Approximate Queries on String Collections

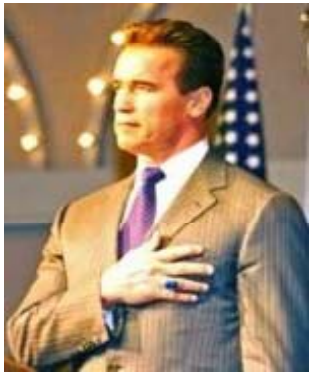
Xiaochun Yang

Institute of Computer Software and Theory
School of Information Science and Engineering
Northeastern University, CHINA
yangxc@mail.neu.edu.cn

Outline

- What is approximate query?
- Q-gram based algorithms
- Our research results
  - VGRAM [VLDB'07]
  - Gram selection [SIGMOD'08]

Example: a movie database



The user doesn't know the exact spelling!

Star	Title	Year	Genre
Keanu Reeves	The Matrix	1999	Sci-Fi
Samuel Jackson	Iron man	2008	Sci-Fi
Schwarzenegger	The Terminator	1984	Sci-Fi
Samuel Jackson	The man	2006	Crime

Gap between Queries and Data

- Errors in the query
  - The user doesn't remember a string exactly
  - The user unintentionally types a wrong string
  - Limited input device

Query: Schwarrzenger.

...

Data : Schwarzenegger

...

Data integration and cleansing

- Errors in the database:
  - Data often is not clean by itself, especially true in data integration and cleansing
    - Typos, Web data, OCR

