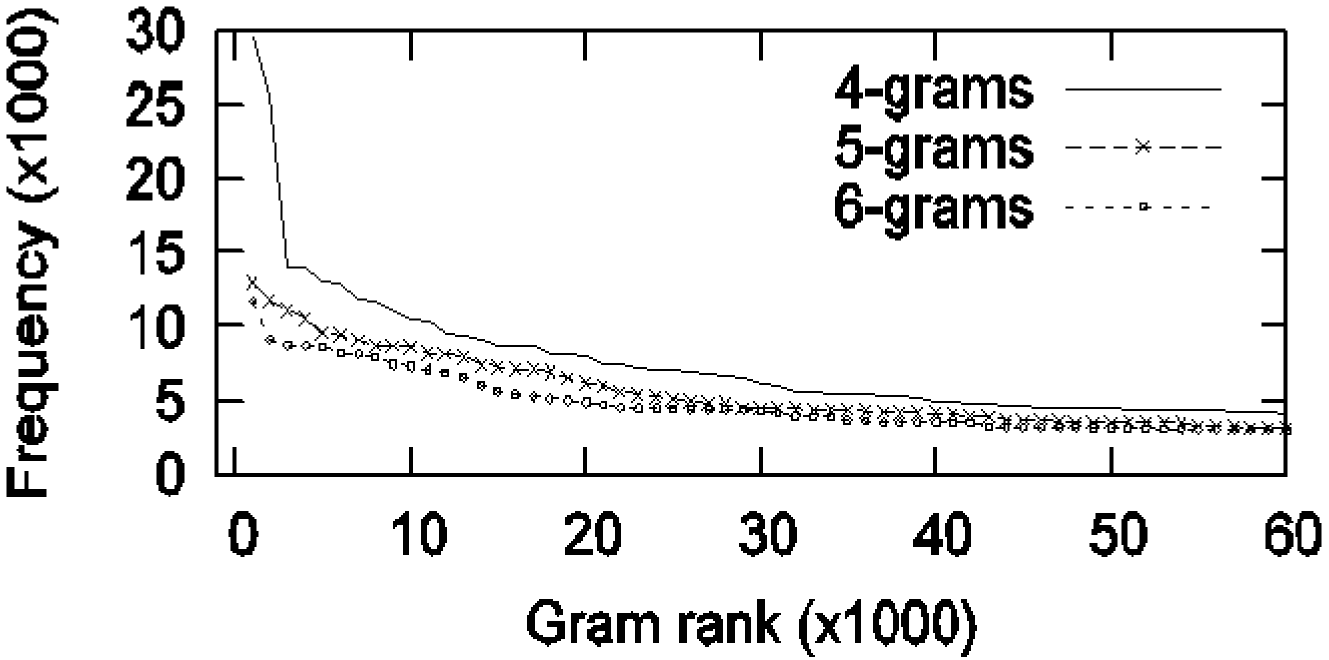


2-grams

at → 4
ch → 0 → 2
ck → 1 → 3
ic → 0 → 1 → 2 → 4
ri → 0
st → 1 → 2 → 3 → 4
ta → 4
ti → 1 → 2 → 4
tu → 3
uc → 3

Observation 2: skew distributions of gram frequencies

- DBLP: 276,699 article titles
- Popular 5-grams: ation (>114K times), tions, ystem, catio



VGRAM: Main idea

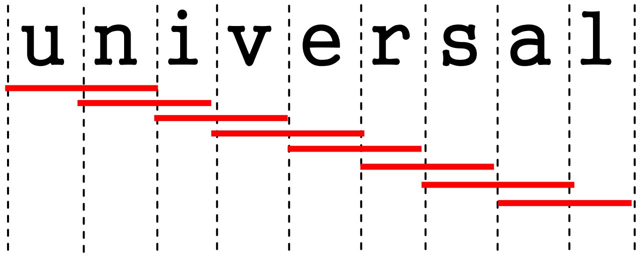
- Grams with variable lengths (between q_{min} and q_{max})
  - zebra
    - ze(123)
  - corrasion
    - co(5213), cor(859), corr(171)
- Advantages
  - Reduce index size ☺
  - Reducing running time ☺
  - Adoptable by many algorithms ☺

Challenges

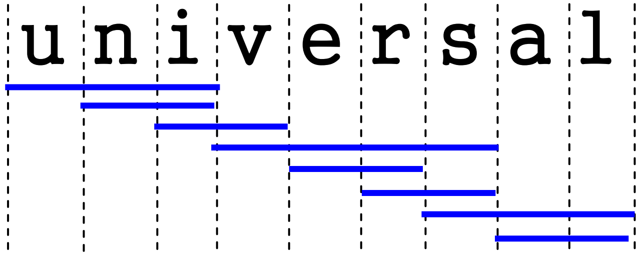
- Generating variable-length grams?
- Constructing a high-quality gram dictionary?
- Relationship between string similarity and their gram-set similarity?
- Adopting VGRAM in existing algorithms?

Challenge 1: String → Variable-length grams?

- Fixed-length 2-grams



- Variable-length grams

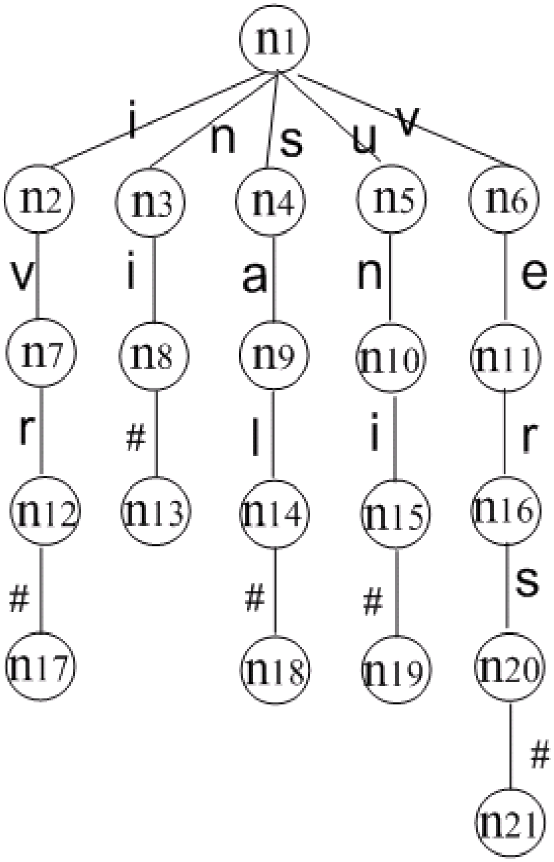


[2,4]-gram dictionary

ni
ivr
sal
uni
vers

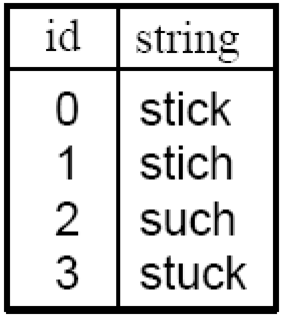
Representing gram dictionary as a trie

ni
ivr
sal
uni
vers



Challenge 2: Constructing gram dictionary

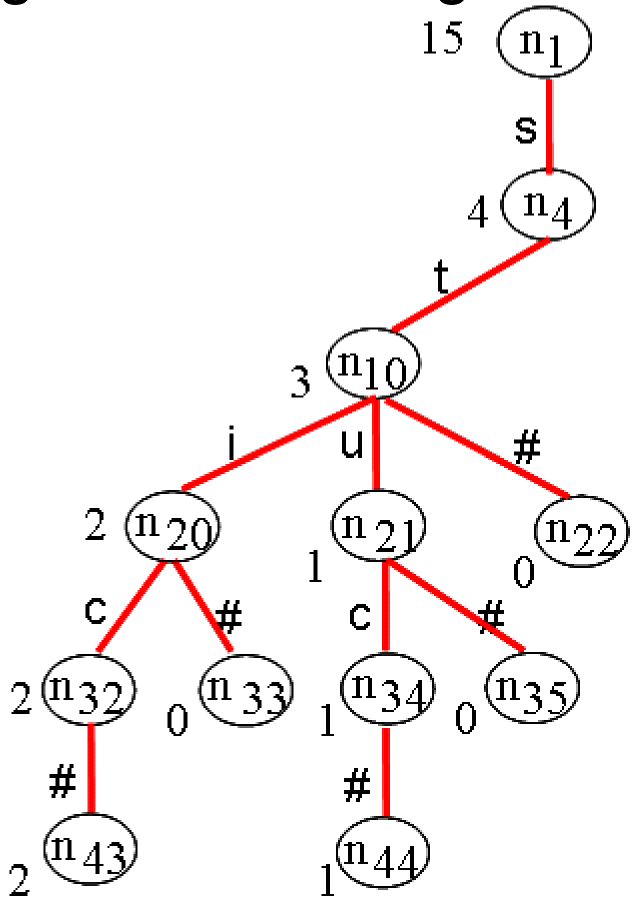
Step 1: Collecting frequencies of grams with length in [qmin, qmax]



id	string
0	stick
1	stich
2	such
3	stuck

(a) strings

st → 0, 1, 3
sti→ 0, 1
stu→3
stic→ 0, 1
stuc→3

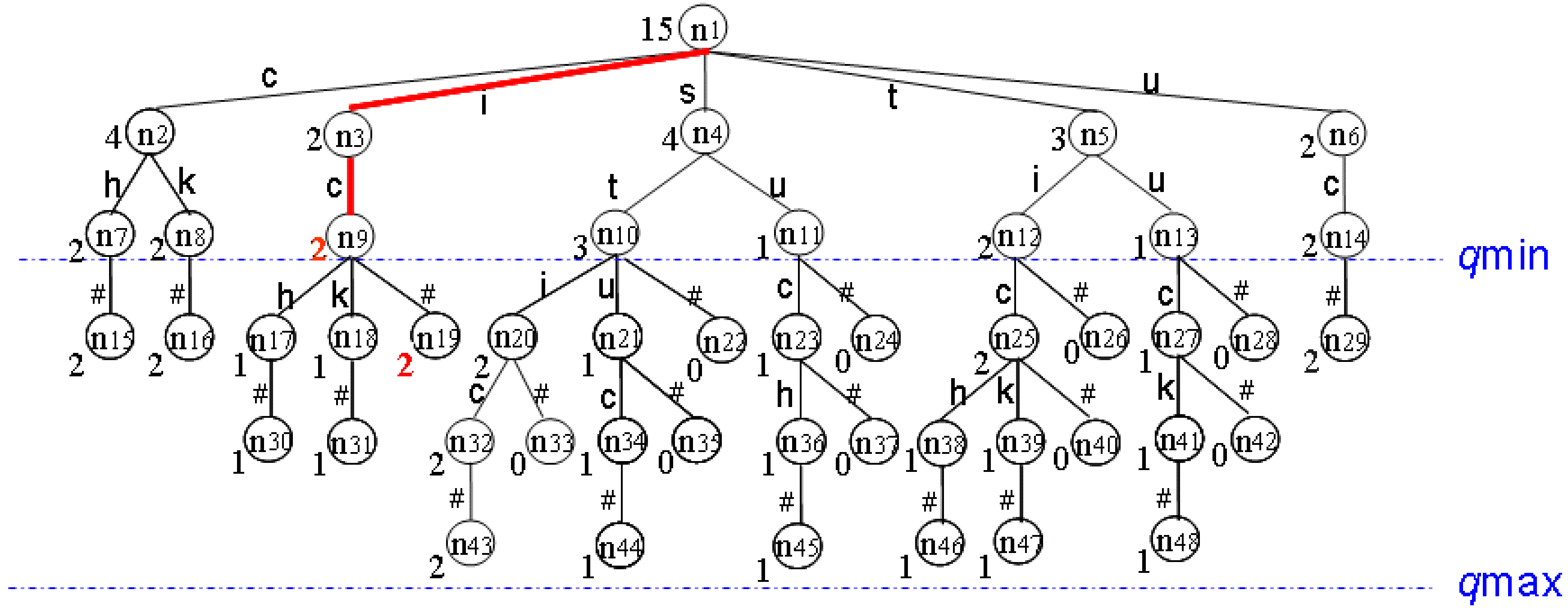


Gram trie with frequencies

Step 2: selecting grams

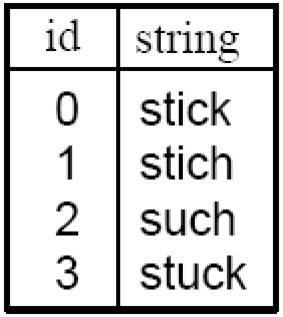
- Pruning trie using a frequency threshold *T* (e.g., 2)

