VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams

Chen Li

Bin Wang and Xiaochun Yang

UCIrvine
University of California, Irvine

Northeastern University, China

Approximate selection queries

Schwarrzenger

Keanu Reeves
Samuel Jackson
Schwarzenegger
Samuel Jackson
…

Query errors:

- Limited knowledge about data
- Typos
- Limited input device (cell phone) input

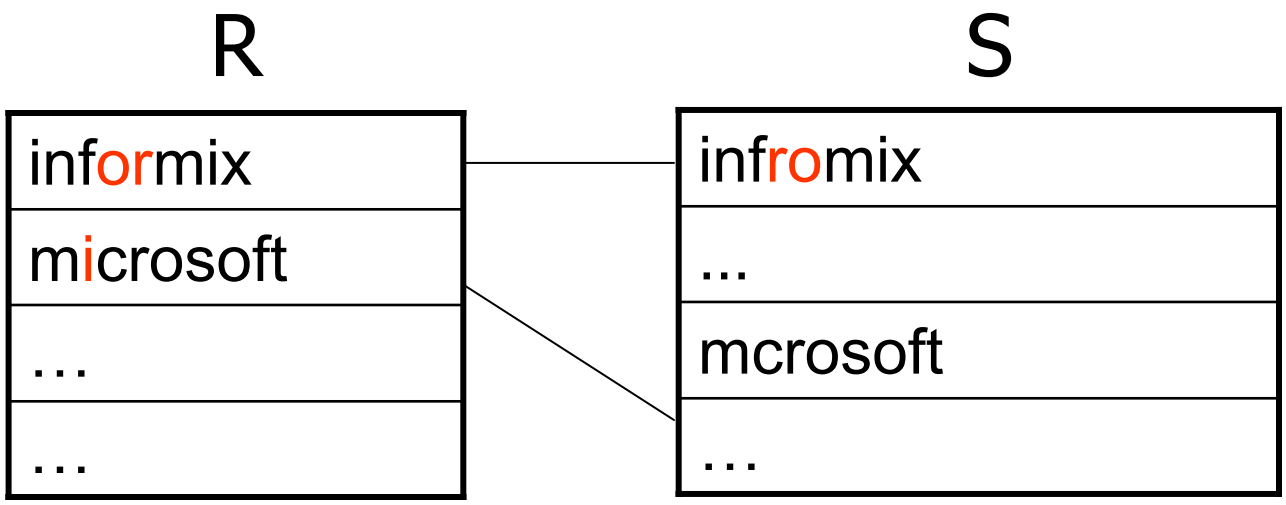
Data errors

- Typos
- Web data
- OCR

Applications

- Spellchecking
- Query relaxation
- …

Record linkage



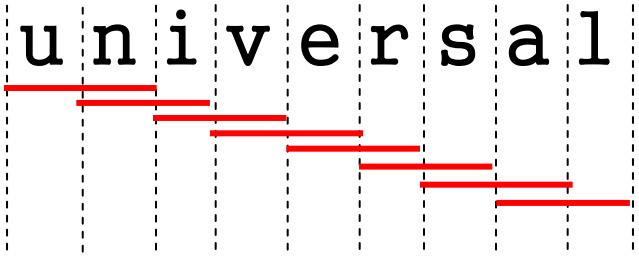
Applications

- Record linkage
- …

Similarity functions:

- Edit distance
- Jaccard
- Cosine
- …

“q-grams” of strings

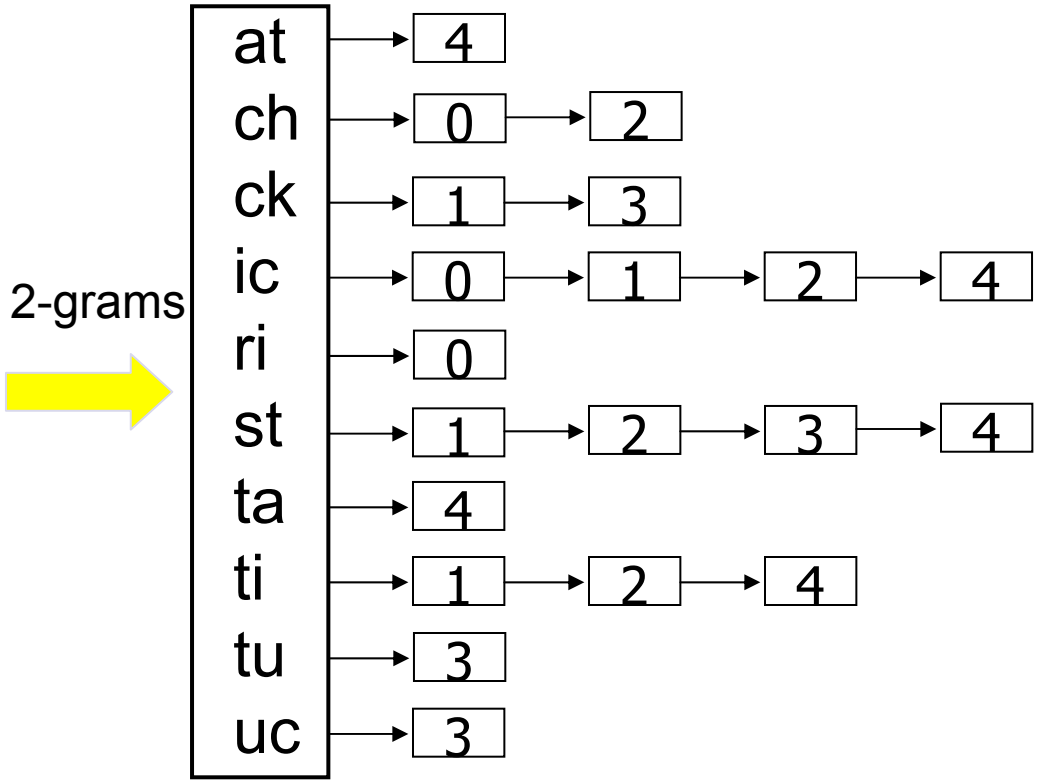


2-grams

q-gram inverted lists

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

2-grams



Searching using inverted lists

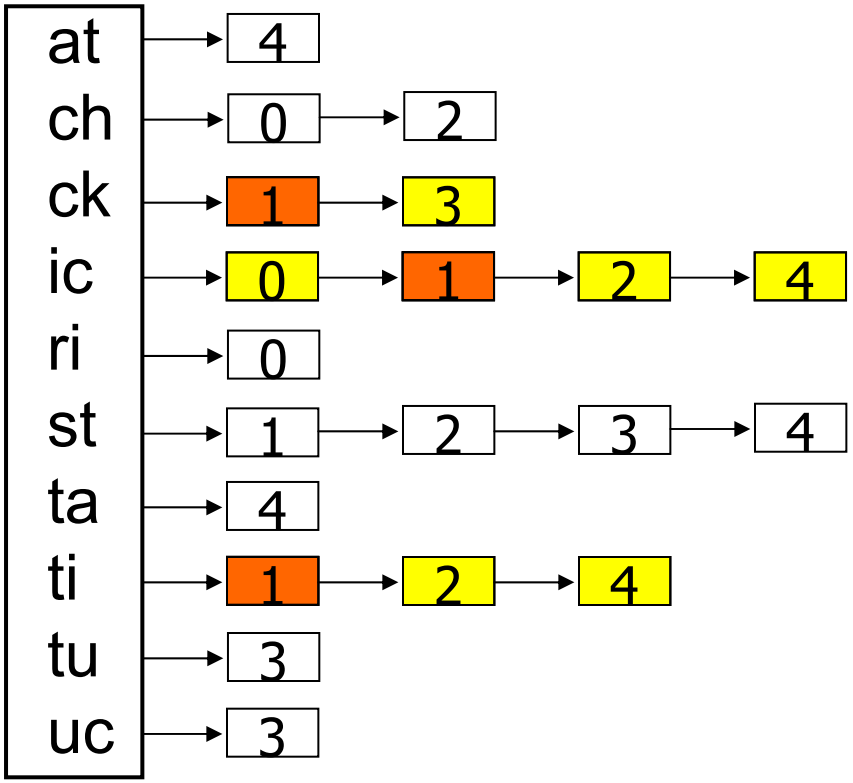
- Query: "shtick", ED(shtick, ?)⩽1

  sh ht ti ic ck

# of common grams >= 3

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

2-grams



2-grams → 3-grams?