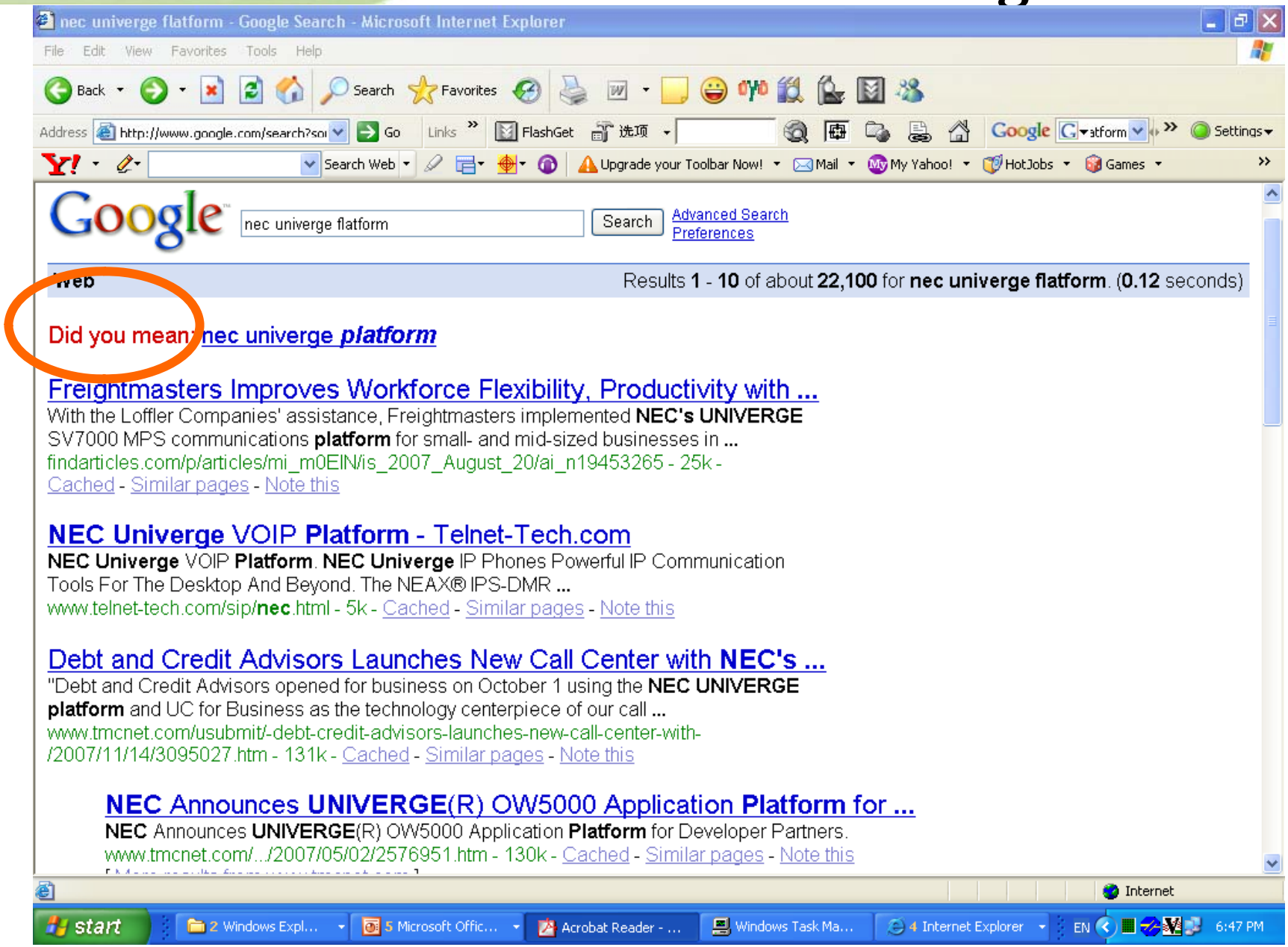
Relation R

Star
Keanu Reeves
Samuel Jackson
Schwarzenegger
Samuel Jackson

Relation S

Star
Keanu Reeves
Samuel L. Jackson
Schwarzenegger
Samuel L. Jackson

Search engine

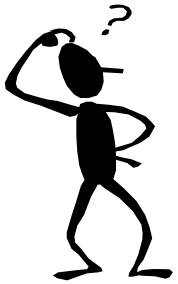


Spell checking

Approximate query

- Approximate search
- Approximate join

Approximate search



Query q

Samule Jackson

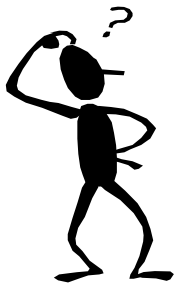
Search

Collection of strings s

Star
Keanu Reeves
Samuel Jackson
Schwarzenegger
Samuel Jackson
...

Output: strings *s* that satisfy **Sim(q,s)≤δ**

Approximate join



Collection R
Jenny Stamatopoulou
John Paul McDougal
Aldridge Rodriguez
Panos Ipeirotis
John Smith
…
…

$\bowtie$

Collection S
Panos Ipirotis
Jonh Smith
…
Jenny Stamatopulou
John P. McDougal
…
Al Dridge Rodriguez

Output: string pair that satisfy **Sim(*r*,*s*)≤δ**

Similarity functions

- Edit distance
- Jaccard similarity
- Hamming distance
- Cosine similarity
- …

Performance is a big issue

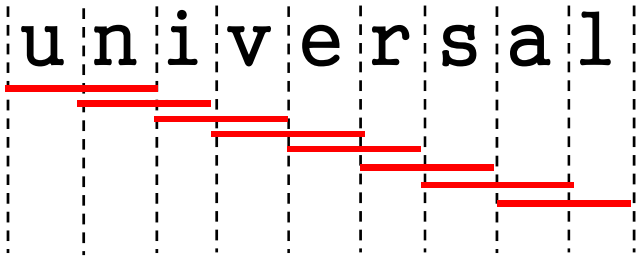
- Answer queries interactively
- Many queries on a server

Response time	5ms/query	20ms/query
Throughput	200 queries/second	50 queries/second

Outline

- What is approximate query?
- Q-gram based algorithms
- Our research results
  - VGRAM [VLDB'07]
  - Gram selection [SIGMOD'08]