id	string
0	stick
1	stich
2	such
3	stuck



A gram-frequency trie: [2,4]-gram

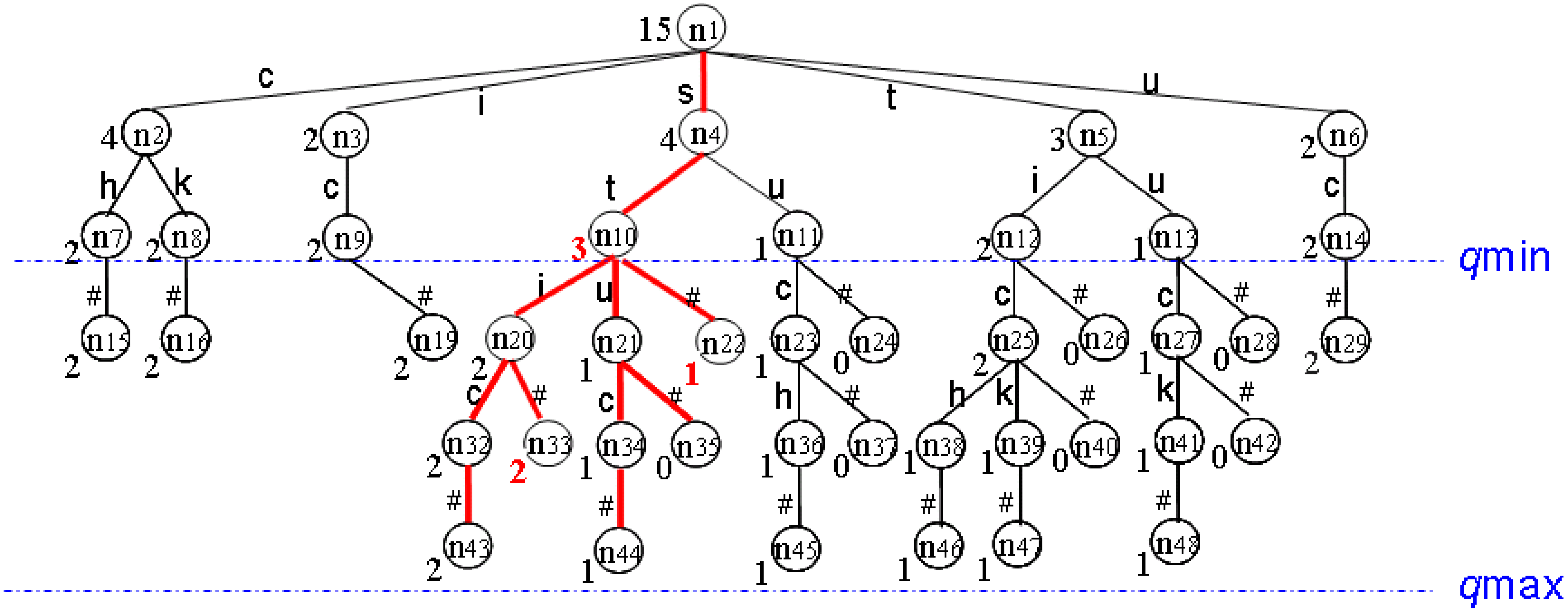
Step 2: selecting grams (cont)



id	string
0	stick
1	stich
2	such
3	stuck

(a) strings

Threshold T = 2

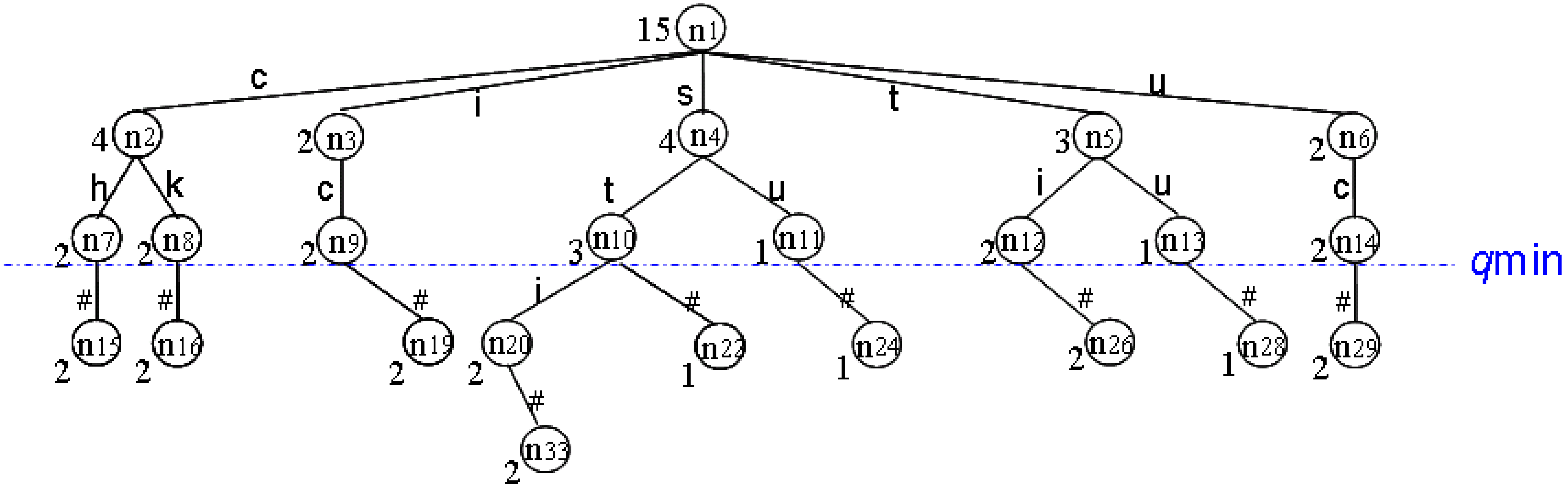


A gram-frequency trie: [2,4]-gram

Final gram dictionary

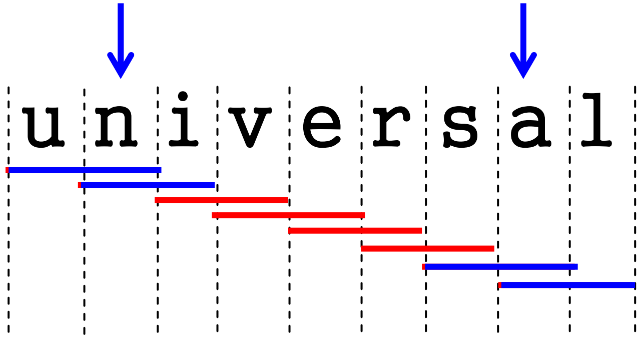
id	string
0	stick
1	stich
2	such
3	stuck

(a) strings



[2,4]-grams

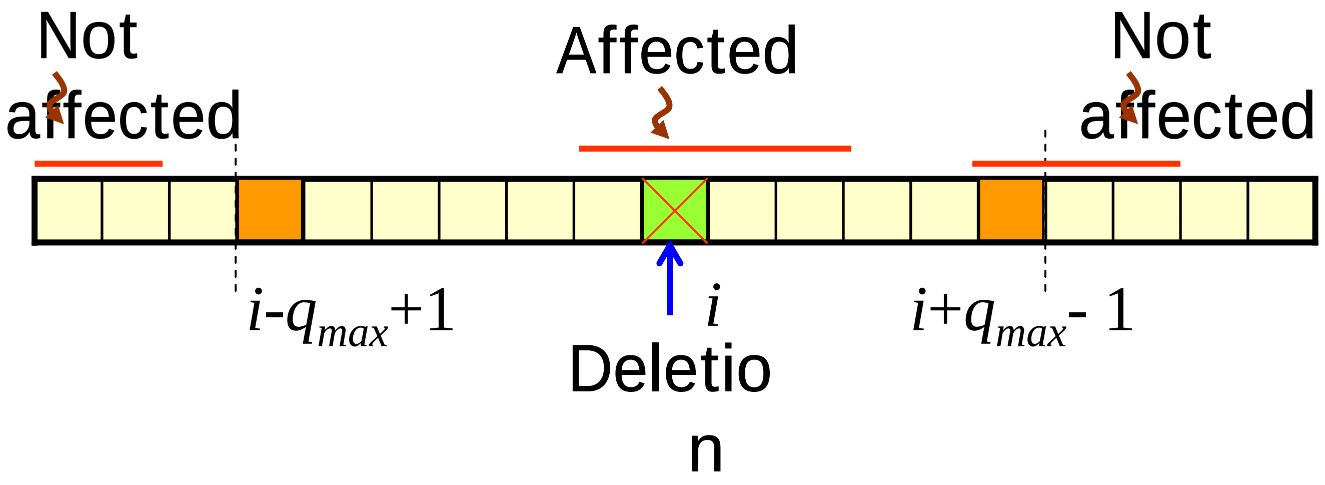
Challenge 3: Edit operation's effect on grams



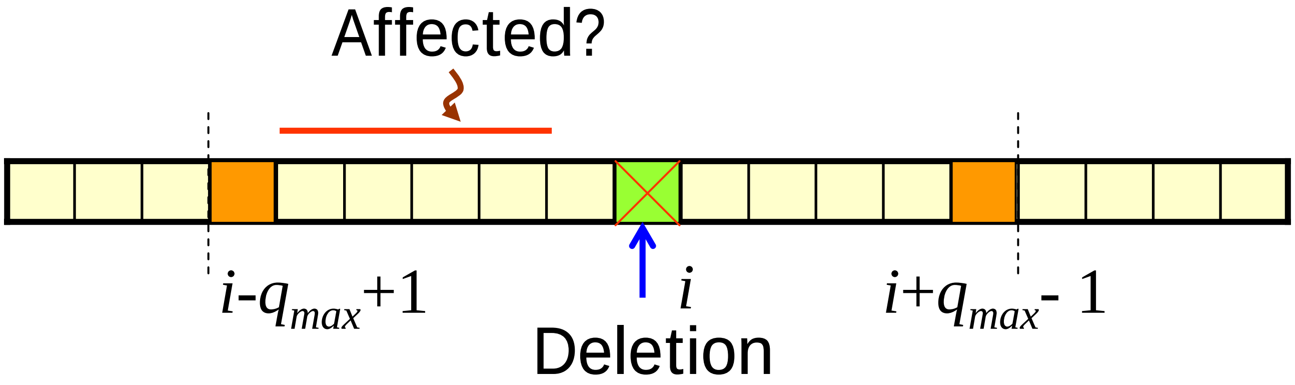
Fixed length: *q*

k operations could affect *k * q* grams

Deletion affects variable-length grams



Grams affected by a deletion



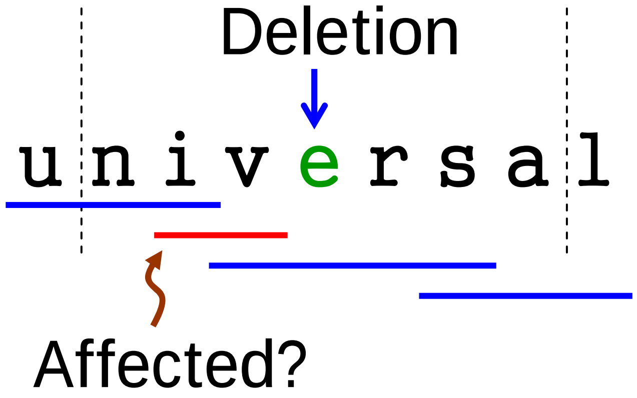
Affected?

$i-q_{max}+1$

i

Deletion

$i+q_{max}-1$



Deletion

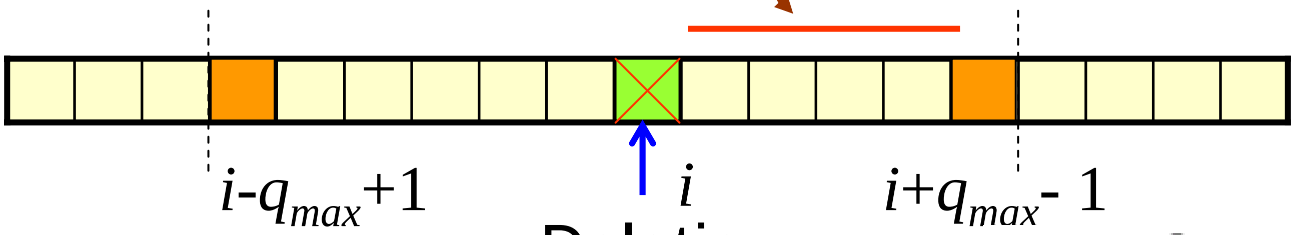
universal

Affected?

[2,4]-grams

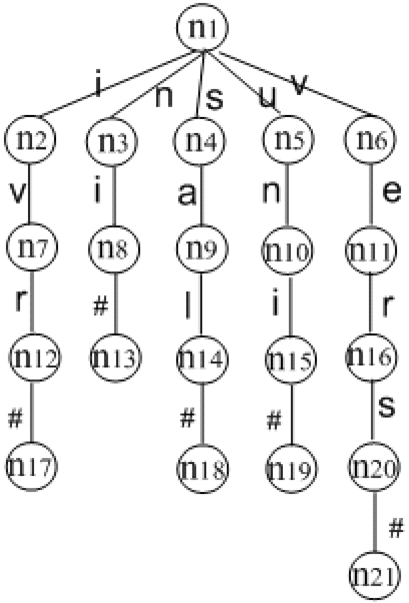
ni
ivr
sal
uni
vers

Grams affected by a deletion (cont)

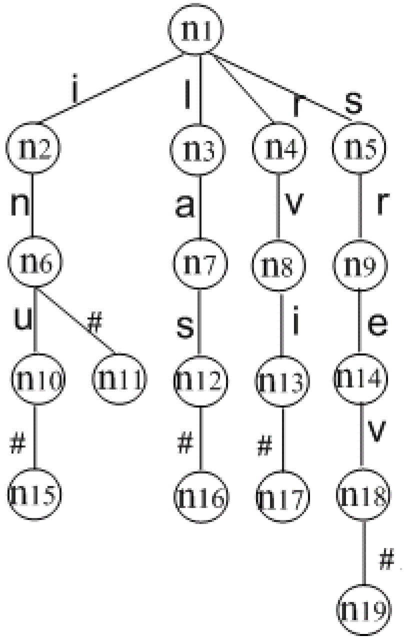


Affected?