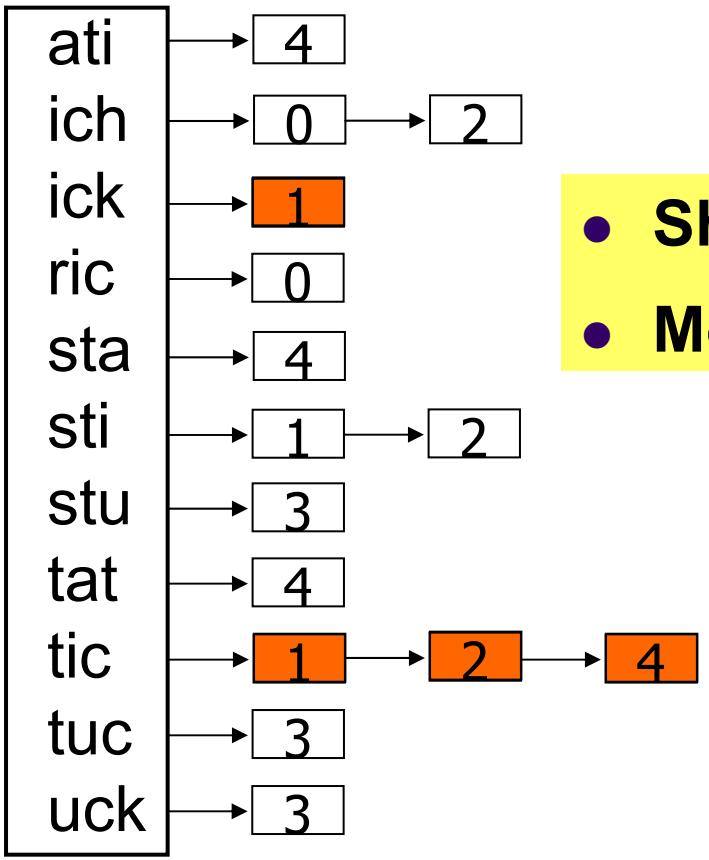
- Query: "shtick", ED(shtick, ?)⩽1

  sht hti tic ick

# of common grams >= 1

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

3-grams



- **Shorter inverted list**
- **More false positive**

Outline

- Motivation
- VGRAM
  - Main idea
  - Decomposing strings to grams
  - Choosing good grams
  - Effect of edit operations on grams
  - Adopting vgram in existing algorithms
- Experiments

Motivation

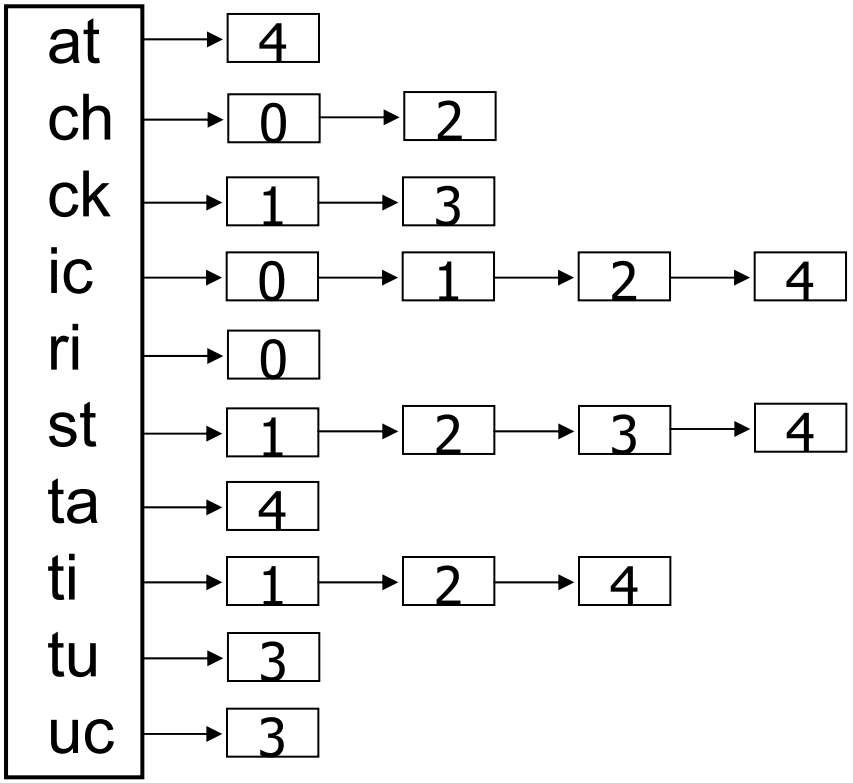
- Small index size (memory)
- Small running time
  - Merge matched inverted lists
  - Calculate ED(query, candidate)

Observation 1: dilemma of choosing “q”

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

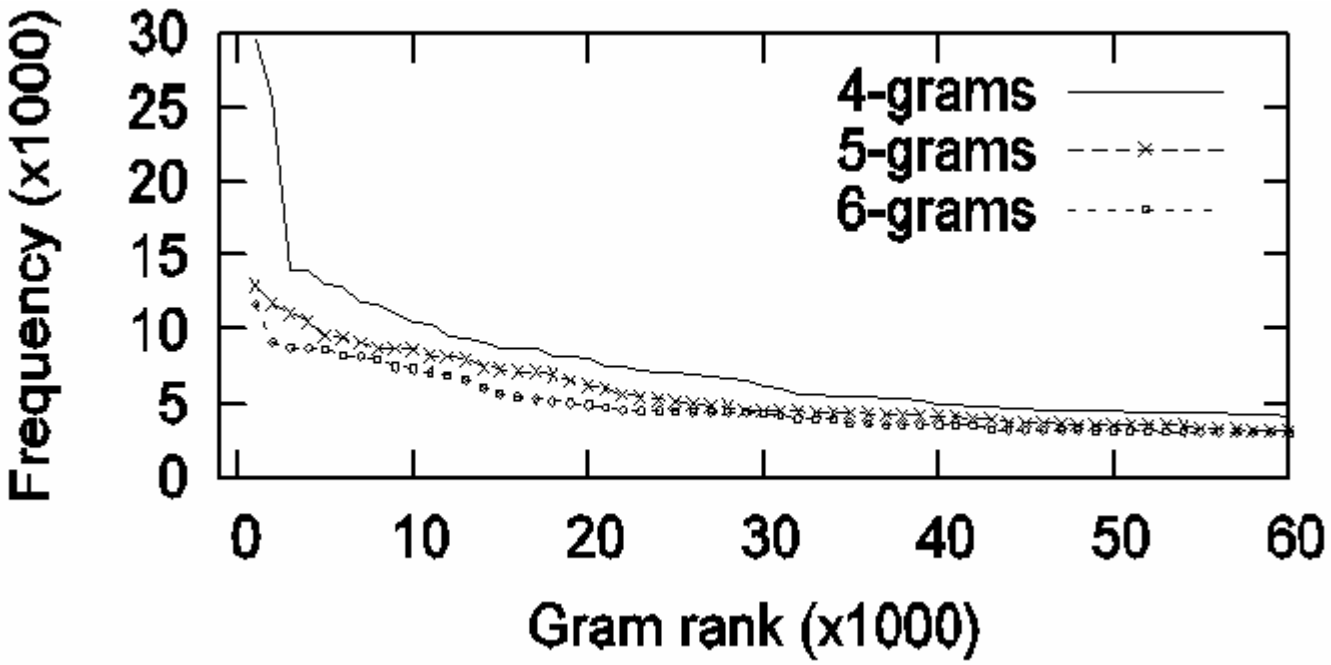
id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

2-grams



Observation 2: skew distributions of gram frequencies

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



VGRAM: Main idea

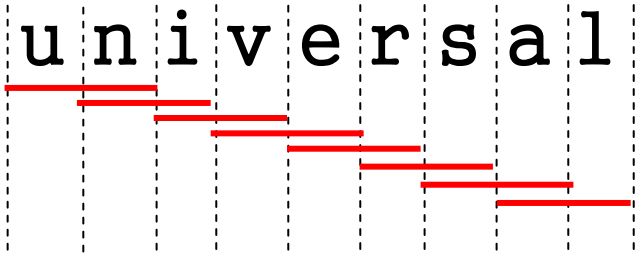
- Grams with variable lengths (between q_{min} and q_{max})
  - `zebra`
    - ze(123)
  - `corrasion`
    - co(5213), cor(859), corr(171)
- Advantages
  - Reducing index size ☺
  - Reducing running time ☺
  - Adoptable by many algorithms ☺

Challenges

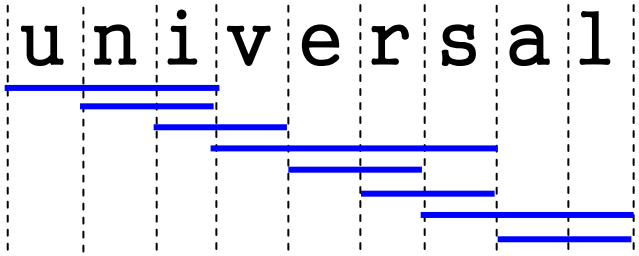
- Generating variable-length grams?
- Constructing a high-quality gram dictionary?
- Relationship between string similarity and their gram-set similarity?
- Adopting VGRAM in existing algorithms?

Challenge 1: String → Variable-length grams?