“q-grams” of strings



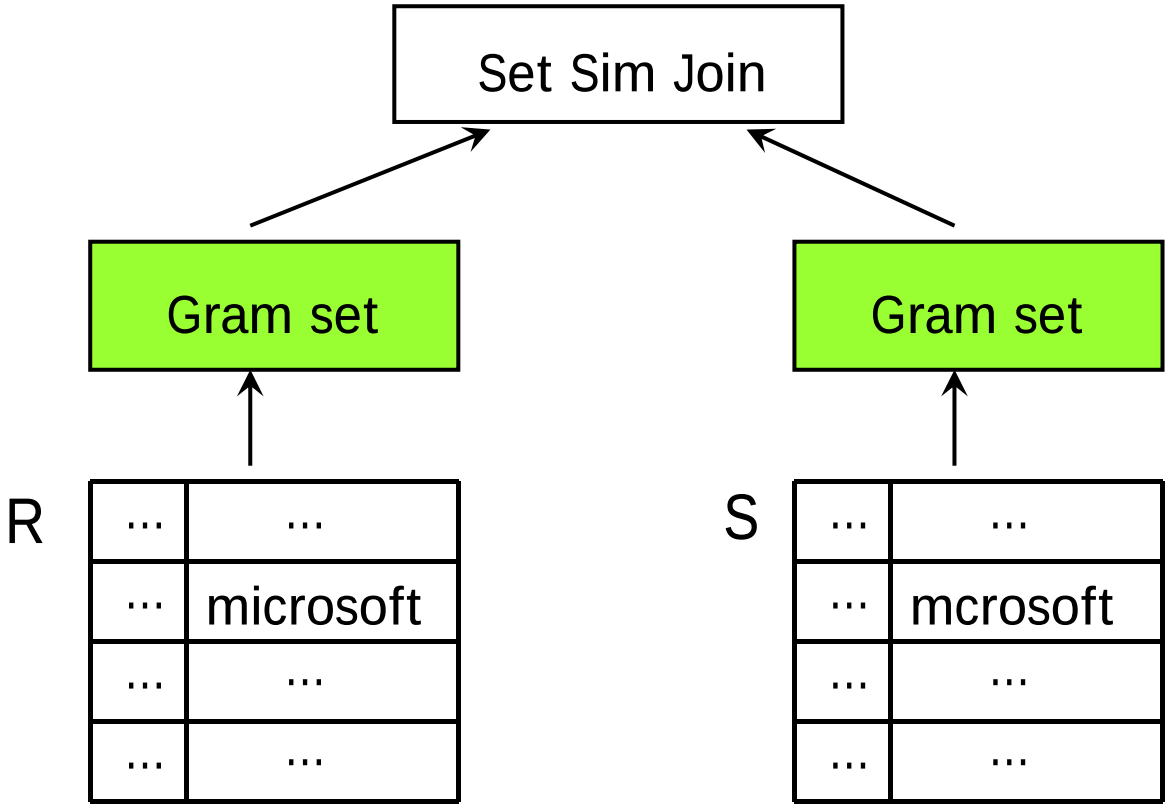
2-grams

V(s,q)

(un, ni, iv, ve, er, rs, sa, al)