Deletio
n

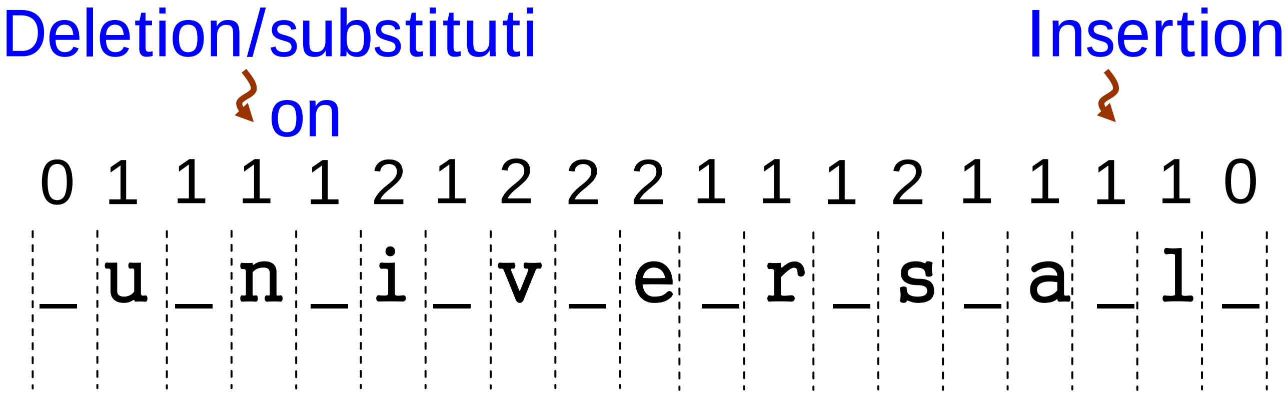


Trie of grams



Trie of reversed grams

# of grams affected by each operation



Deletion/substitution

Insertion

0 1 1 1 1 2 1 2 2 2 1 1 1 2 1 1 1 1 0

_ u _ n _ i _ v _ e _ r _ s _ a _ l _

Max # of grams affected by k operations

Vector of s = <2,4,6,8,9>

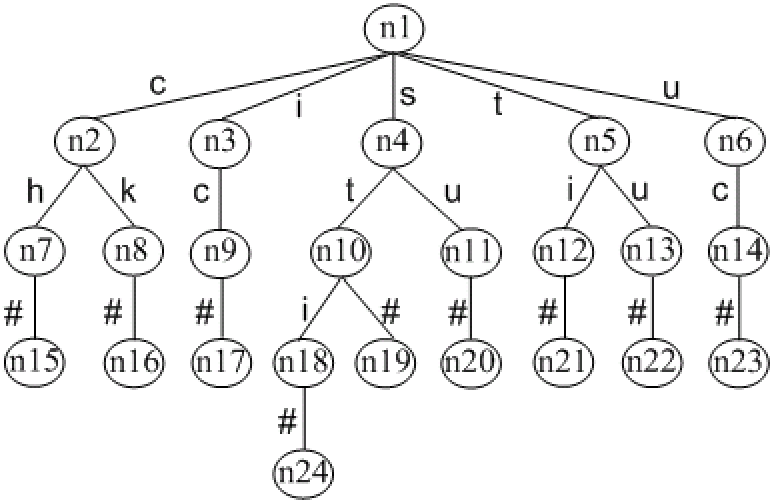
With 2 edit operations, at most 4 grams can be affected

- Called NAG vector (# of affected grams)
- Precomputed and stored

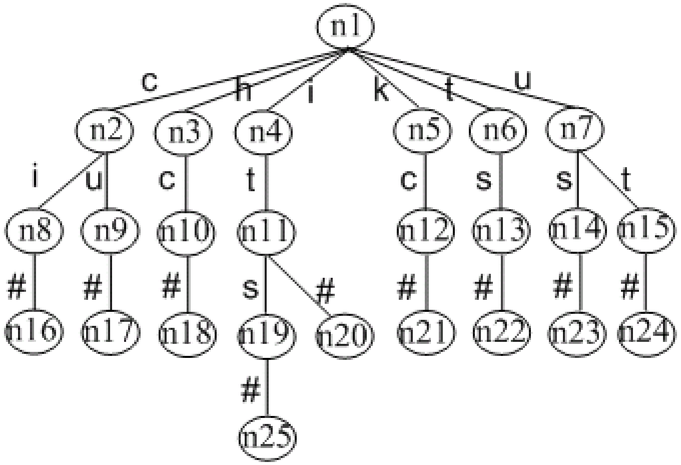
Summary of VGRAM index

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings



(b) Gram dictionary as a trie



(c) Reversed-gram trie

id	NAG vector
0	2, 3
1	2, 3
2	2, 3
3	3, 4

(d) NAG vectors

Challenge 4: adopting VGRAM

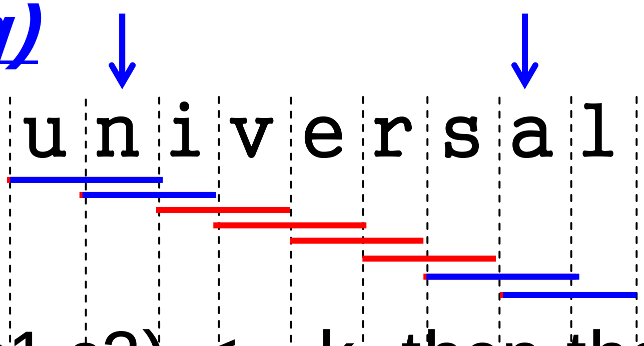
Easily adoptable by many algorithms

Basic interfaces:

- String s → grams
- String s1, s2 such that ed(s1,s2) <= k → min # of their common grams

Lower bound on # of common grams

Fixed length (*q*)



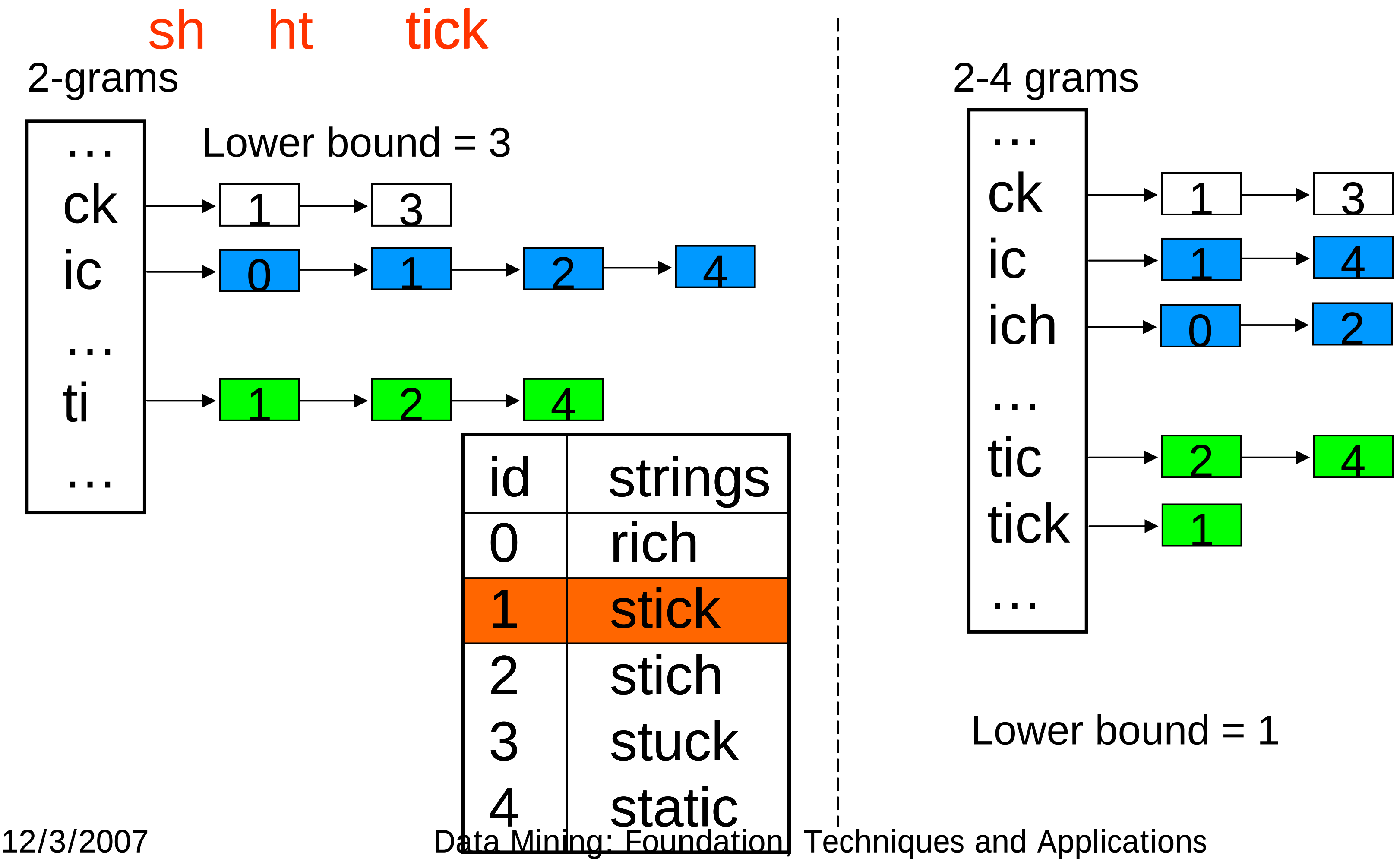
If ed(s1,s2) <= k, then their # of common grams >=:

$$(|s_1|- q + 1) – k * q$$

Variable lengths: # of grams of s1 – NAG(s1,k)

Example: algorithm using inverted lists

- Query: “shtick”, ED(shtick, ?)≤1



id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

PartEnum + VGRAM

PartEnum, fixed q-grams:

$$ed(s1,s2) <= k$$

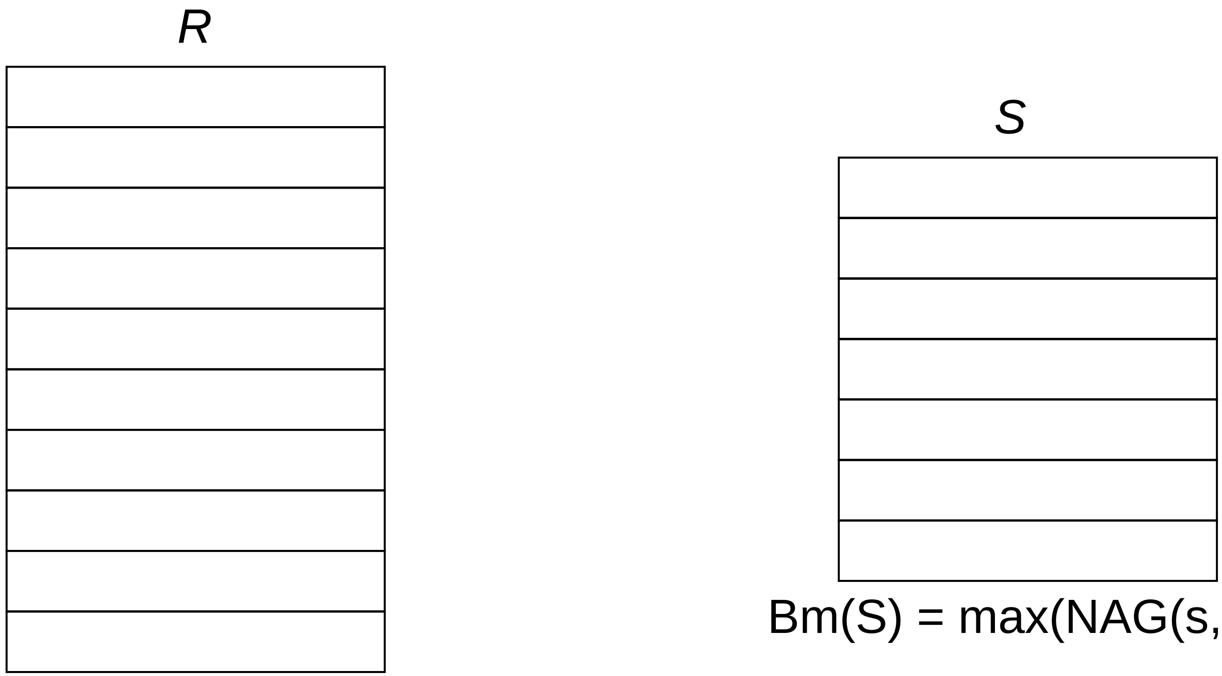
➔ hamming(grams(s1),grams(s2)) <= k * q

VGRAM:

$$ed(s1,s2) <= k$$

➔ hamming(VG (s1),VG(s2)) <= NAG(s1,k) + NAG(s2,k)