- Fixed-length 2-grams



- Variable-length grams

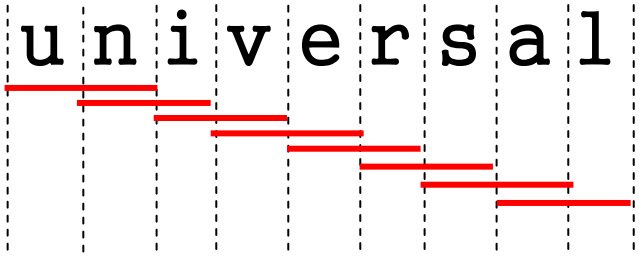


[2,4]-gram dictionary

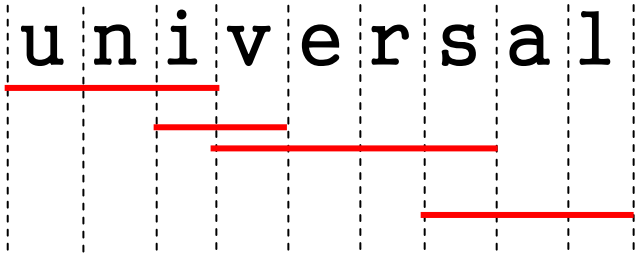
ni
ivr
sal
uni
vers

Representing gram dictionary as a trie

- Fixed-length 2-grams

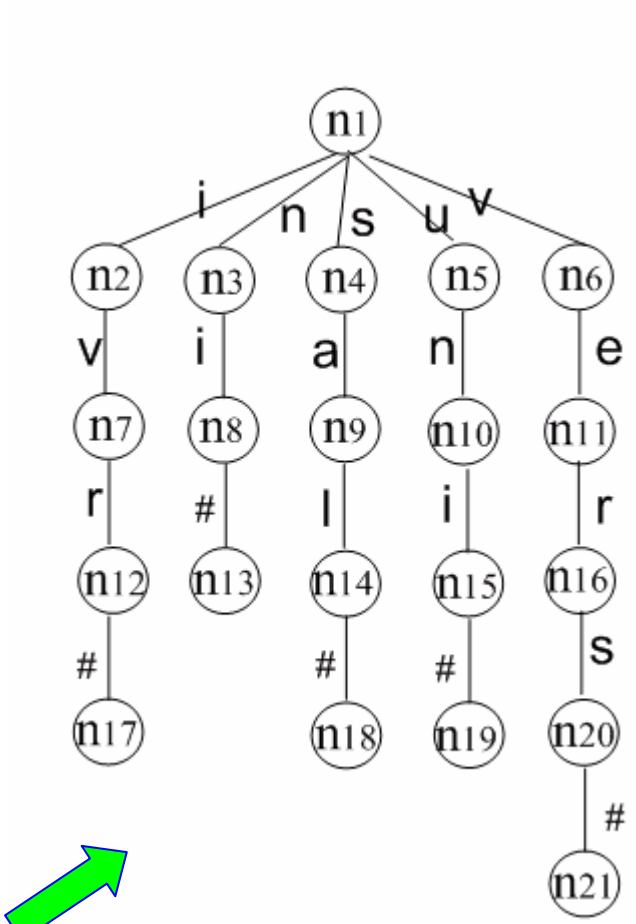


- Variable-length grams



[2,4]-gram

ni
ivr
sal
uni
vers

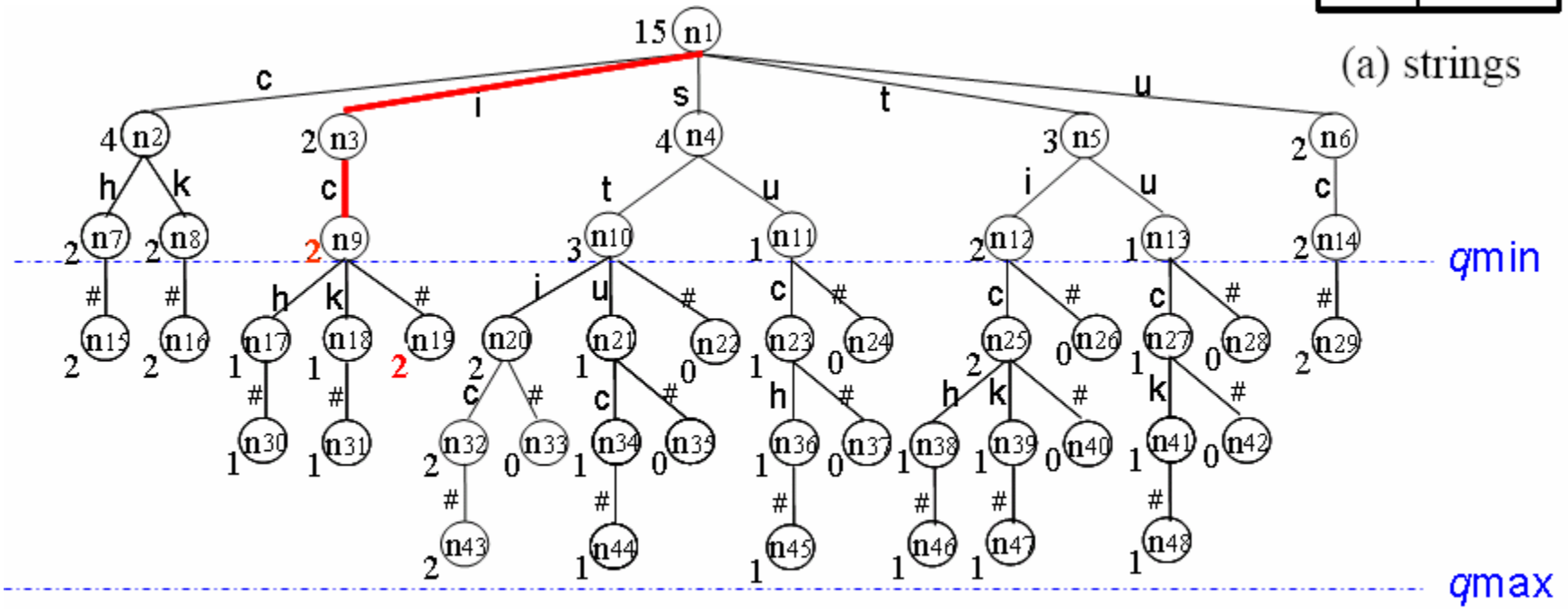


Challenge 2: Constructing gram dictionary

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

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings



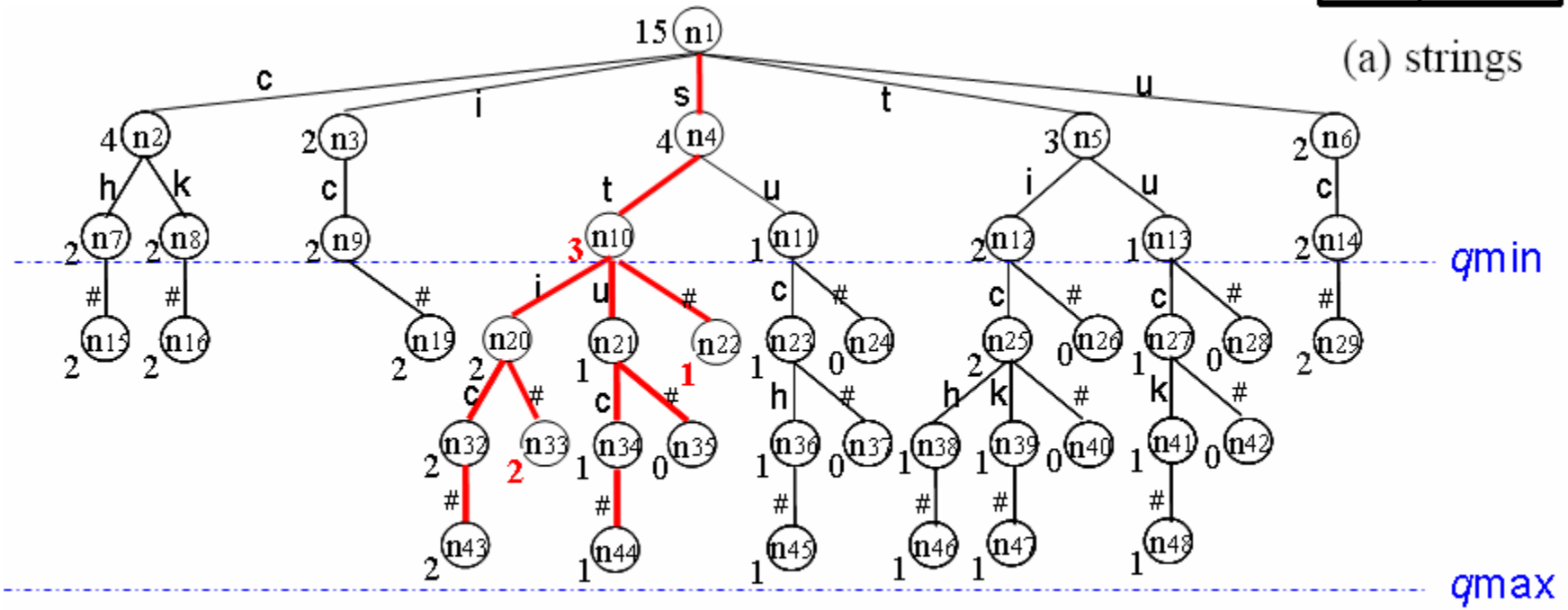
A gram-frequency trie: [2,4]-gram

Challenge 2: Constructing gram dictionary

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