String with length L → **L - q + 1** q-grams

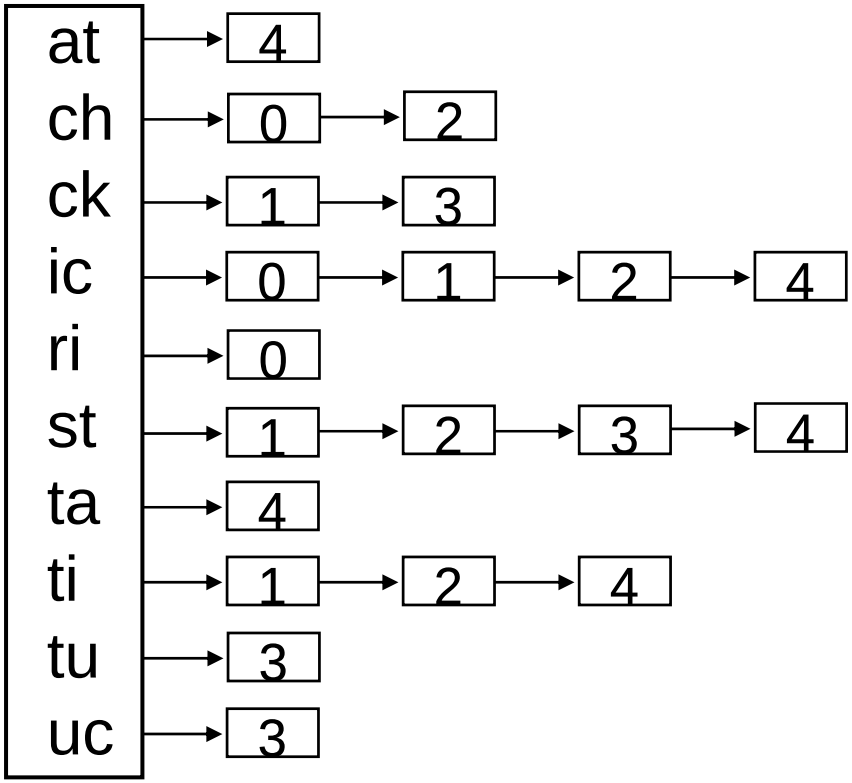
Similarity between gram sets



q-gram based inverted lists index

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

2-grams



Approximate query processing

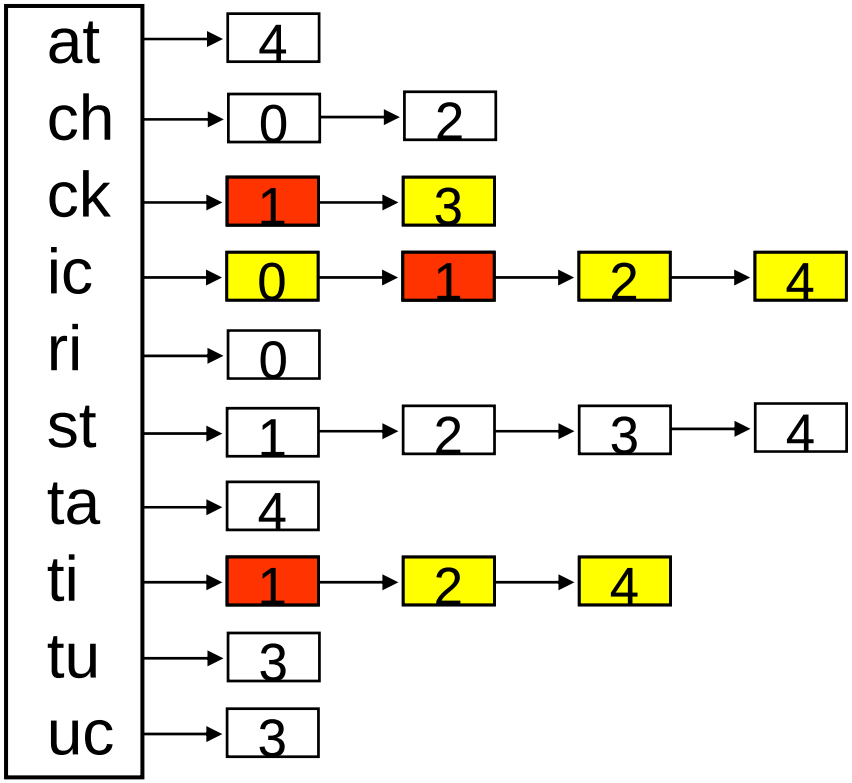
- Query: "shtick", ED(shtick, ?)⩽1

  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?