PartEnum + VGRAM (naïve)

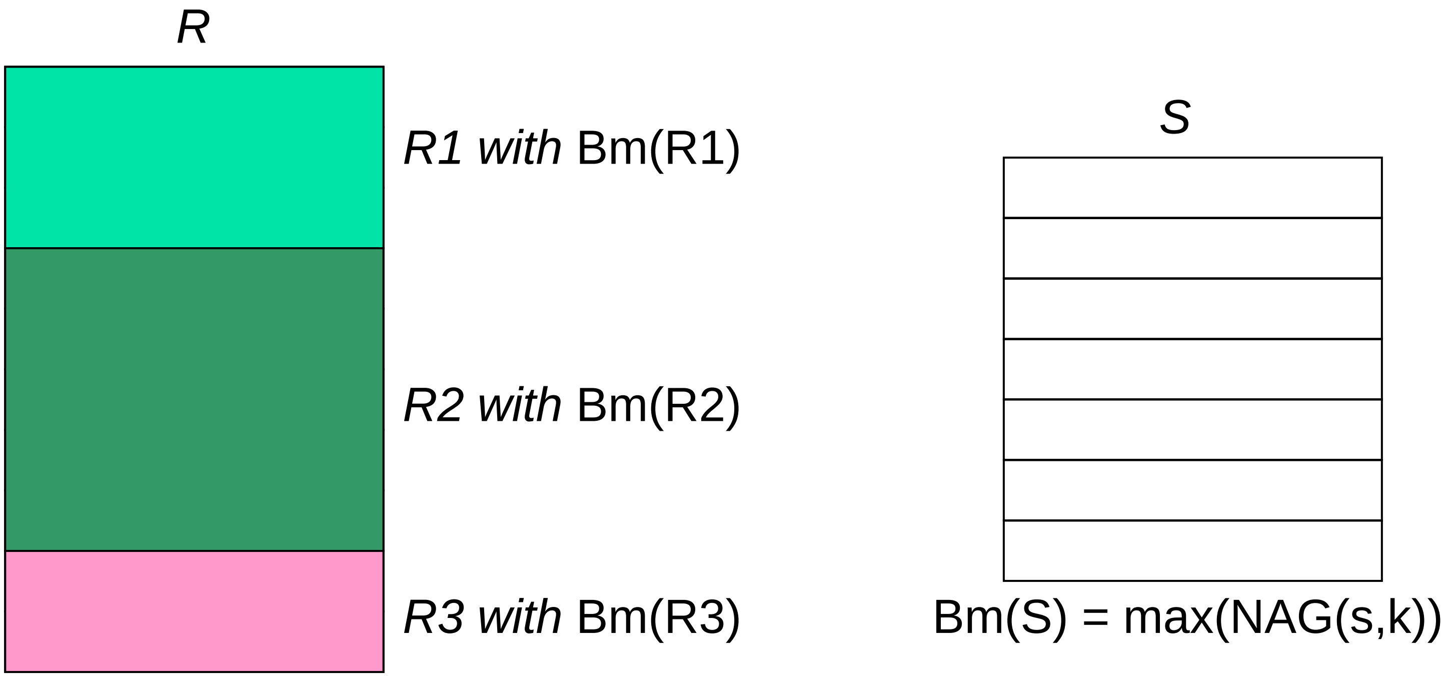


Bm(R) = max(NAG(r,k))

Bm(S) = max(NAG(s,k))

- Both are using the same gram dictionary.
- Use Bm(R) + Bm(S) as the new hamming bound.

PartEnum + VGRAM (optimization)



R

R1 with Bm(R1)

R2 with Bm(R2)

R3 with Bm(R3)

S

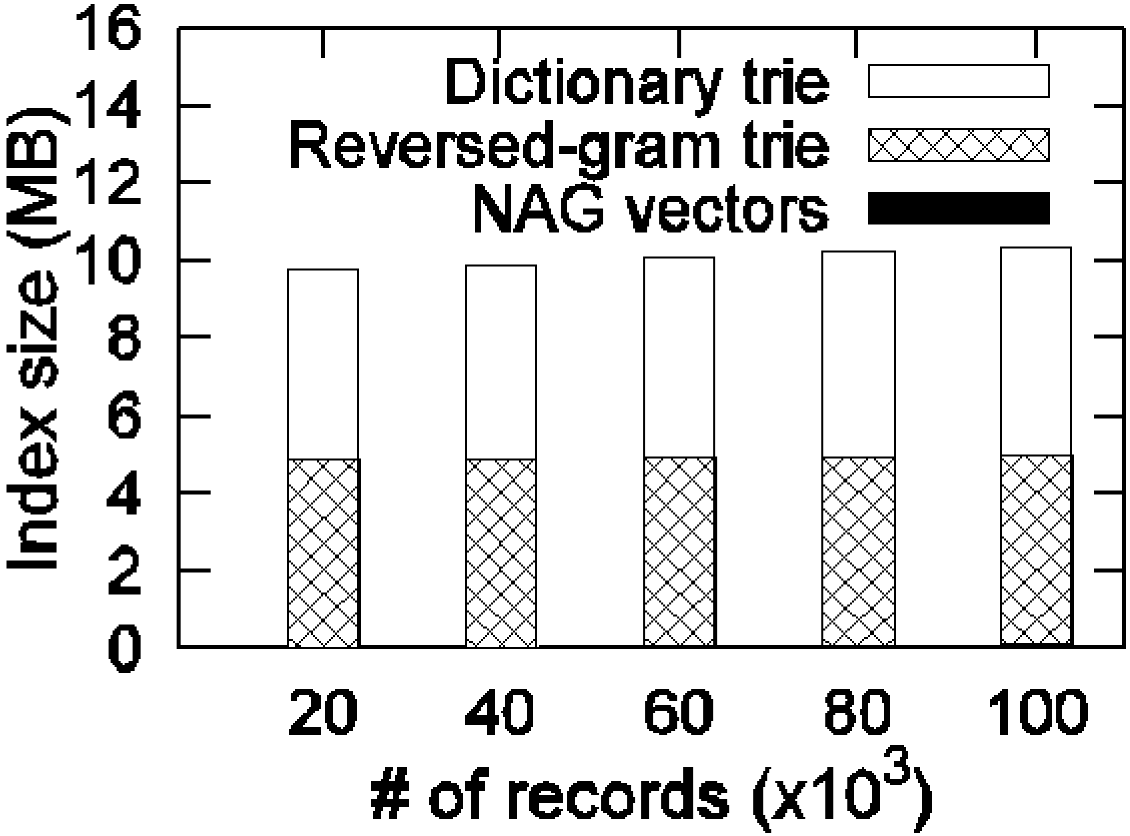
Bm(S) = max(NAG(s,k))

- Group R based on the NAG(r,k) values
- Join(R1,S) using Bm(R1) + Bm(S)
- Similarly, Join(R2,S), Join(R3,S)
- Local bounds tighter → better signatures generated
- Grouping *S* also possible.

Data sets

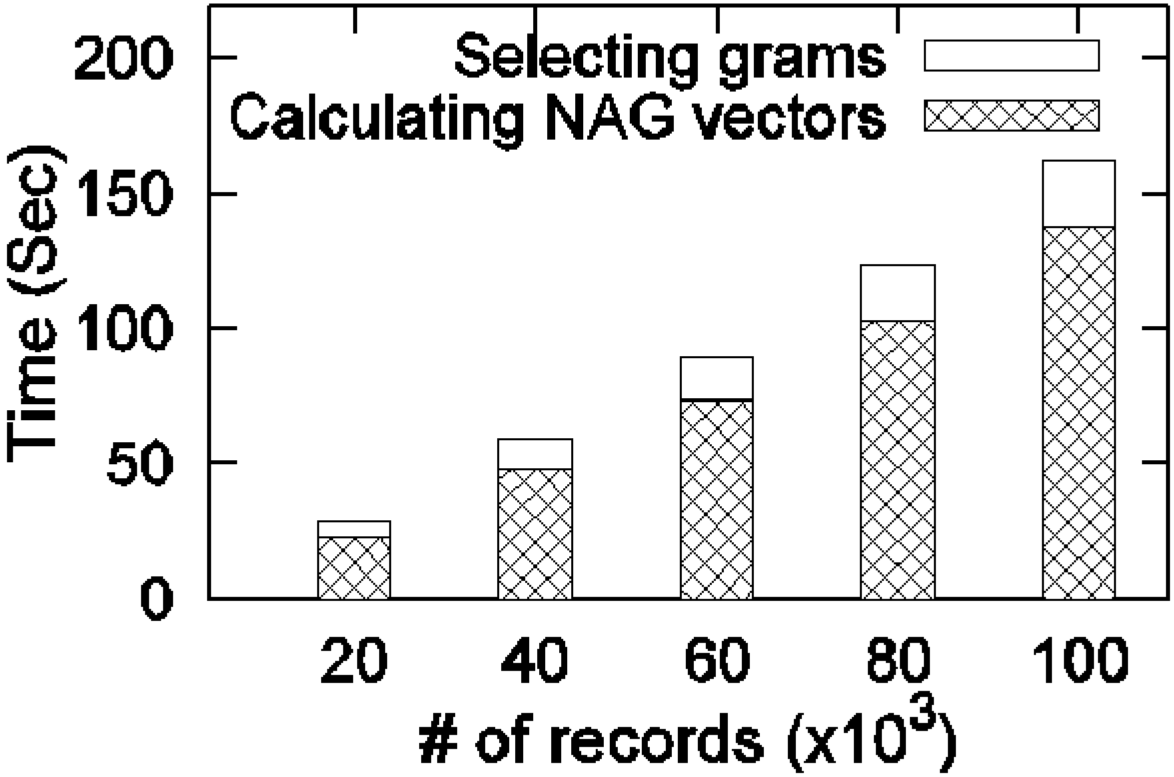
- *Data set 1*: Texas Real Estate Commission.
  - 151*K* person names, average length = 33.
- *Data set 2*: English dictionary from the Aspell spellchecker for Cygwin.
  - 149,165 words, average length = 8.
- *Data set 3*: DBLP Bibliography.
  - 277*K* titles, average length = 62.

VGRAM overhead (index size)



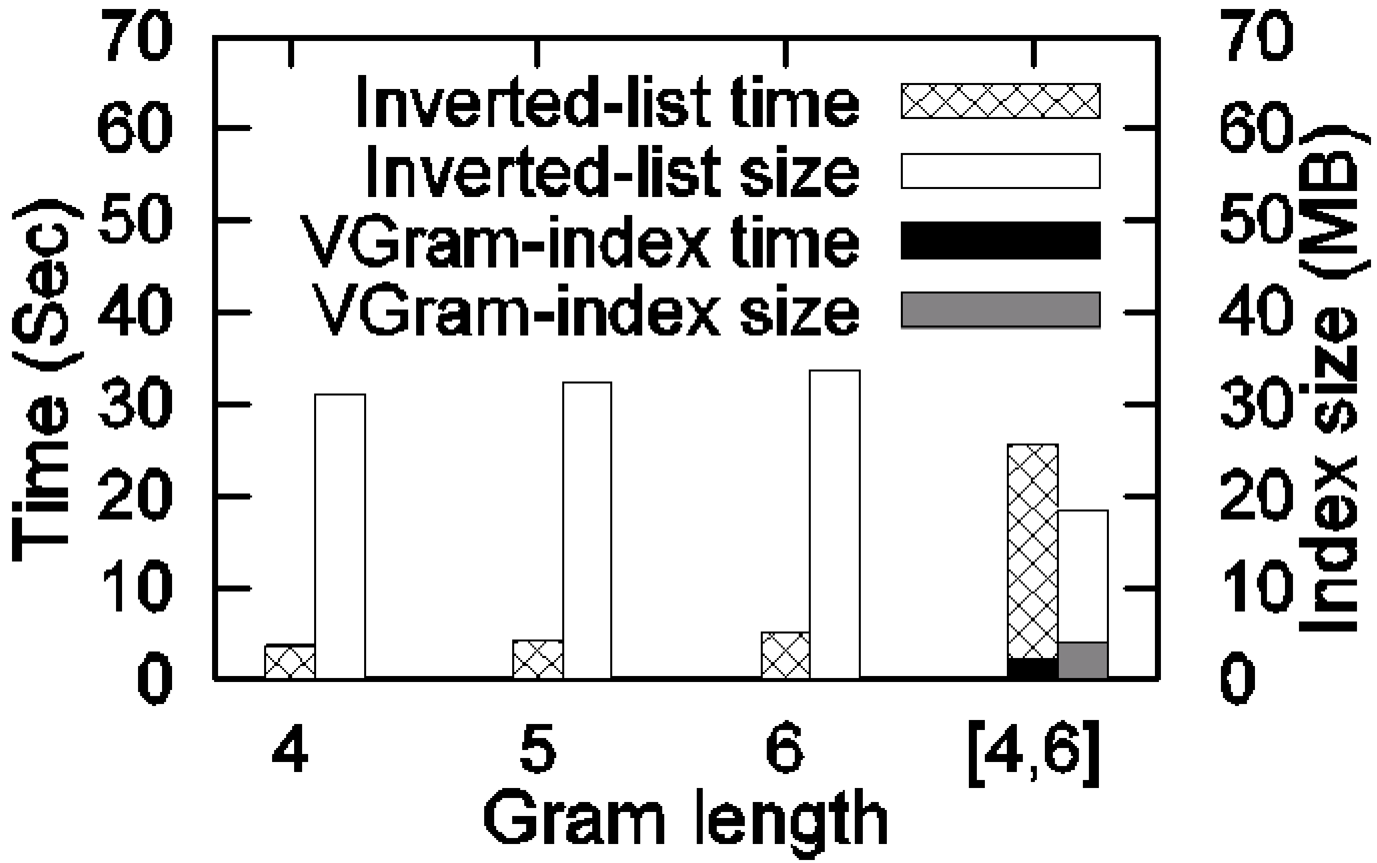
Dataset 3: DBLP titles

VGRAM overhead (construction time)