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings

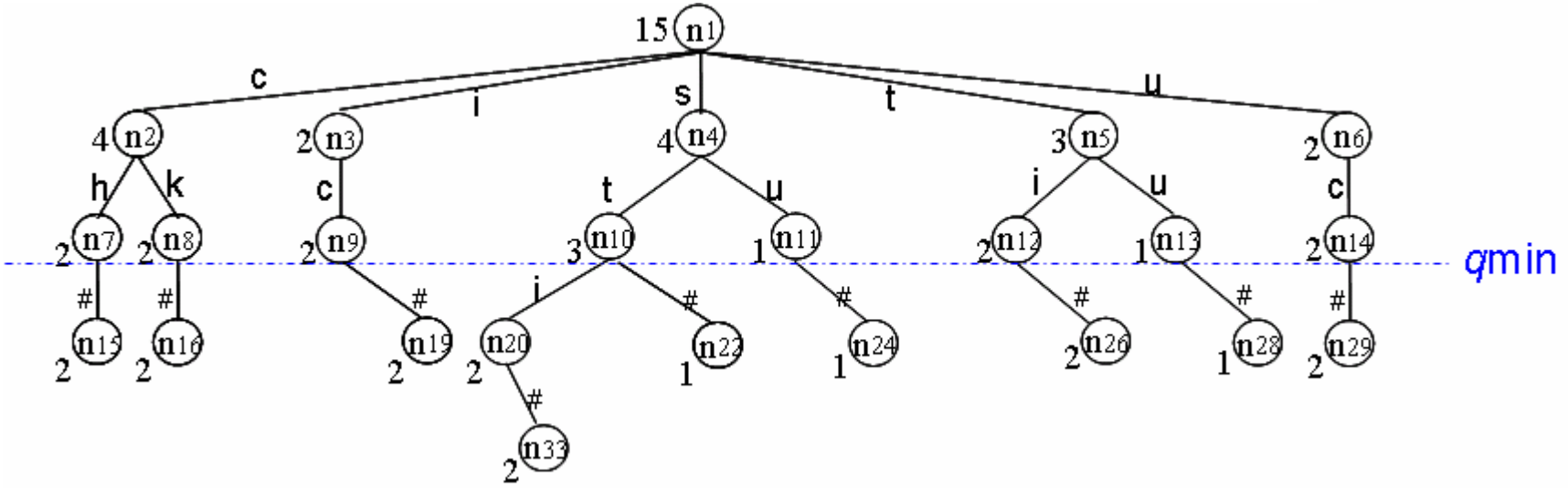


A gram-frequency trie: [2,4]-gram

Final gram dictionary

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings

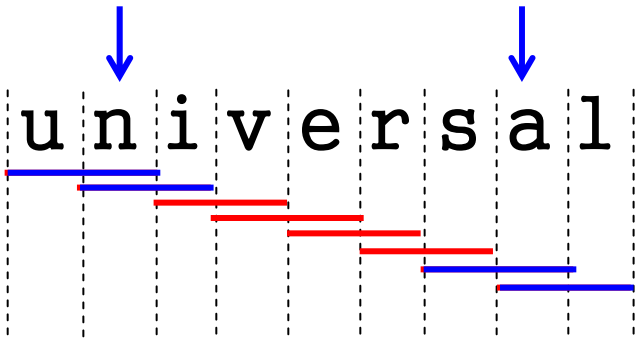


Final grams

Outline

- Motivation
- VGRAM
  - Main idea
  - Decomposing strings to grams
  - Choosing good grams
  - → Effect of edit operations on grams
  - Adopting vgram in existing algorithms
- Experiments

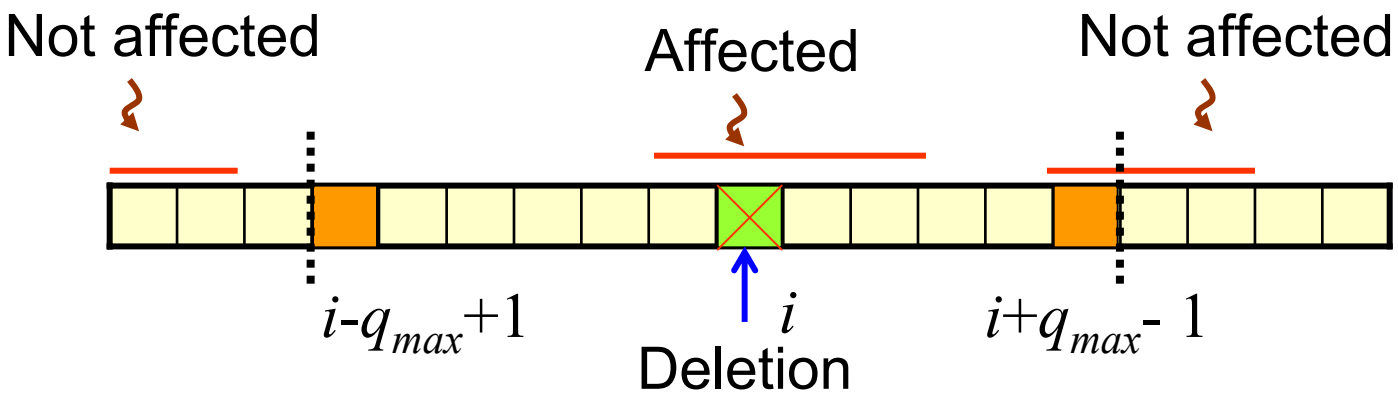
Challenge 3: Edit operation's effect on grams



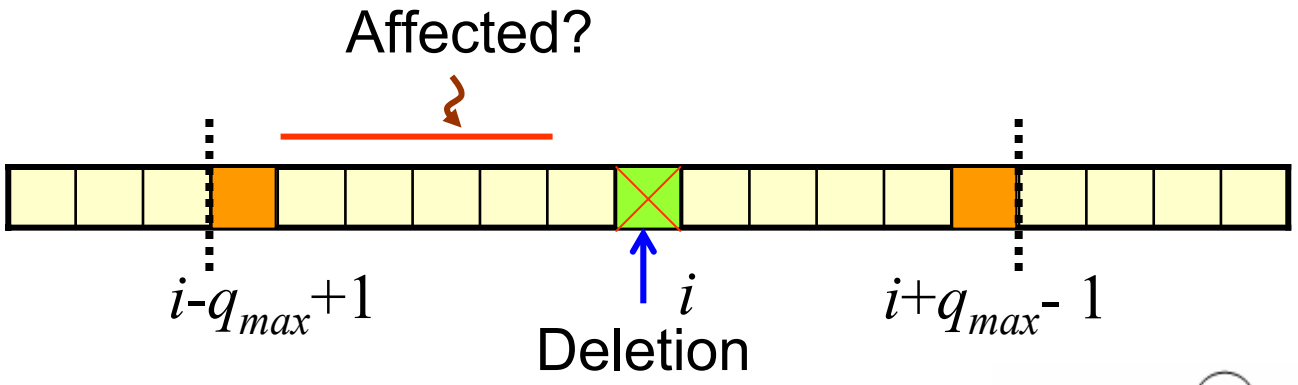
Fixed length: *q*

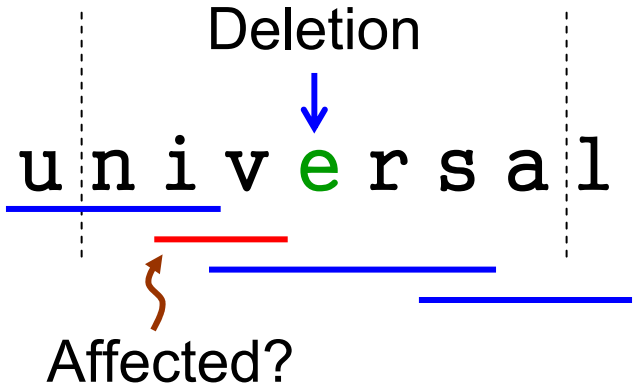
k operations could affect *k* * *q* grams