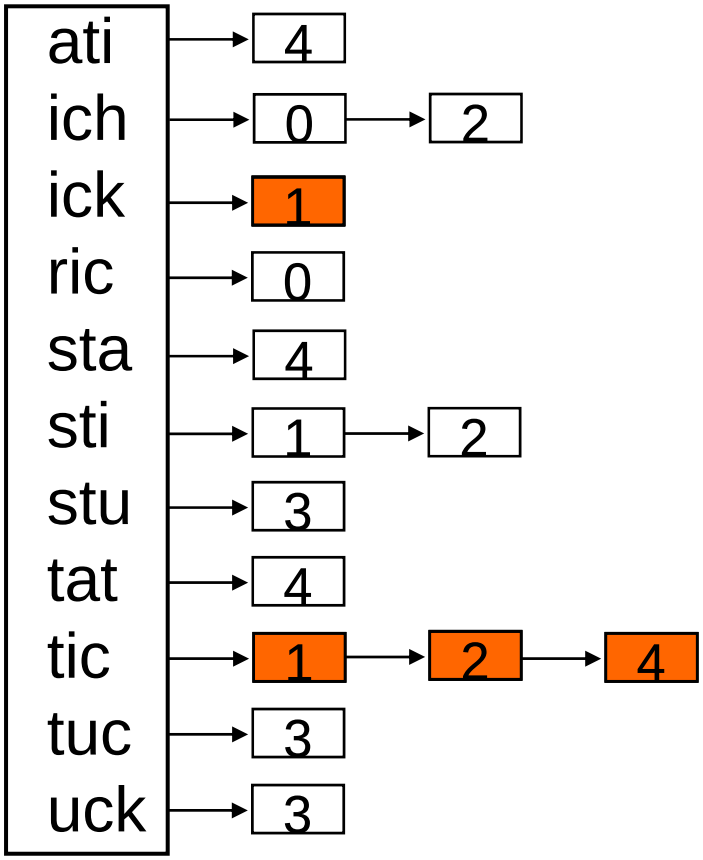
- Query: "shtick", ED(shtick, ?)≤1

  sht hti tic ick

# of common grams >= 1

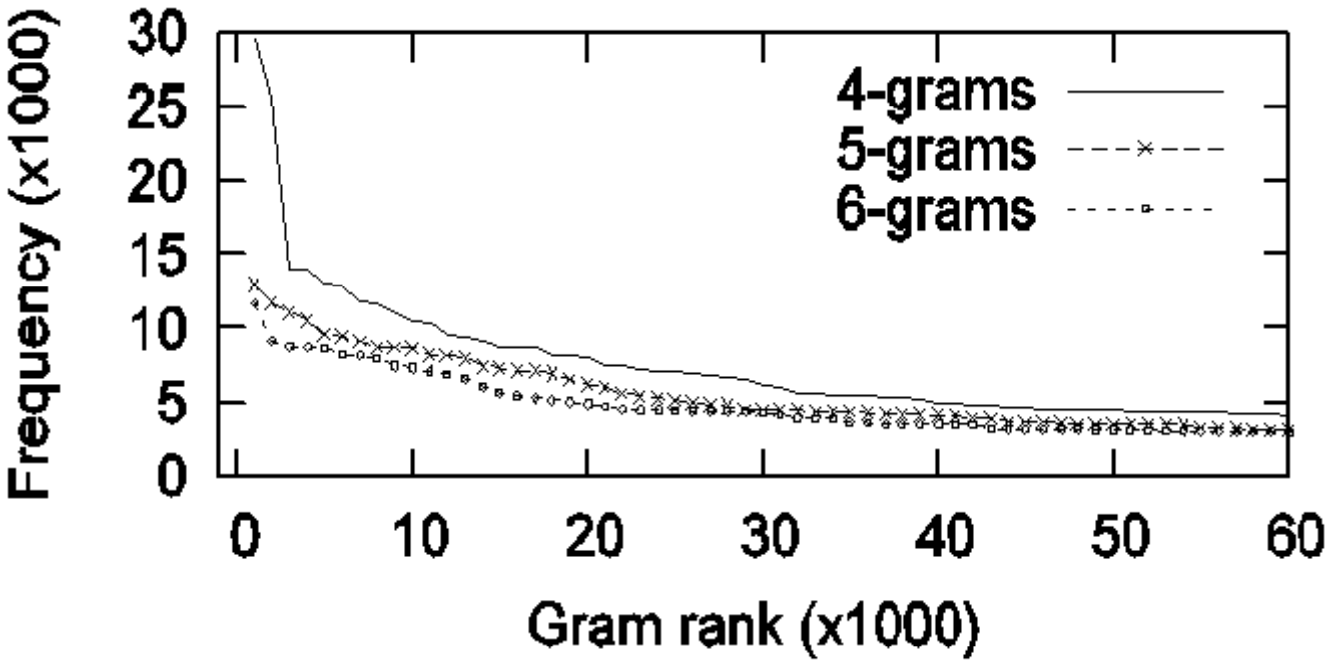
id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

3-grams



Observation 1: skew distributions of gram frequencies

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

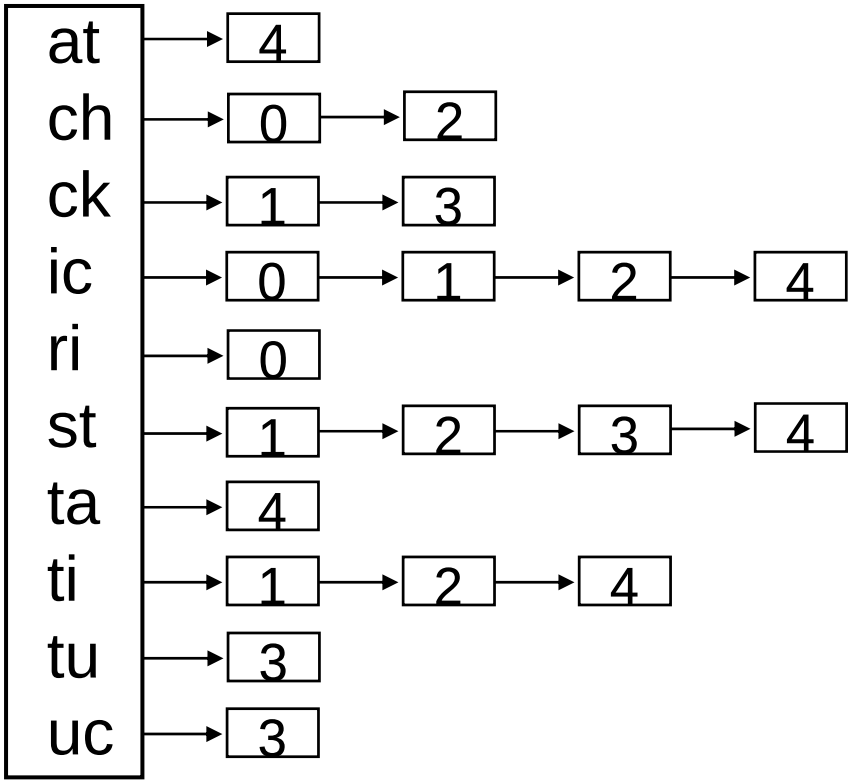


Observation 2: dilemma of choosing "q"

- 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



Motivation

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

What we got?