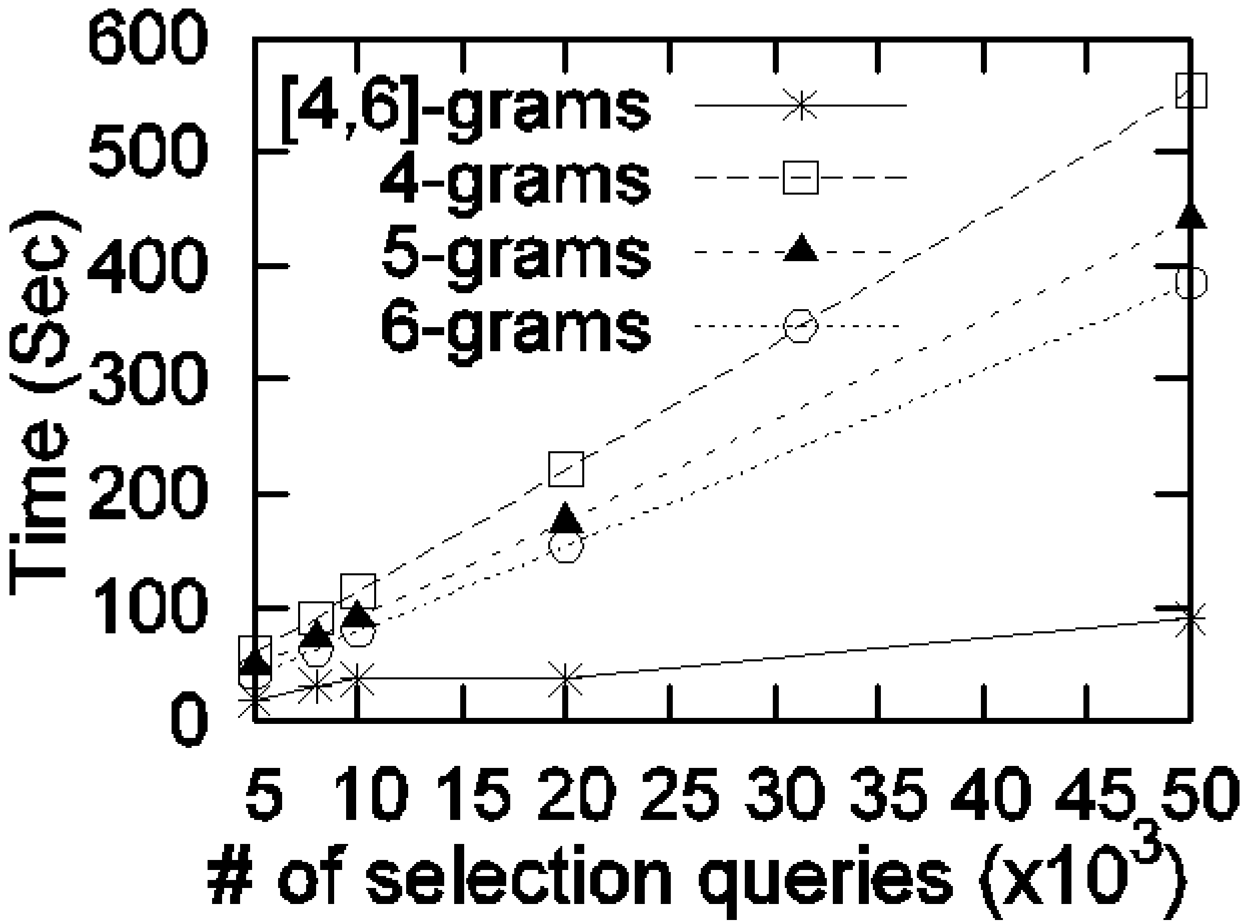
Dataset 3: DBLP titles

Benefits over fixed-length grams (index)



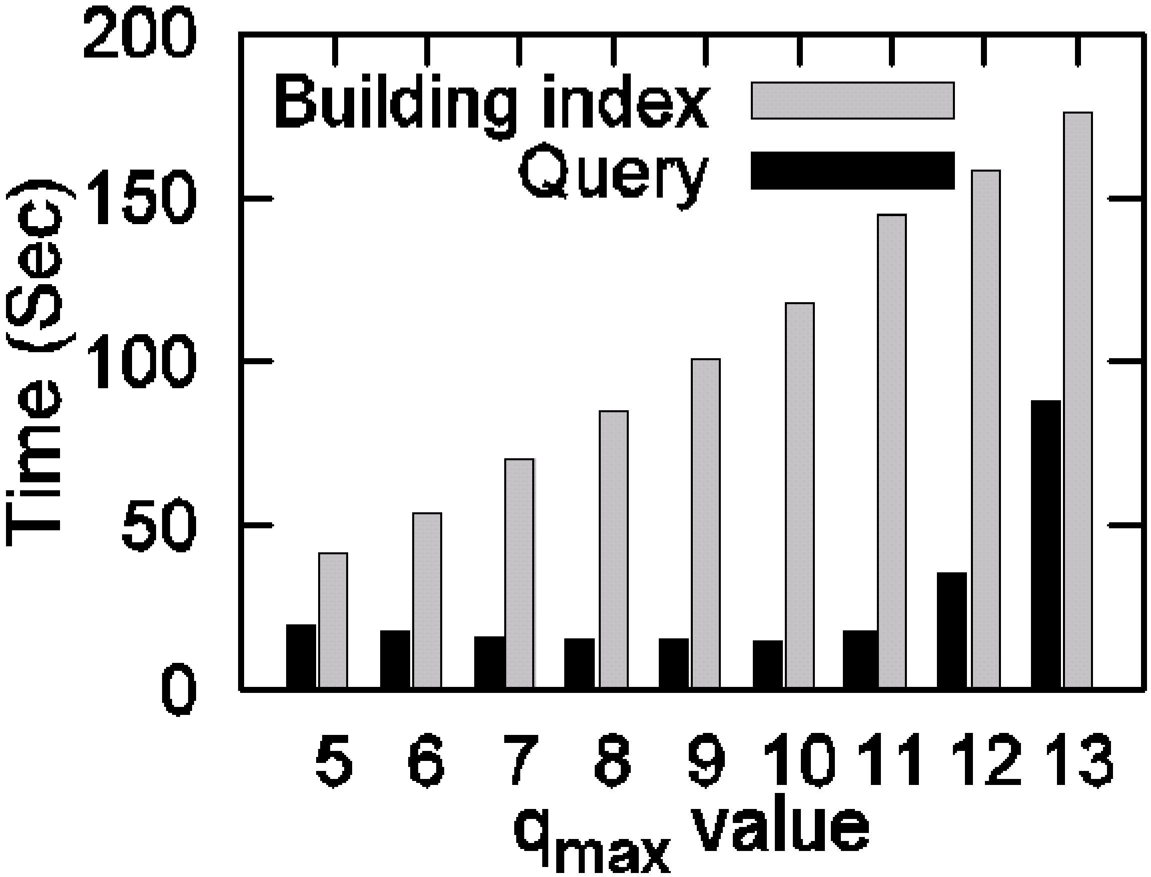
Dataset 1: Person names

Benefits over fixed-length grams (running time)