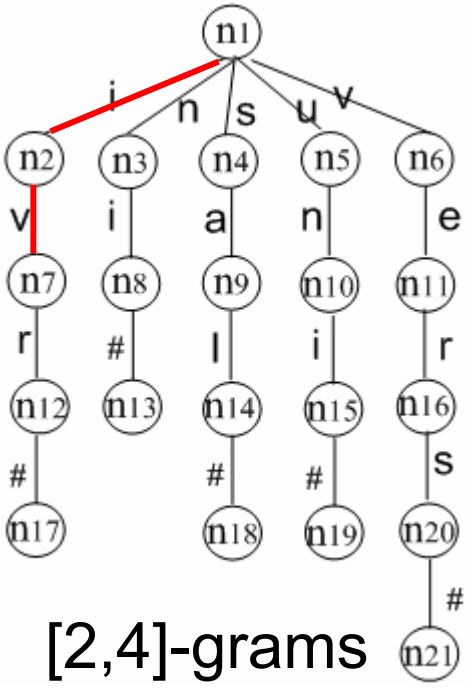
Deletion affects variable-length grams



Grams affected by a deletion

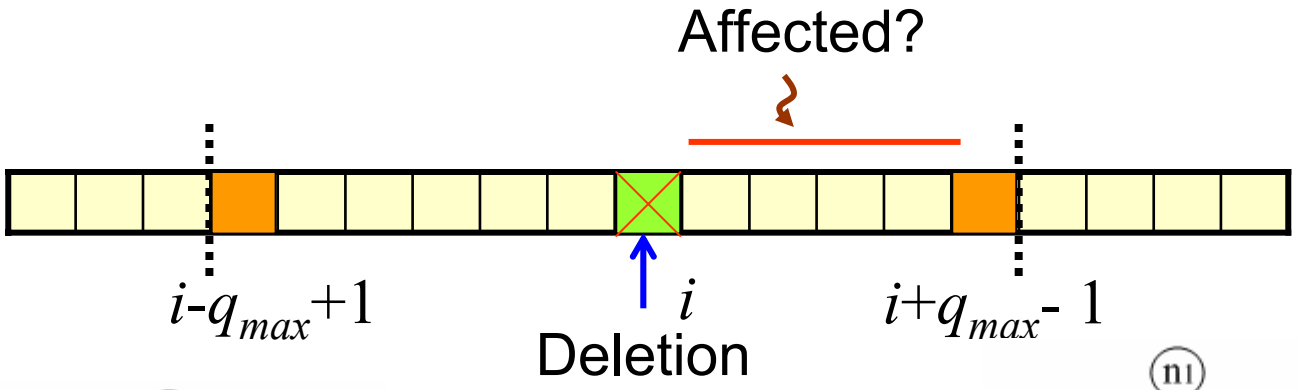


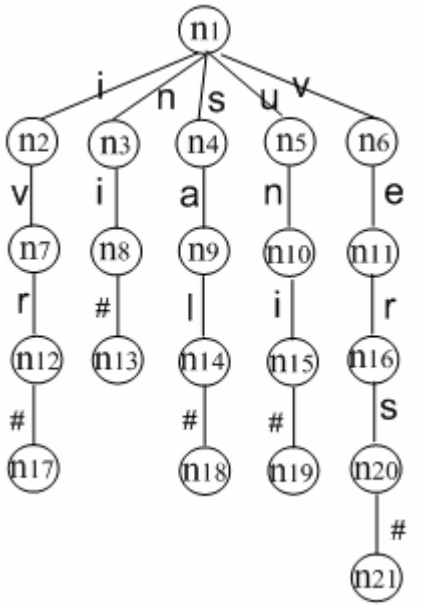




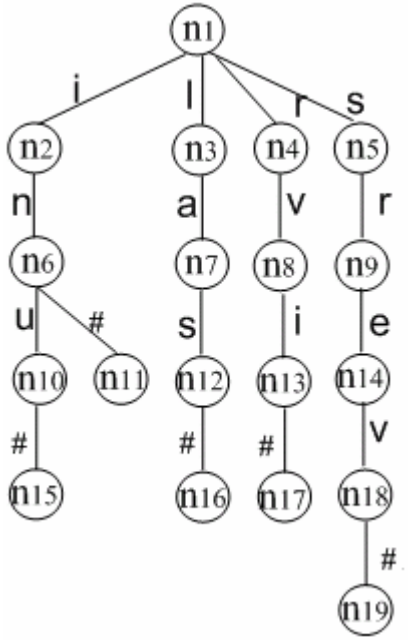
[2,4]-grams

Grams affected by a deletion (cont)



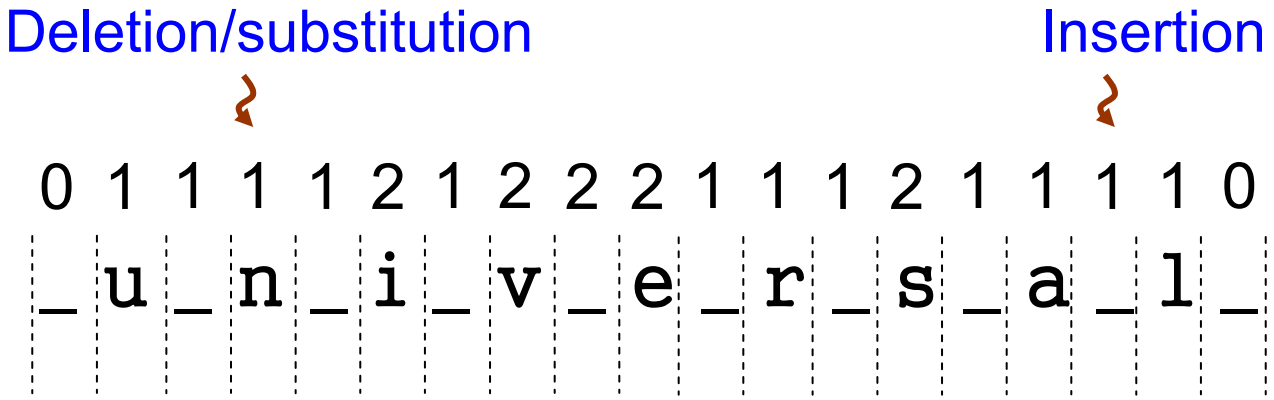


Trie of grams

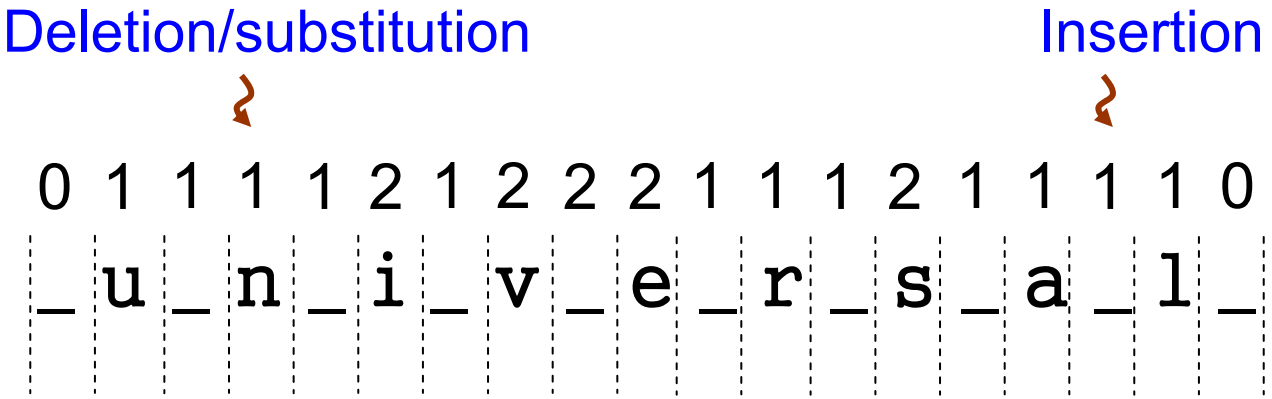


Trie of reversed grams

of grams affected by each operation



Max # of grams affected by k operations



Vector of s = <2,4>

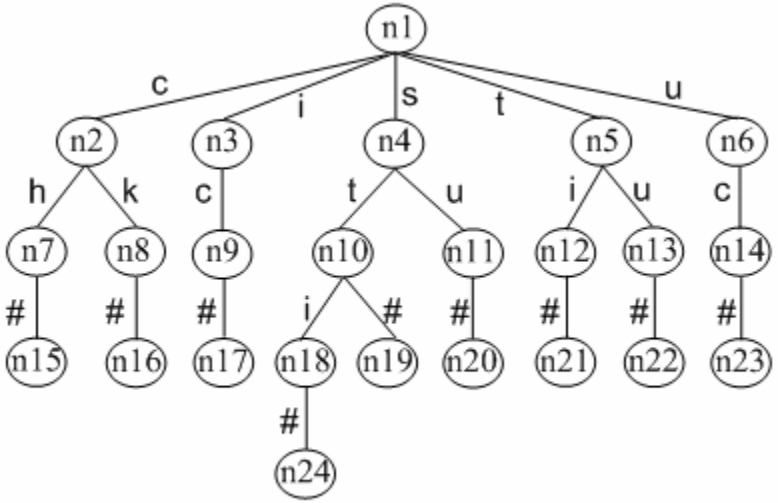
With 2 edit operations, at most 4 grams can be affected

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

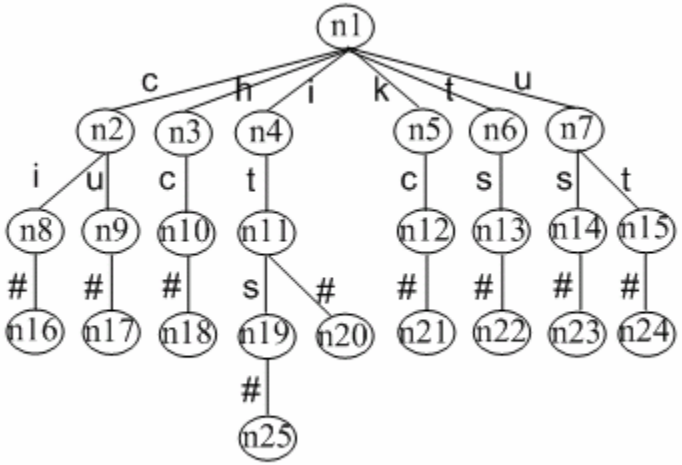
Summary of VGRAM index

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

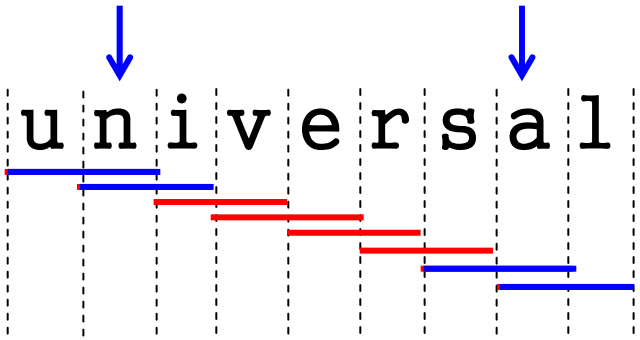
Easily adoptable by many algorithms

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:

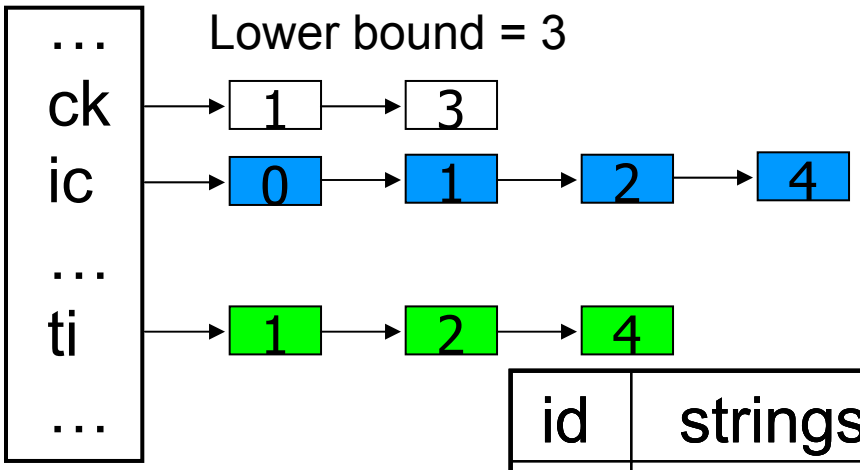
lower bound = # of grams of s1 – NAG(s1,k)

Example: algorithm using inverted lists

- Query: "shtick", ED(shtick, ?)$\leqslant$1

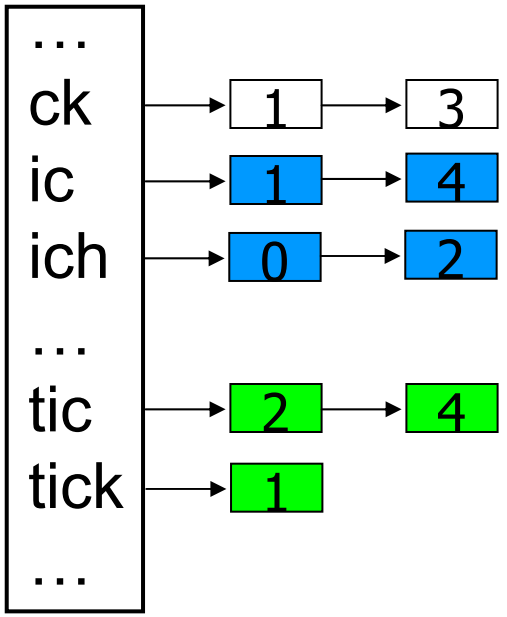
sh ht tick

2-grams



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

2-4 grams



Lower bound = 1

Outline

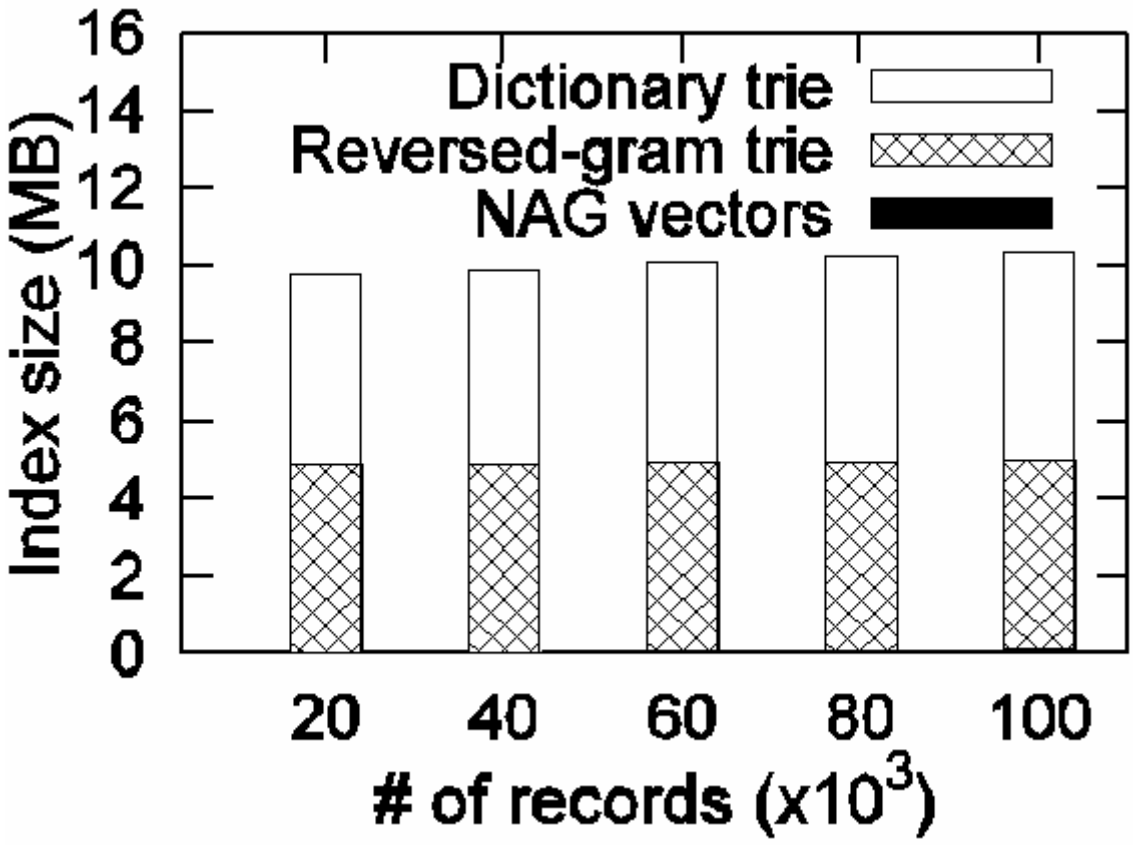
- Motivation
- VGRAM
  - Main idea
  - Decomposing strings to grams
  - Choosing good grams
  - Effect of edit operations on grams
  - Adopting vgram in existing algorithms
- Experiments

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.

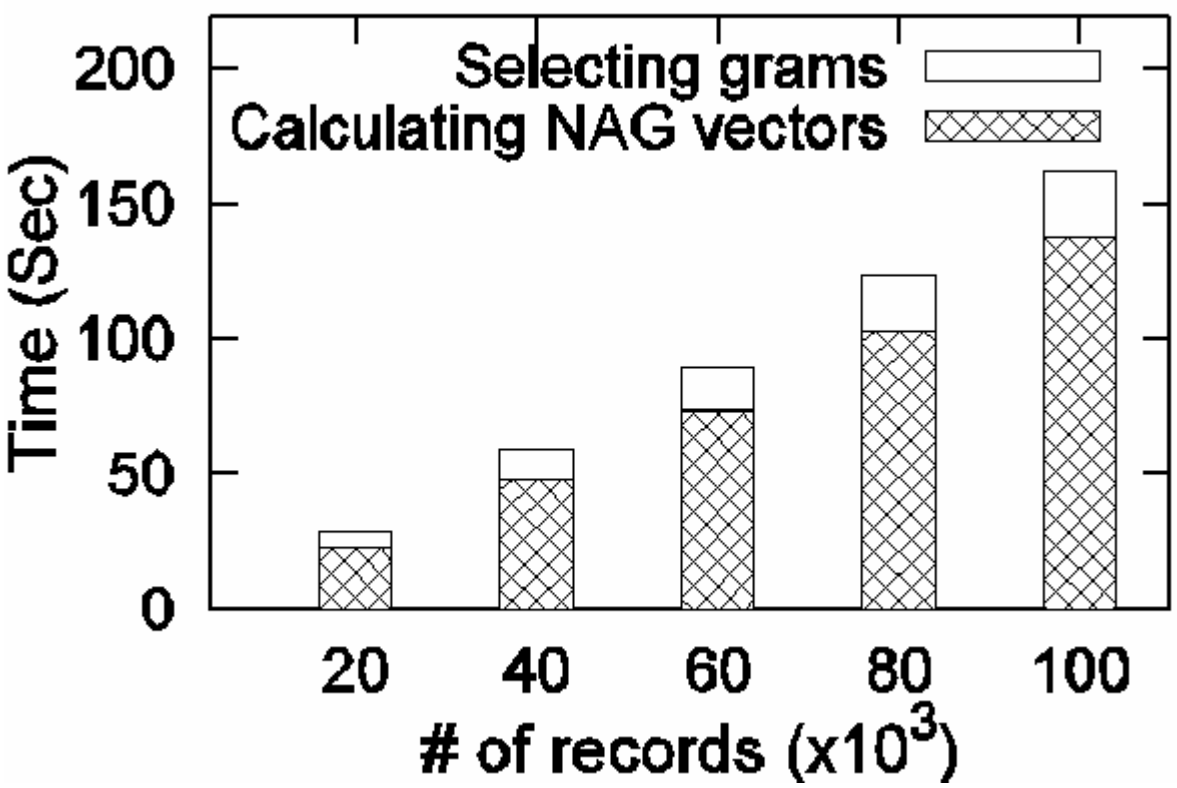
Environment:
VC++, Dell GX620 PC with an Intel Pentium 3.40Hz Dual Core CPU, 2GB memory, Window XP O.S.