- VLDB 2007
  - VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams
- SIGMOD 2008
  - Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently

VGRAM: Main idea [VLDB07]

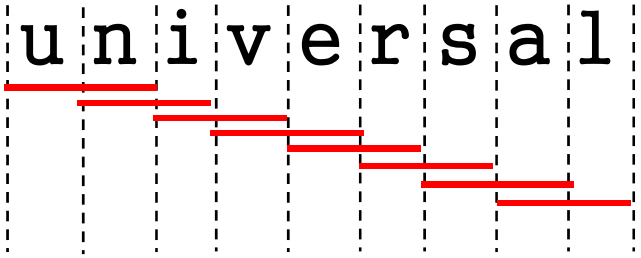
- Grams with variable lengths (between q_{min} and q_{max})
  - `zebra`
    - ze(123)
  - `corrasion`
    - co(5213), cor(859), corr(171)
- Advantages
  - Reduce index size ☺
  - Reducing running time ☺
  - Adoptable by many algorithms ☺

Challenges

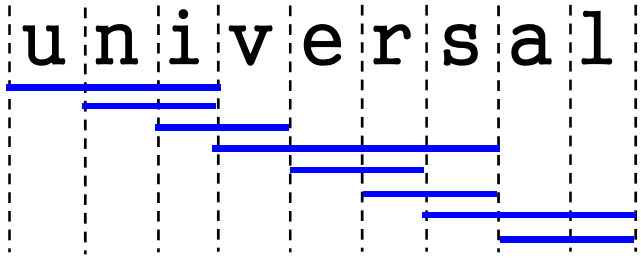
- Generating variable-length grams?
- Constructing a high-quality gram dictionary?
- Relationship between string similarity and their gram-set similarity?
- Adopting VGRAM in existing algorithms?

Challenge 1: String → Variable-length grams?