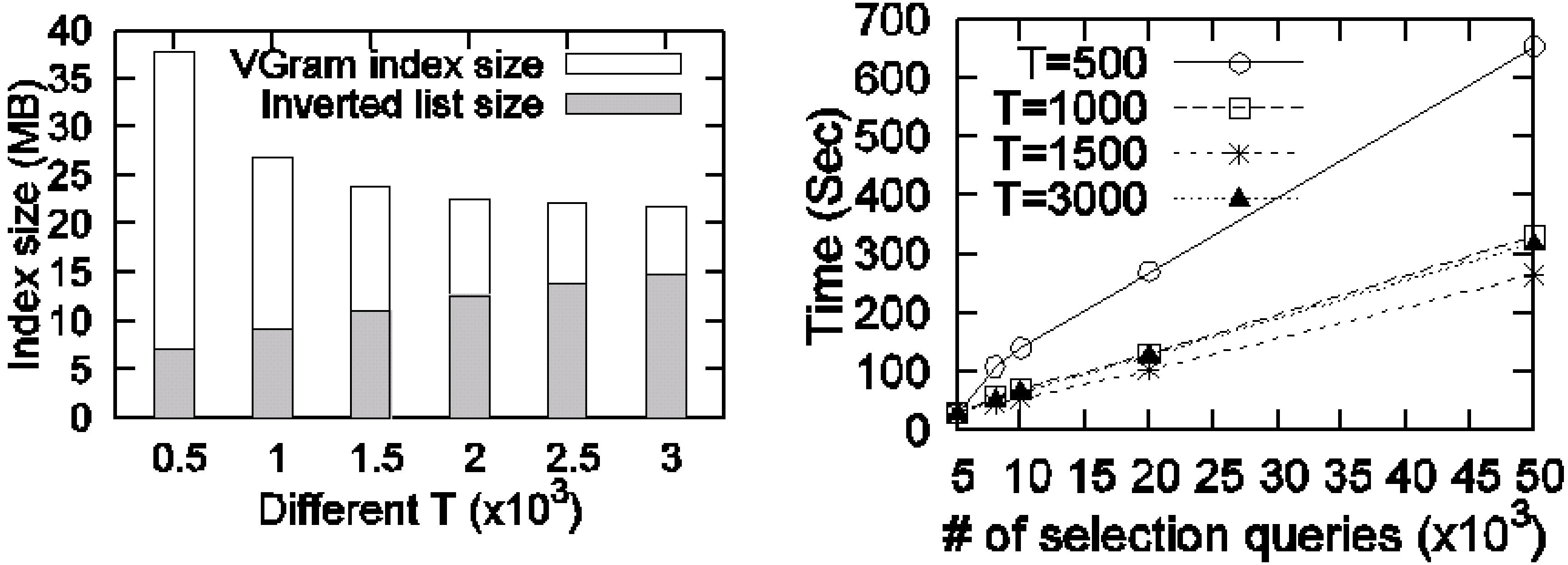
Dataset 1: Person names

Effect of qmax



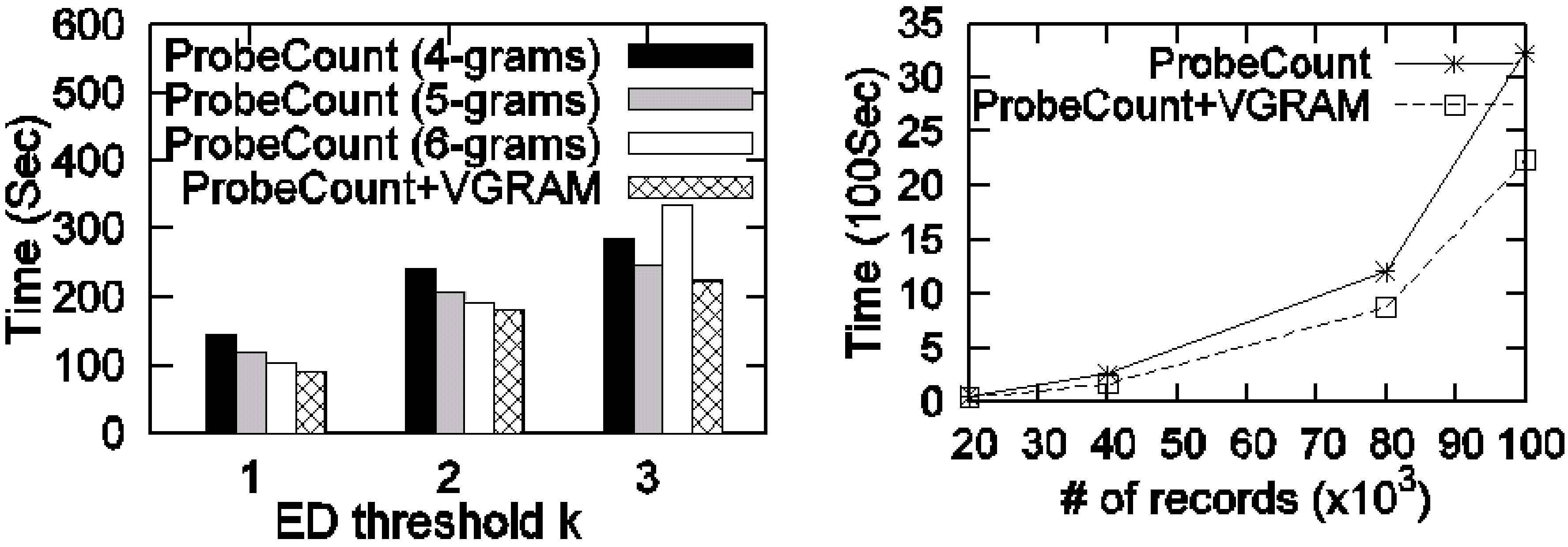
Dataset 1: Person names

Effect of frequency threshold T



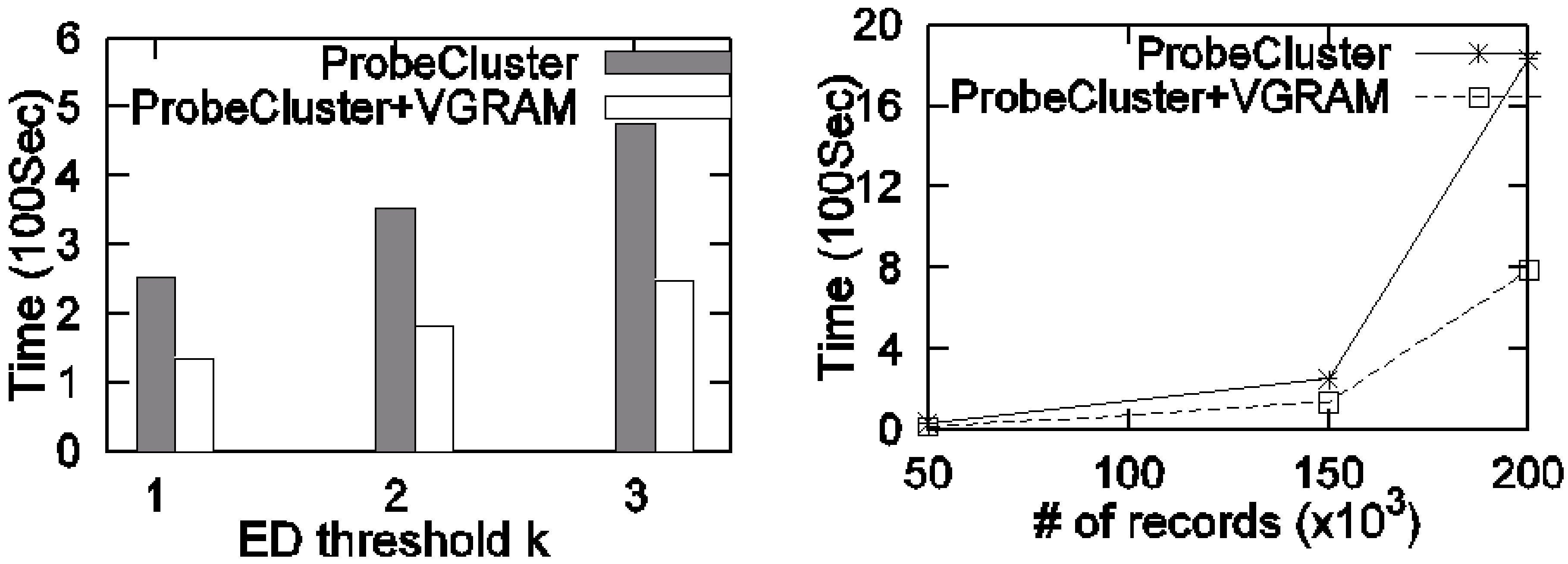
Dataset 1: Person name

Improving algorithm ProbeCount



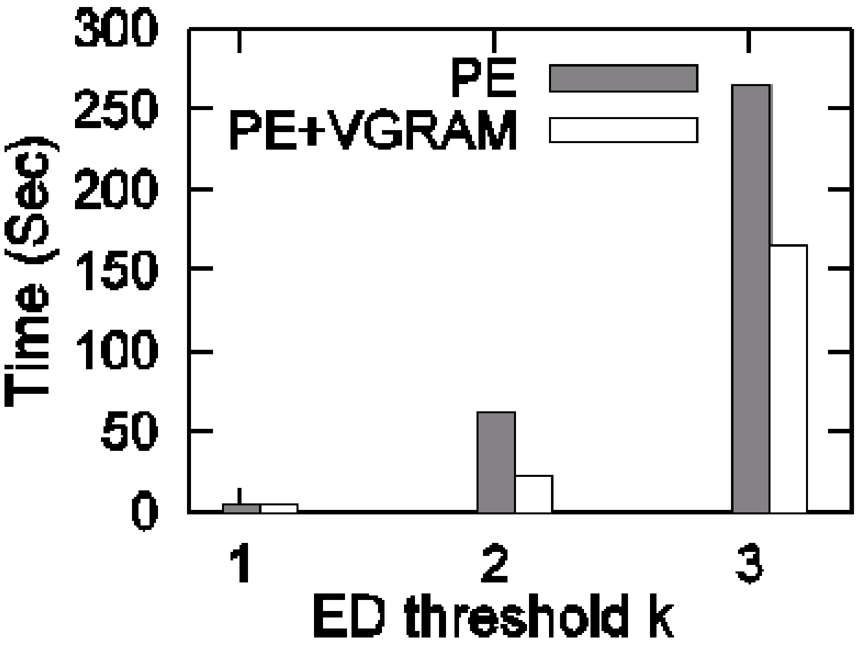
Dataset 1: Person name

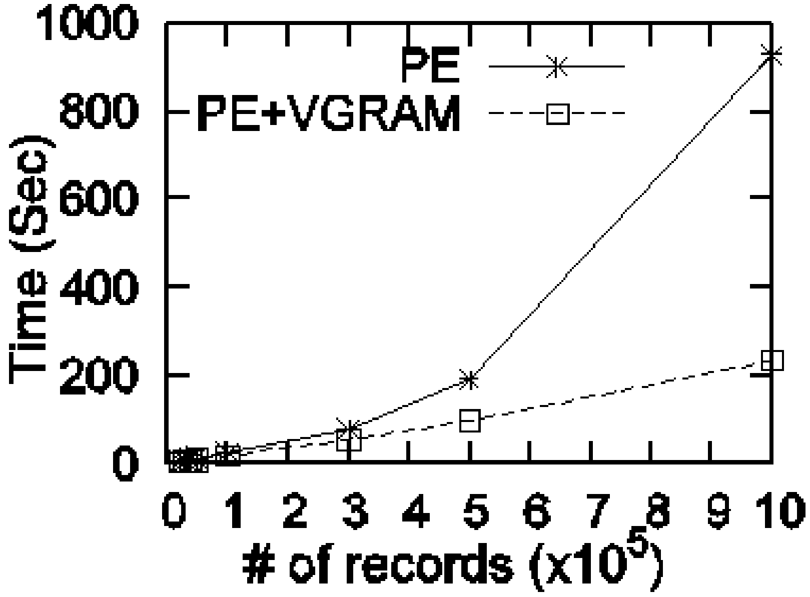
Improving algorithm ProbeCluster



Dataset 1: Person name

Improving algorithm PartEnum





Dataset 1: Person name

Discussions

- Dynamic maintenance
- Edit distance variants
  - Approximate substring queries
  - Block moves
- Using VGRAM in DBMS

Outline

- Types of sequences
- Foundation
  - Full matching: Building a disk based suffix tree
  - Approximate matching Using vgrams
- Technique & Application
  - Finding global partial order in sequence
  - Finding motif in sequence

Motivation

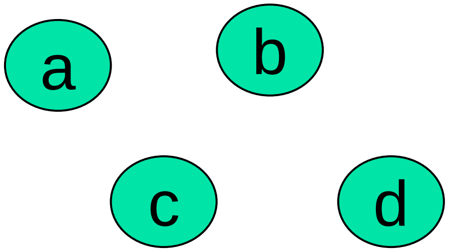
- Can we describe the data using model?
- More specifically, can we describe the sequence data by few representative sequence?
- Existing work like Hidden Markov Model can provide useful information but not an understandable global view

Example

- Sequence data
  - “a b c d” – 100 times
  - “a c b d” – 100 times
- Note : sequence data is ordered data, such as web page traversal
- Representative of the above sequence data
  - “a => c => d” and “a => b => d”

Representative type

- Trivial pattern =>
  - Lost ordering information
  - Too generic



- Specific patter => a => b => c => d
  - Too specific
  - Might represent only a part of data
- Partial Order => a => {c,b} => d
  - A combined approach
  - Partially represent the data by two order

Partial order

- To understand the partial order, an informal definition is,

p is a partial order of s,

if p is substring of s

partial order p1 is compatible with partial order p2,

if p2 is substring of p1

Problem Definition

- Find one or more partial order M,

  Which describes many sequences as well as maintained ordering information

Or

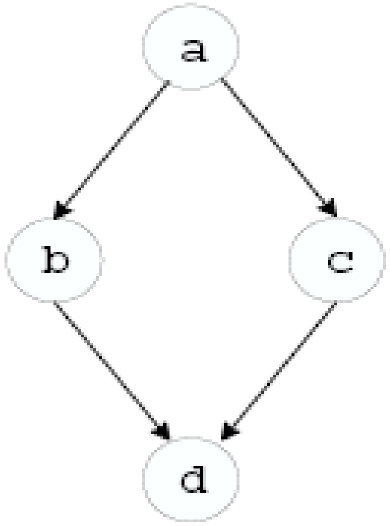
- Find M, so the probability of generating all sequences from S is maximum

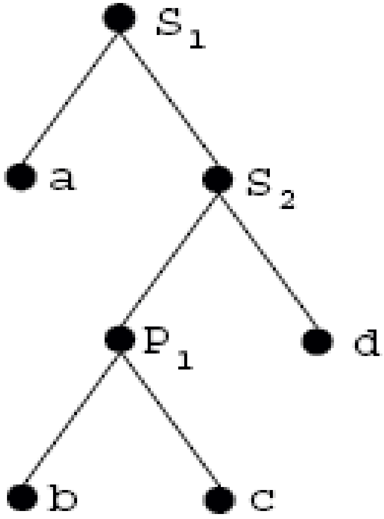
  max { P(S|M) }

Problem solution

- Assumption
  - Same event will not be repeated in the single sequence
    - “ a b c a ” will be considered as “ a b c ”
  - Partial order must be in form of series parallel tree
    - Explained later
- $\alpha(M)$ = a set of all complete extension of M

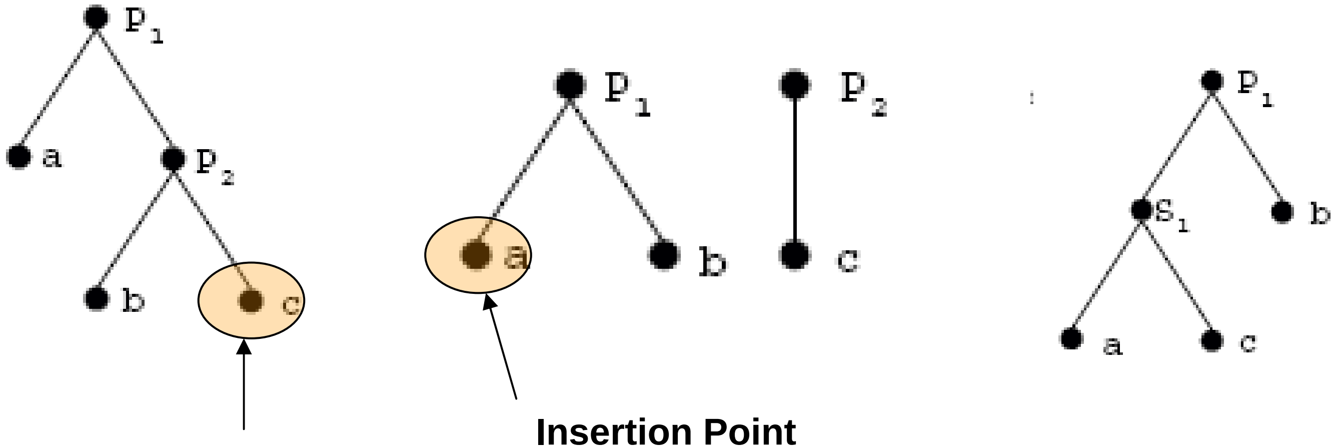
Series Parallel tree





$$
\begin{aligned}
\alpha(v) &= 1 \\
\alpha(S(M_1, M_2)) &= \alpha(M_1) \times \alpha(M_2); \\
\alpha(P(M_1, M_2)) &= \frac{(n(M_1) + n(M_2))!}{n(M_1)!n(M_2)!} \times \alpha(M_1) \times \alpha(M_2);
\end{aligned}
$$

Generating various order



Insertion Point

Deletion Point

Mixture model

- Single partial order can not generate the all sequences
- We need a mixture of various partial order
- Mixture model – weighed combination of the various partial order

Example

Sequence Data :-

“ a b c d ” – 100 times

“ a c b d ” – 100 times

“ d b c a ” – 5 times

Mixture Model =

{ d → b → c → a with weight 0.025,

a → {b,c} → d with weight 0.975 }

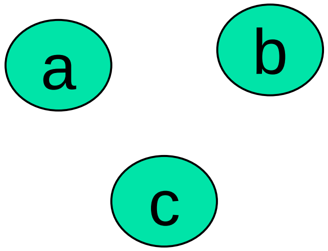
Algorithm

- Step 1: Start from trivial partial order
- Step2 : Apply the operation to current best model and try to increase likelihood of the partial order
- Repeat step 2, until no improvement

Example

- Given sequences
  - abc – 100 times
  - bac – 100 times

- Step 1: Start from trivial order



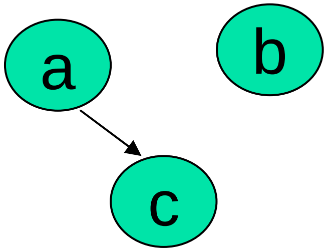
Continue...