VGRAM overhead (index size)



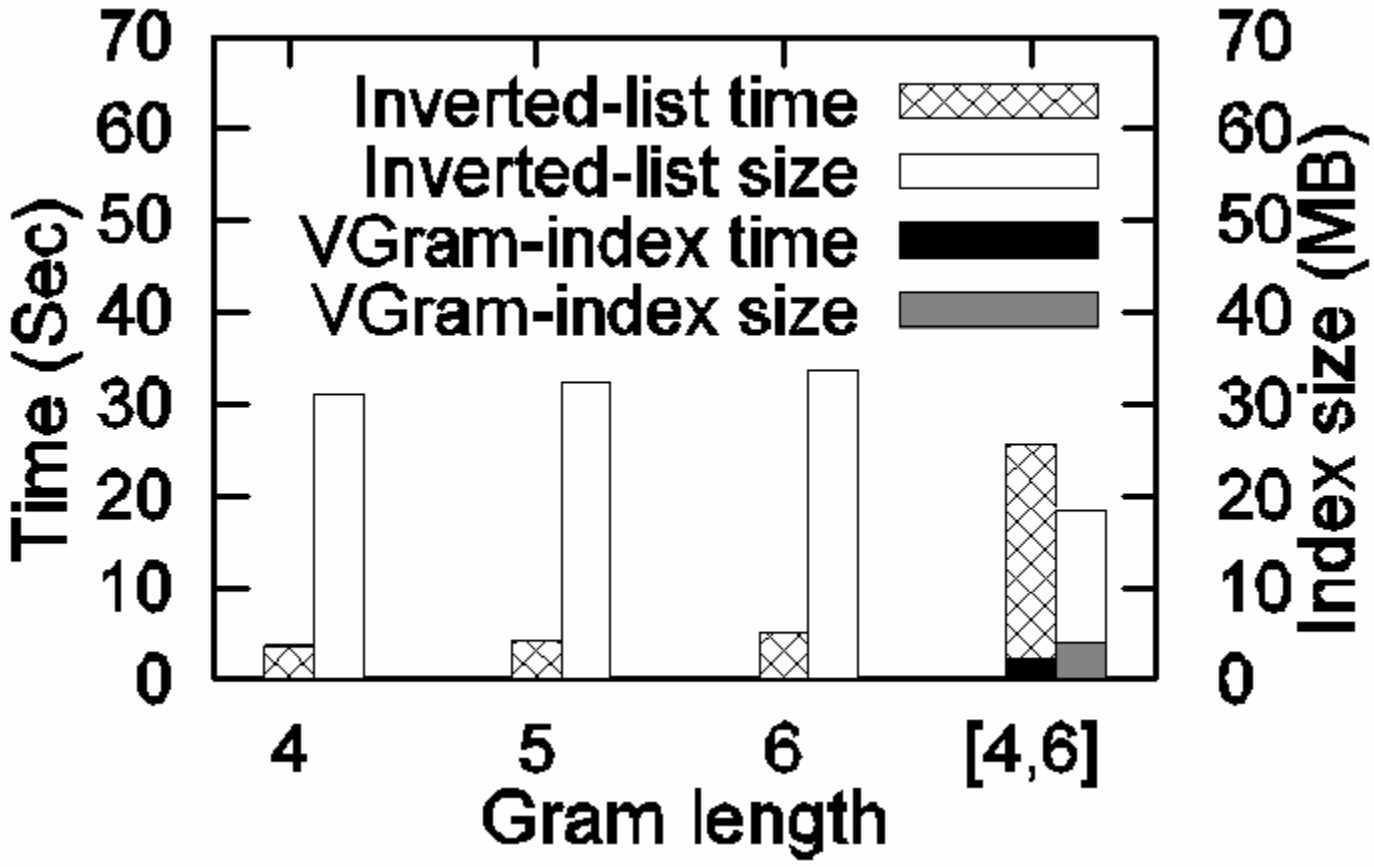
Dataset 3: DBLP titles, [5,7]-gram, T=500, LargeFirst pruning policy

VGRAM overhead (construction time)



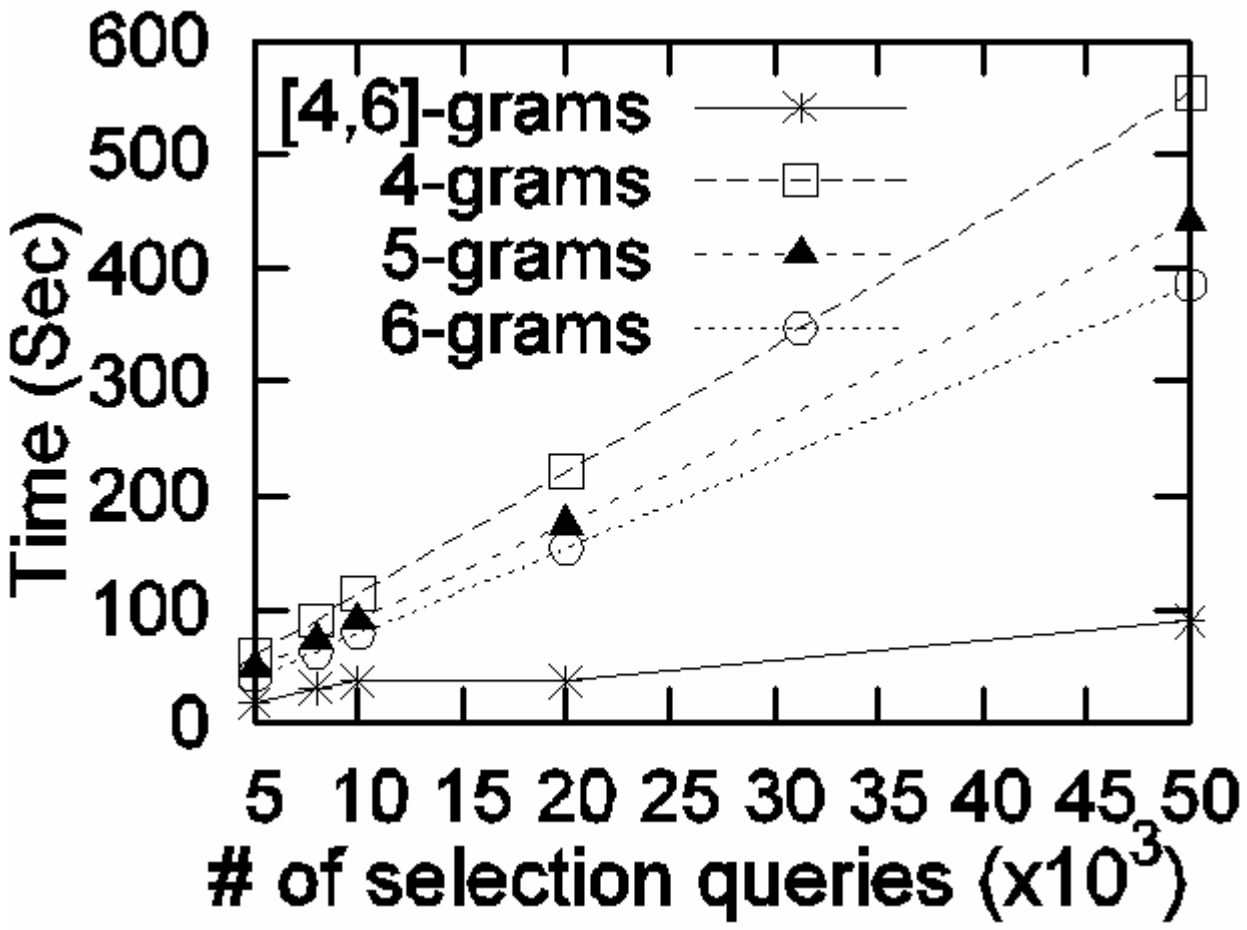
Dataset 3: DBLP titles, [5,7]-gram, T=500, LargeFirst pruning policy

Benefits over fixed-length grams (index)



Dataset 1: 150K Person names, k=1, MergeCount algorithm, T=1000, LargeFirst pruning policy

Benefits over fixed-length grams (running time)

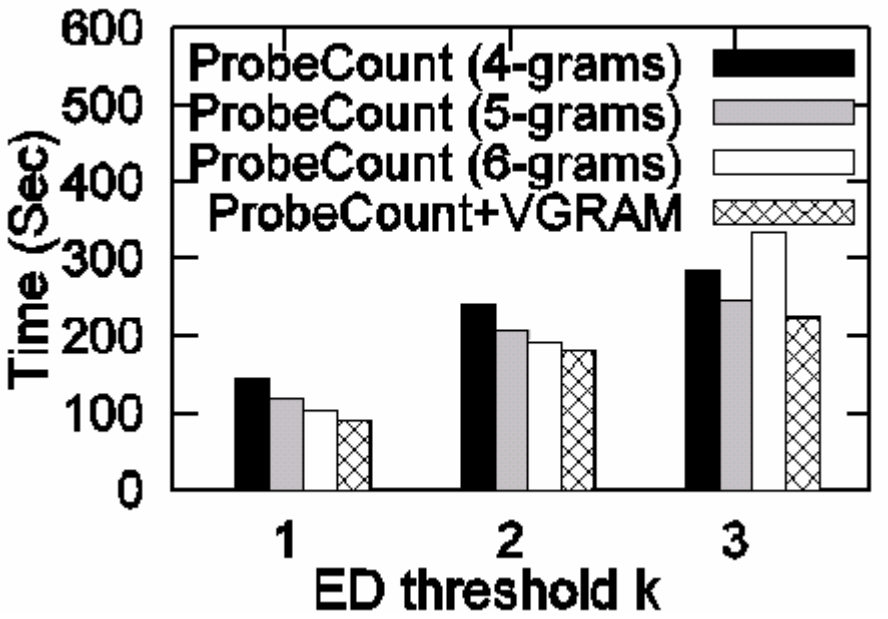


Dataset 1: 150K Person names, k=1, MergeCount algorithm, T=1000, LargeFirst pruning policy