- Fixed-length 2-grams



- Variable-length grams

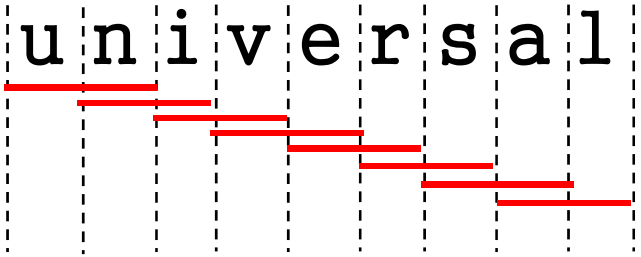


[2,4]-gram dictionary

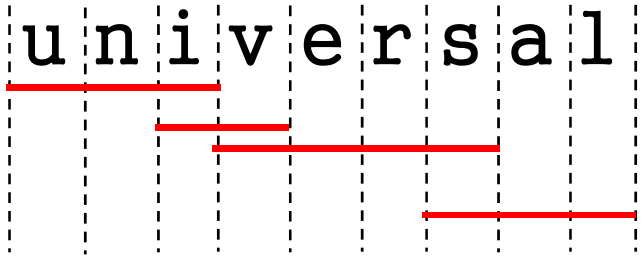
ni
ivr
sal
uni
vers

Representing gram dictionary as a trie

- Fixed-length 2-grams

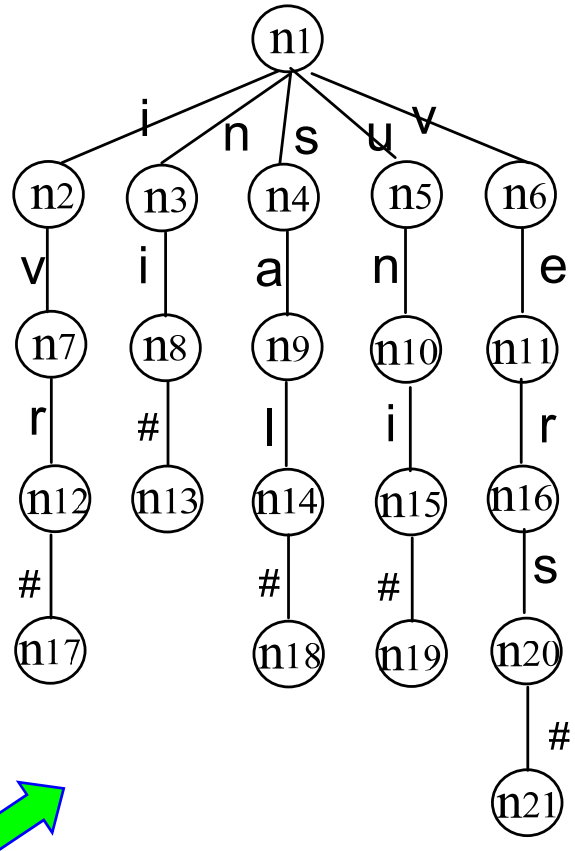


- Variable-length grams



[2,4]-gram

ni
ivr
sal
uni
vers

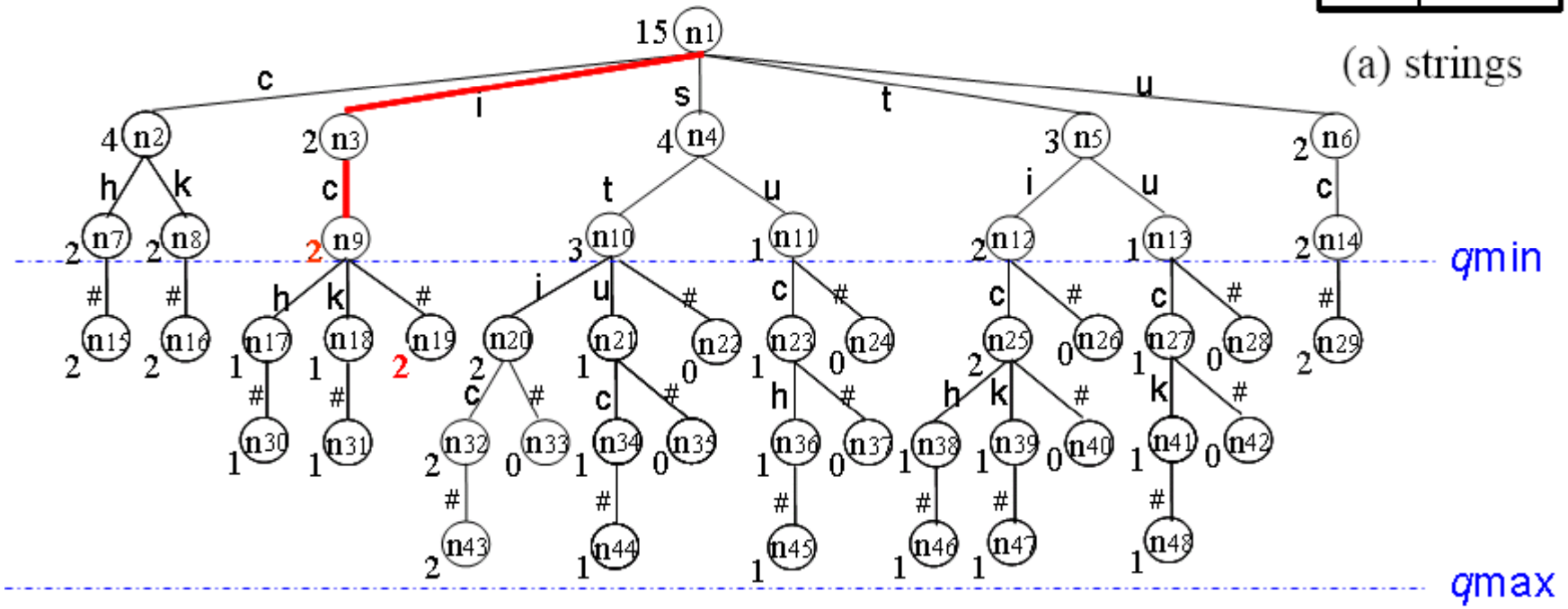


Challenge 2: Constructing gram dictionary

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