= Max { Pro(a → b),**Pro(a → c),Pro(b → c)**,Pro(b → a)}

Select either one : **Pro(a → c),Pro(b → c)**,

Say “a → c” is selected =>

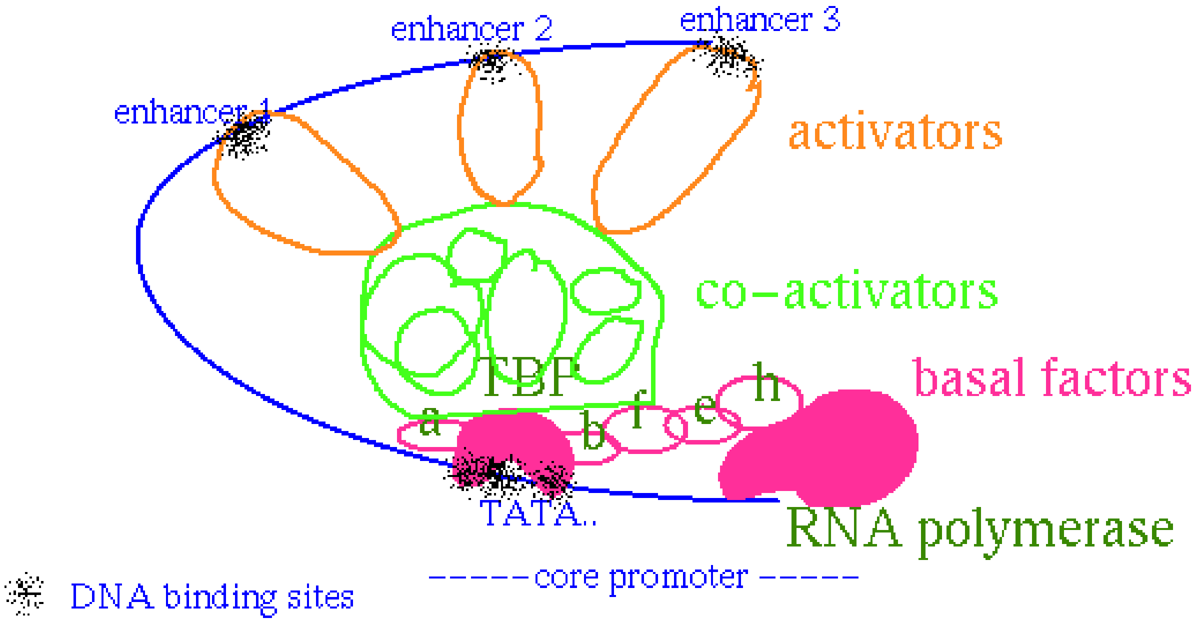
Step 2: Iterate the same...



Outline

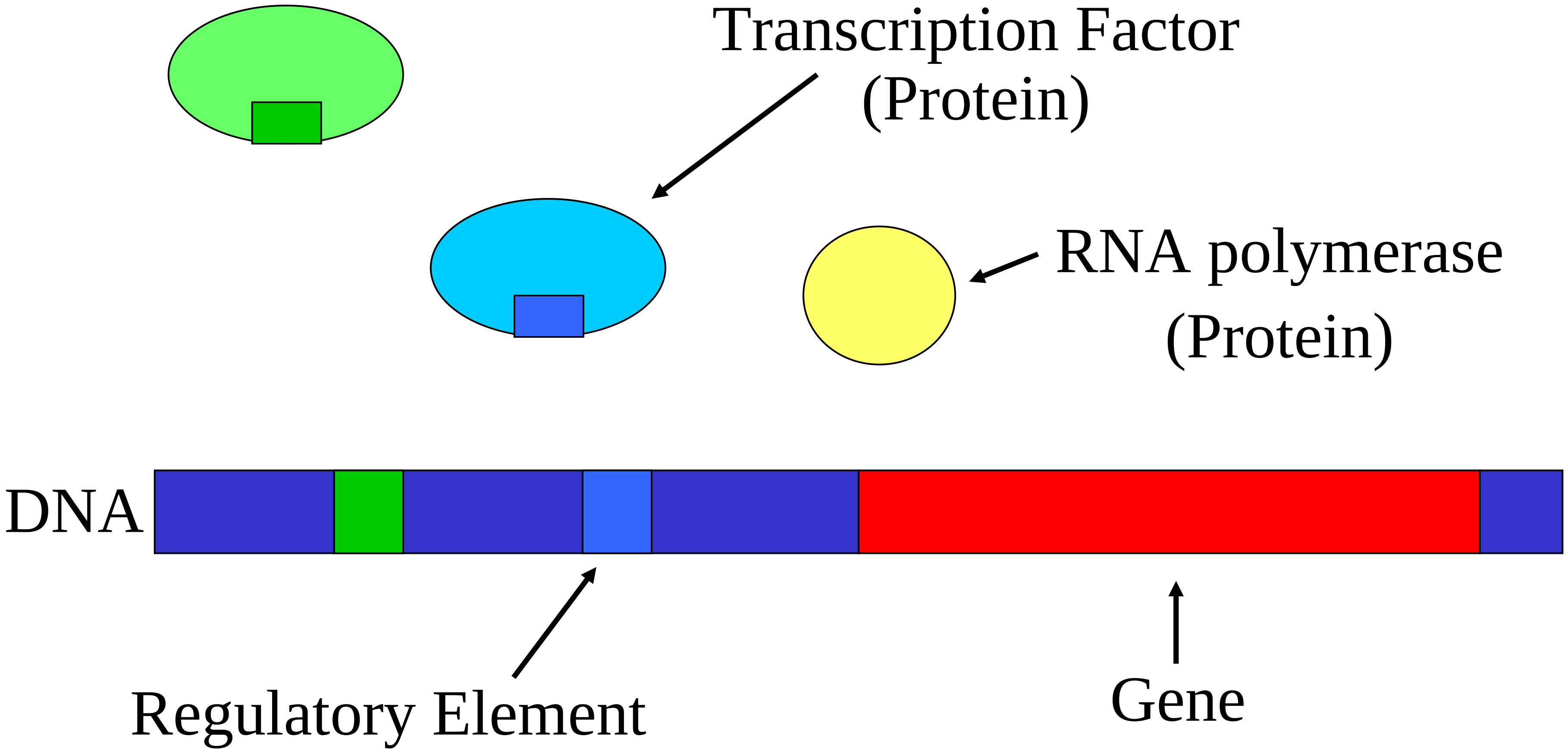
- Types of sequences
- Foundation
  - Full matching: Building a disk based suffix tree
  - Approximate matching Using vgrams
- Technique & Application
  - Finding global partial order in sequence
  - Finding motif in sequence

Promoter and Enhancers

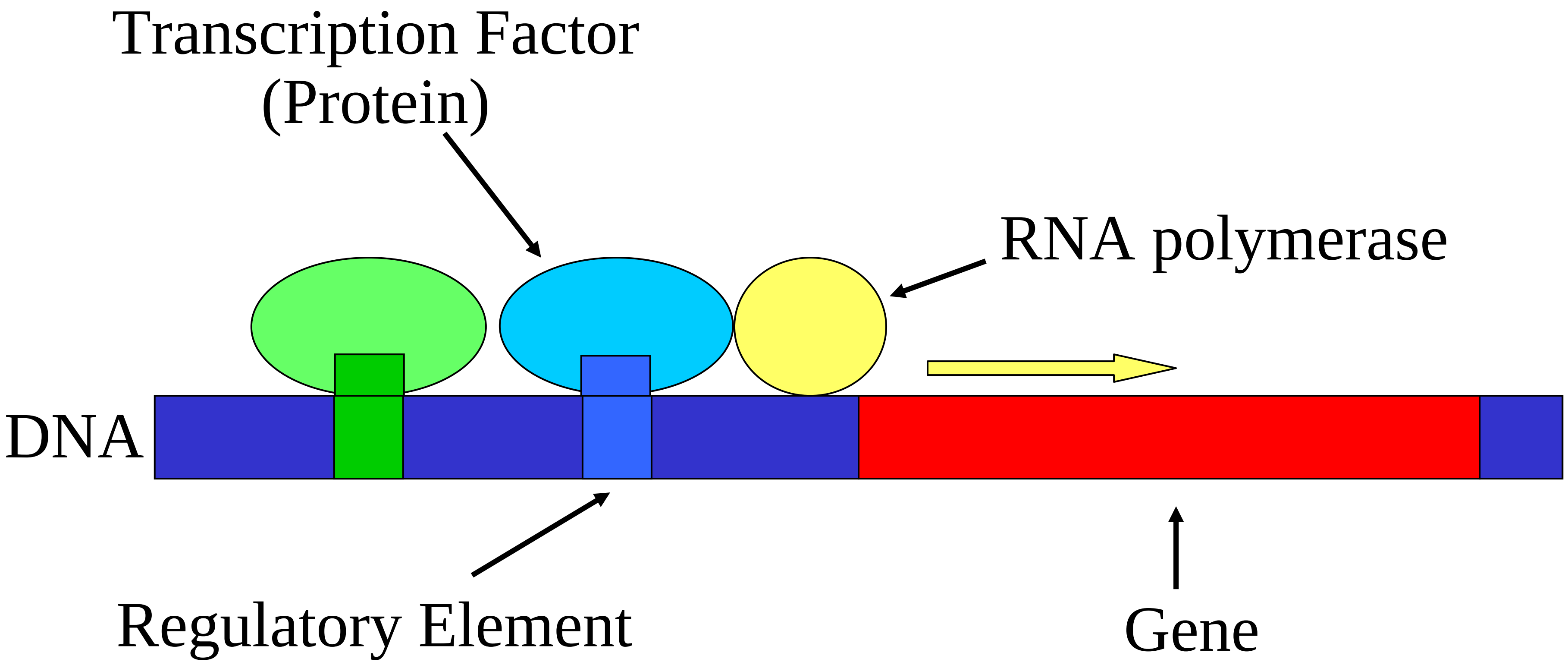


- Promoter necessary to start transcription
- Enhancers can affect transcription from afar