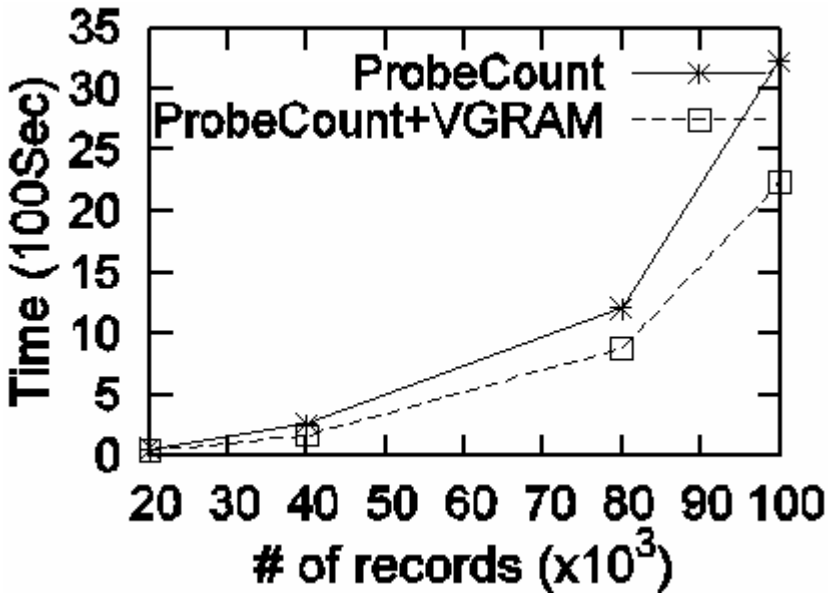
Enhance approximate join algorithms

- ProbeCount
- ProbeCluster
- PartEnum

Improving algorithm ProbeCount



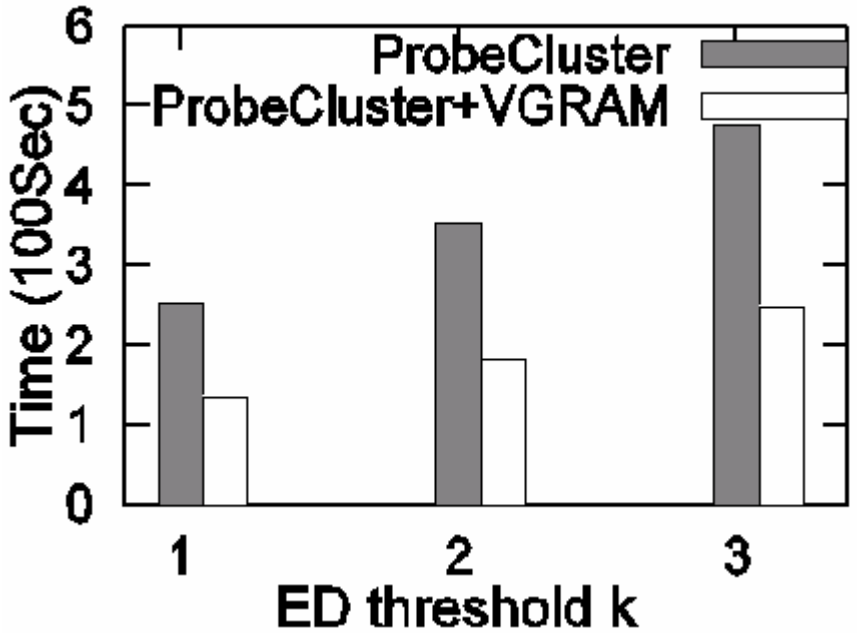
50K person names

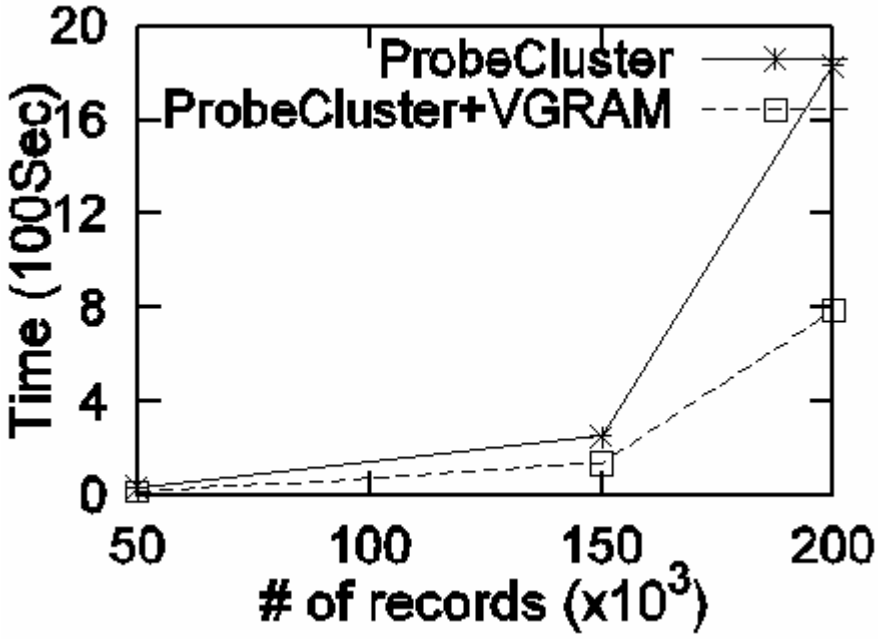


K=3

Dataset 1: [4,6]-gram, T=200, LargeFirst pruning policy

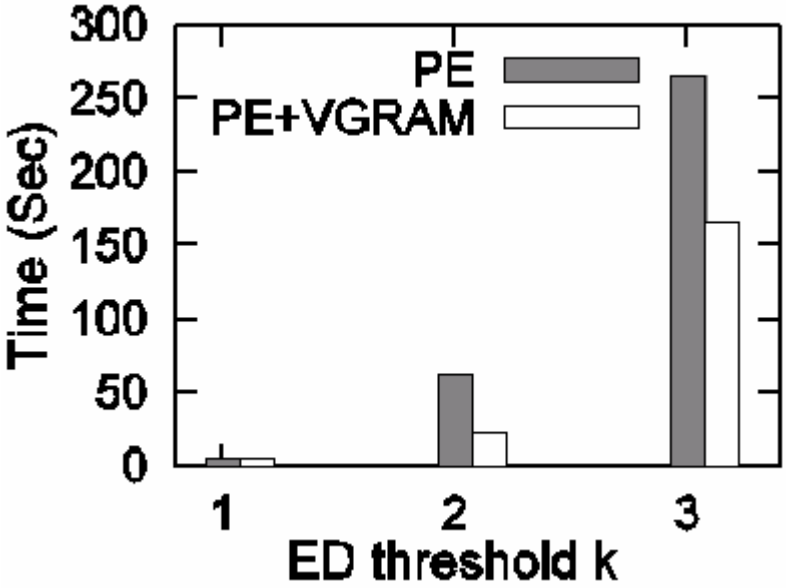
Improving algorithm ProbeCluster

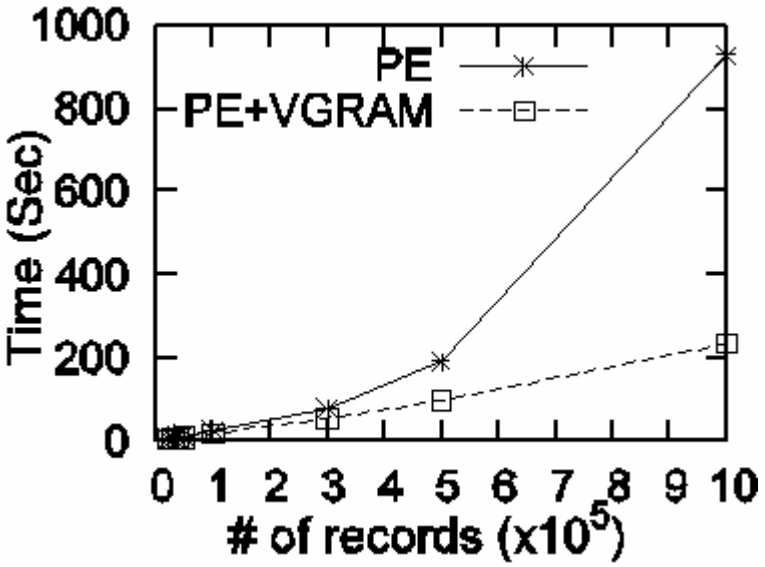




Dataset 1: [5,7]-gram, T=1000, LargeFirst pruning policy

Improving algorithm PartEnum





Dataset 1: [4,6]-gram, T=1000, LargeFirst pruning policy

Conclusions

- VGRAM: using grams of
  - variable-length
  - high-quality
- Adoptable in existing algorithms
  - Reduce index size
  - Reduce running time

Related work

- Approximate String Matching
  - q-Grams, q-Samples
  - Inside DBMS
  - Substring matching
- Set similarity join
- Variable length gram applications
  - Speech recognition, information retrieval, artificial intelligence
  - Substring selectivity estimation
- Improve space and time efficiency
  - n-Gram/2L

Questions or Comments?

Thank you