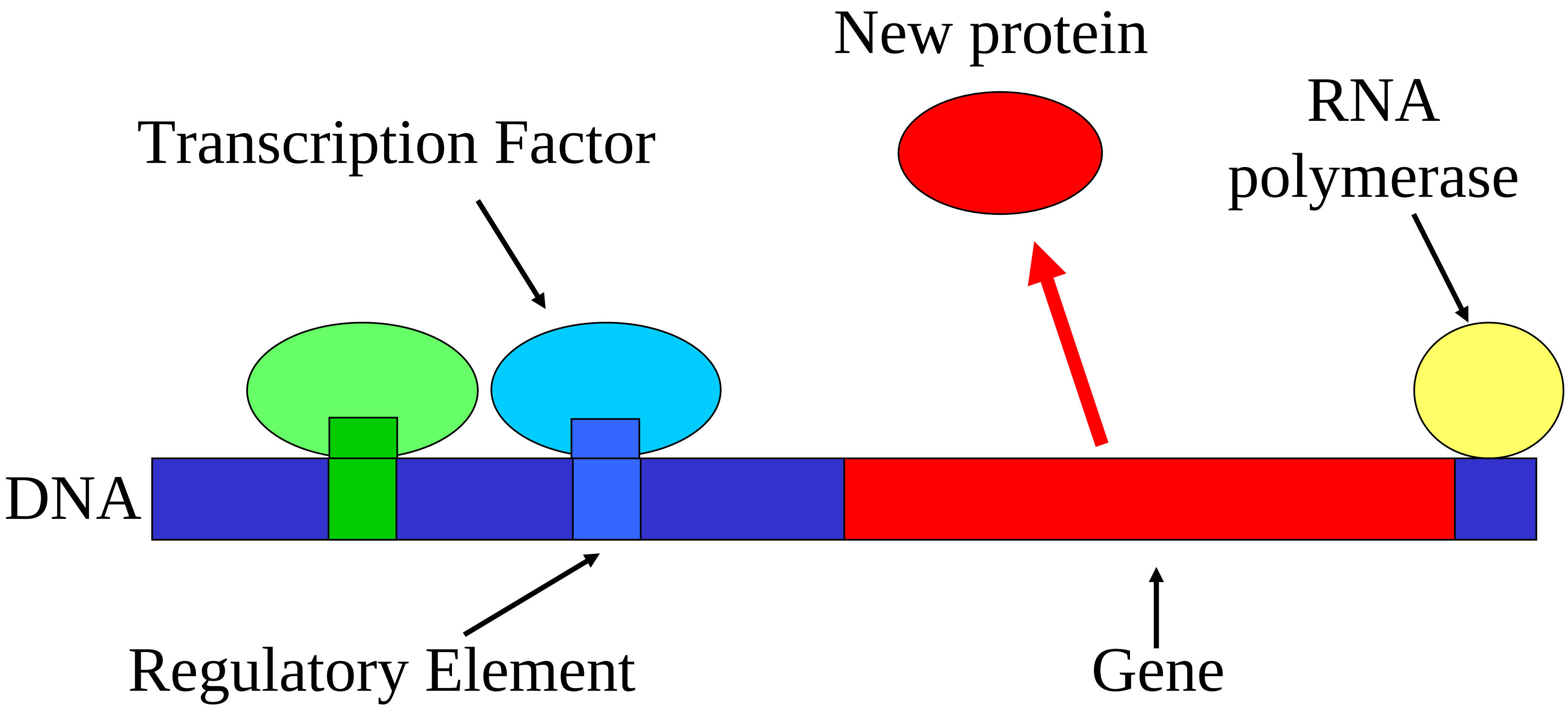
Regulation of Genes



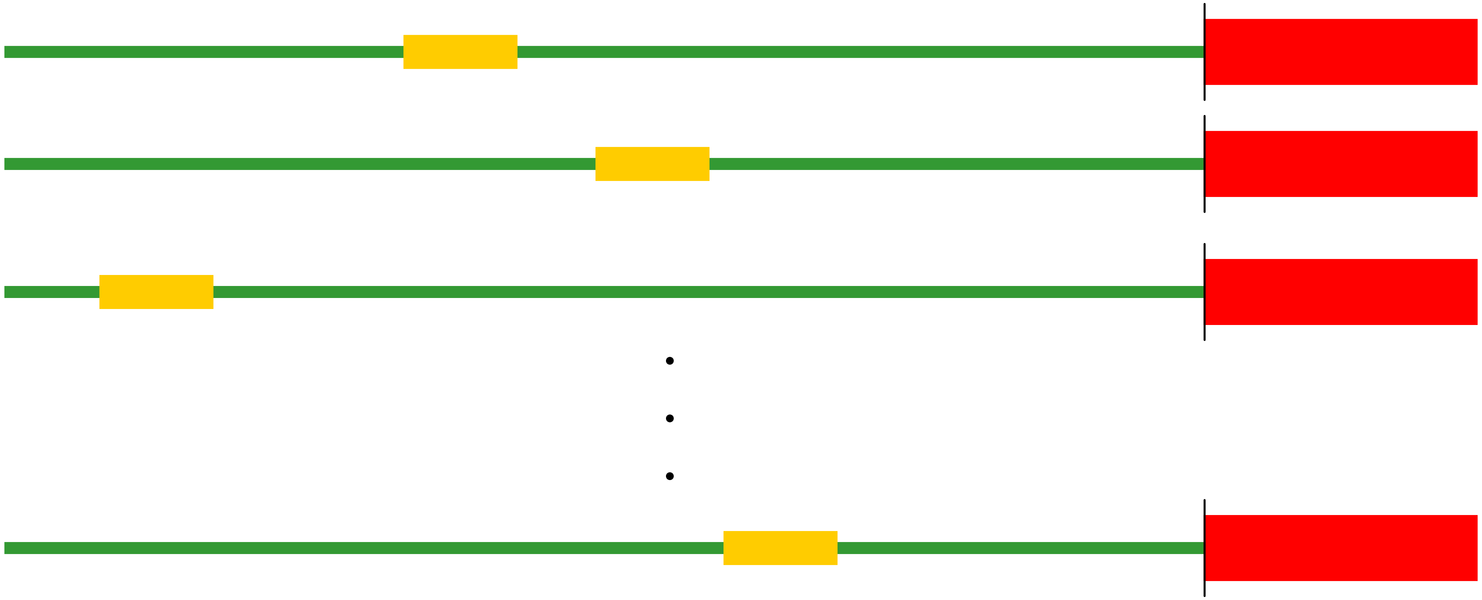
Regulation of Genes



Regulation of Genes



Finding Regulatory Motifs



Given a collection of genes with common expression,

Find the TF-binding motif in common

Problem Definition

Given a collection of promoter sequences $s_1, \ldots, s_N$ of genes with common expression

Probabilistic

Motif: M_{ij}; $1 \le i \le W$

$1 \le j \le 4$

M_{ij} = Prob[letter j, pos i]

Find best M, and positions $p_1, \ldots, p_N$ in sequences

Combinatorial

Motif M: $m_1 \ldots m_W$

Some of the m_i's blank

Find M that occurs in all s_i with $\le$ k differences

Algorithms

- Probabilistic
  1. Expectation Maximization:

     MEME
  2. Gibbs Sampling:

     AlignACE, BioProspector
- Combinatorial

  CONSENSUS, TEIRESIAS, SP-STAR, others

Expectation Maximization

The MM algorithm, part of MEME package uses Expectation Maximization

Algorithm (sketch):

1. Given genomic sequences find all K-long words
2. Assume each word is motif or background
3. Find likeliest
   - Motif Model
   - Background Model
   - Classification of words into either Motif or Background

Expectation Maximization

- Given sequences $x^1, \ldots, x^N$,
- Find all k-long words $X_1, \ldots, X_n$
- Define motif model:

  $M = (M_1, \ldots, M_K)$

  $M_i = (M_{i1}, \ldots, M_{i4})$ (assume {A, C, G, T})

  where M_{ij} = Prob[letter j occurs in motif position i]
- Define background model:

  $B = B_1, \ldots, B_4$

  B_i = Prob[letter j in background sequence]

Expectation Maximization

- Define

  Z_{i1} = { 1, if X_i is motif;

  0, otherwise }

  Z_{i2} = { 0, if X_i is motif;

  1, otherwise }

- Given a word $X_i = x[1]...x[k]_{,}$

  $P[\ X_i, Z_{i1}=1\ ] = \lambda\ M_{1x[1]}...M_{kx[k]}$

  $P[\ X_i, Z_{i2}=1\ ] = (1 - \lambda)\ B_{x[1]}...B_{x[K]}$

  Let $\lambda_1 = \lambda;\ \lambda_2 = (1- \lambda)$

Expectation Maximization

Define:

Parameter space θ = (M,B)

θ_1: Motif; θ_2: Background

Objective:

Maximize log likelihood of model:

$$\log P(X_1 \ldots X_n, Z \mid \theta, \lambda) = \sum_{i=1}^{n} \sum_{j=1}^{2} Z_{ij} \log(\lambda_j P(X_i \mid \theta_j))$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{2} Z_{ij} \log P(X_i \mid \theta_j) + \sum_{i=1}^{n} \sum_{j=1}^{2} Z_{ij} \log \lambda_j$$

Expectation Maximization

- Maximize expected likelihood, in iteration of two steps:

Expectation:

Find expected value of log likelihood:

$$E[\log P(X_1 \ldots X_n, Z \mid \theta, \lambda)]$$

Maximization:

Maximize expected value over θ, λ

Expectation Maximization: E-step

Expectation:

Find expected value of log likelihood:

$$E[\log P(X_1 \ldots X_n, Z \mid \theta, \lambda)] =$$

$$\sum_{i=1}^{n} \sum_{j=1}^{2} E[Z_{ij}] \log P(X_i \mid \theta_j) + \sum_{i=1}^{n} \sum_{j=1}^{2} E[Z_{ij}] \log \lambda_j$$

where expected values of Z can be computed as follows:

$$E[Z_{ij}] = \frac{\lambda_j P(X_i \mid \theta_j)}{\sum_{k=1}^{2} \lambda_k P(X_i \mid \theta_k)}$$

Expectation Maximization: M-step

Maximization:

Maximize expected value over θ and λ independently

For λ, this is easy:

$$\lambda_j^{NEW} = \arg\max_{\lambda_j} \sum_{i=1}^{n} E[Z_{ij}] \log \lambda_j = \sum_{i=1}^{n} \frac{Z_{ij}}{n}$$

Expectation Maximization: M-step

- For θ = (M, B), define

  c_{jk} = E[# times letter k appears in motif position j]

  c_{0k} = E[# times letter k appears in background]

  - c_{ij} values are calculated easily from E[Z] values

  It easily follows:

$$M_{jk}^{NEW} = \frac{c_{jk}}{\sum_{k=1}^{4} c_{jk}} \qquad B_k^{NEW} = \frac{c_{0k}}{\sum_{k=1}^{4} c_{0k}}$$

to not allow any 0's, add pseudocounts

Initial Parameters Matter!

Consider the following “artificial” example:

$x^1, \ldots, x^N$ contain:

- 2^{12} patterns on {A, T}: A...A, A...AT,......, T... T
- 2^{12} patterns on {C, G}: C...C , C...CG,...... , G...G
- D << 2^{12} occurrences of 12-mer ACTGACTGACTG

Some local maxima:

λ ≈ ½; B = ½C, ½G; M_i = ½A, ½T, i = 1,..., 12

λ ≈ $D/2^{k+1}$; B = ¼A,¼C,¼G,¼T;
M_1 = 100% A, M_2= 100% C, M_3 = 100% T, etc.

Overview of EM Algorithm

1. Initialize parameters $\theta = (M, B), \lambda$:
   - Try different values of λ from $N^{-1/2}$ up to $1/(2K)$
2. Repeat:
   a. Expectation
   b. Maximization
3. Until change in $\theta = (M, B), \lambda$ falls below ε
4. Report results for several “good” λ

Overview of EM Algorithm

- One iteration running time: O(NK)
  - Usually need < N iterations for convergence, and < N starting points.
  - Overall complexity: unclear – typically $O(N^2K)$ - $O(N^3K)$
- EM is a local optimization method
- Initial parameters matter

MEME: Bailey and Elkan, ISMB 1994.

Gibbs Sampling

- Given:
  - $x^1, \ldots, x^N$,
  - motif length K,
  - background B,
- Find:
  - Model M
  - Locations $a_1, \ldots, a_N$ in $x^1, \ldots, x^N$

Maximizing log-odds likelihood ratio:

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \log \frac{M(k, x^i_{a_i+k})}{B(x^i_{a_i+k})}$$

Gibbs Sampling

- AlignACE: first statistical motif finder
- BioProspector: improved version of AlignACE

Algorithm (sketch):

1. Initialization:
   a. Select random locations in sequences $x^1, \ldots, x^N$
   b. Compute an initial model M from these locations
2. Sampling Iterations:
   a. Remove one sequence x^i
   b. Recalculate model
   c. Pick a new location of motif in x^i according to probability the location is a motif occurrence

Gibbs Sampling

Running Gibbs Sampling:

1. Initialize
2. Run until convergence
3. Repeat 1,2 several times, report common motifs

Advantages / Disadvantages

- Very similar to EM

Advantages:

- Easier to implement
- Less dependent on initial parameters
- More versatile, easier to enhance with heuristics

Disadvantages:

- More dependent on all sequences to exhibit the motif
- Less systematic search of initial parameter space

References

- Benjarath Phoophakdee, Mohammed J. Zaki: "Genome-scale disk-based suffix tree indexing". SIGMOD Conference 2007: 833-844
- Chen Li, Bin Wang, and Xiaochun Yang . "VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams". In VLDB 2007.
- Heikki Mannila, Christopher Meek: Global partial orders from sequential data. KDD 2000: 161-168
- T.L. Bailey and Elkan C. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In Proc. Int. Conf. Intell. Syst. Mol. Biol., volume 2, pages 28--36. 1994

Optional References:

- U. Keich and P. Pevzner. Subtle motifs: defining the limits of motif finding algorithms. Bioinformatics, in press, 2002.
- B Ma, J Tromp, M Li. PatternHunter: faster and more sensitive homology search - Bioinformatics, 2002 - Oxford Univ Press
- Xia Cao, Shuai Cheng Li, Anthony K. H. Tung. "Indexing DNA Sequences Using q-grams". Best Paper Award. To appear in DASFAA 2005.

Acknowledgements

- Ken Sung
- Mohammad Zaki
- Chen Li