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings



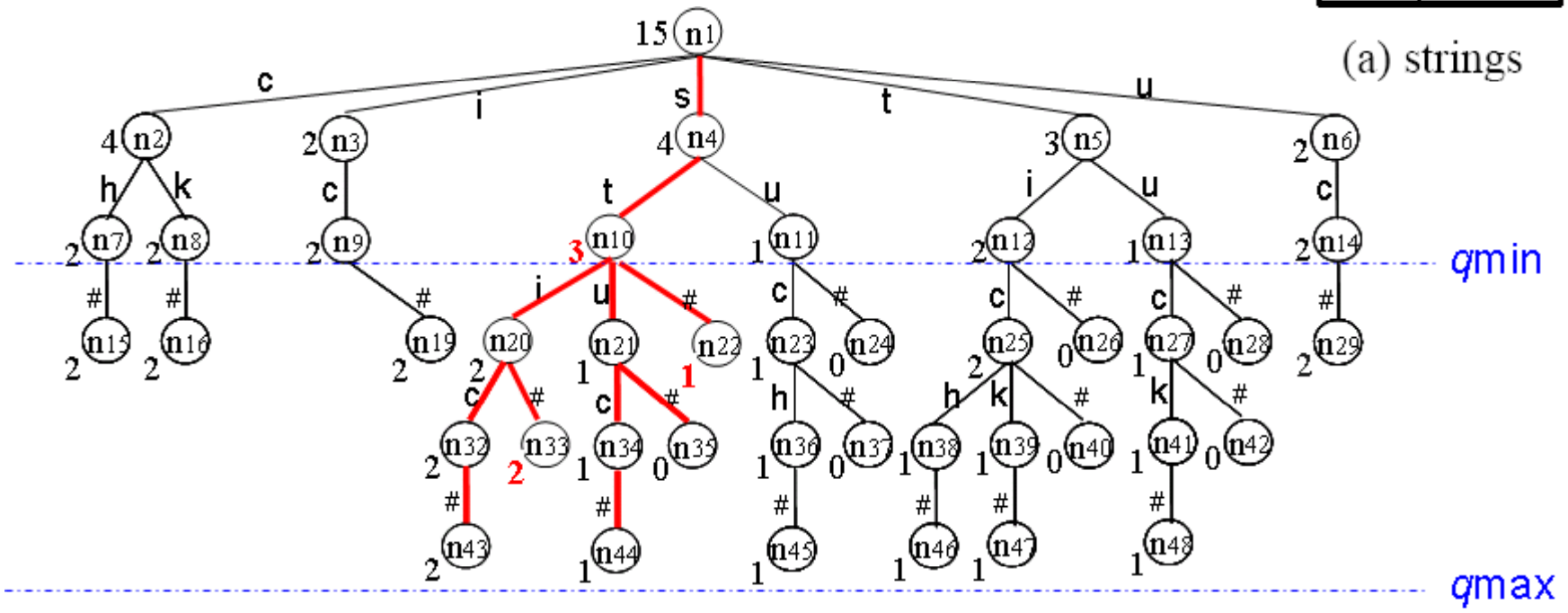
A gram-frequency trie: [2,4]-gram

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

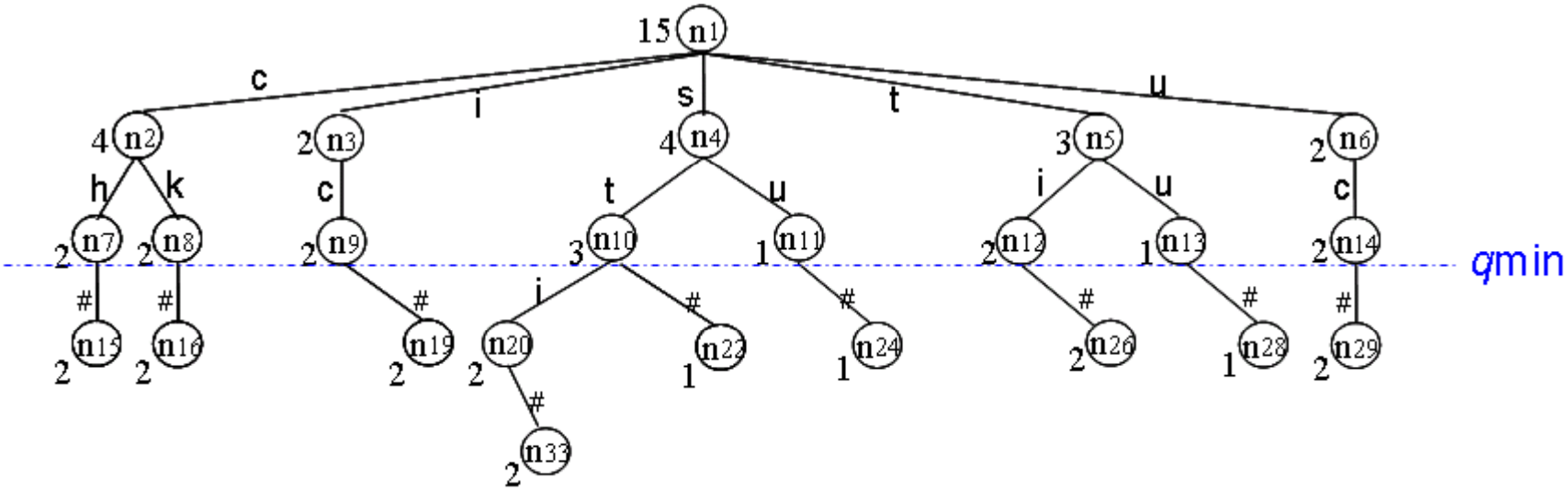


A gram-frequency trie: [2,4]-gram

Final gram dictionary

id	string
0	stick
1	stich
2	such
3	stuck

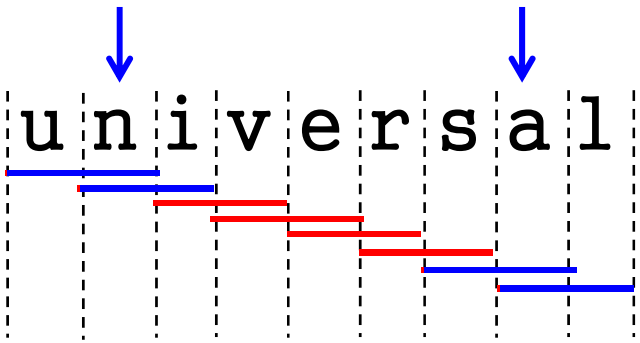
(a) strings



Final grams

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

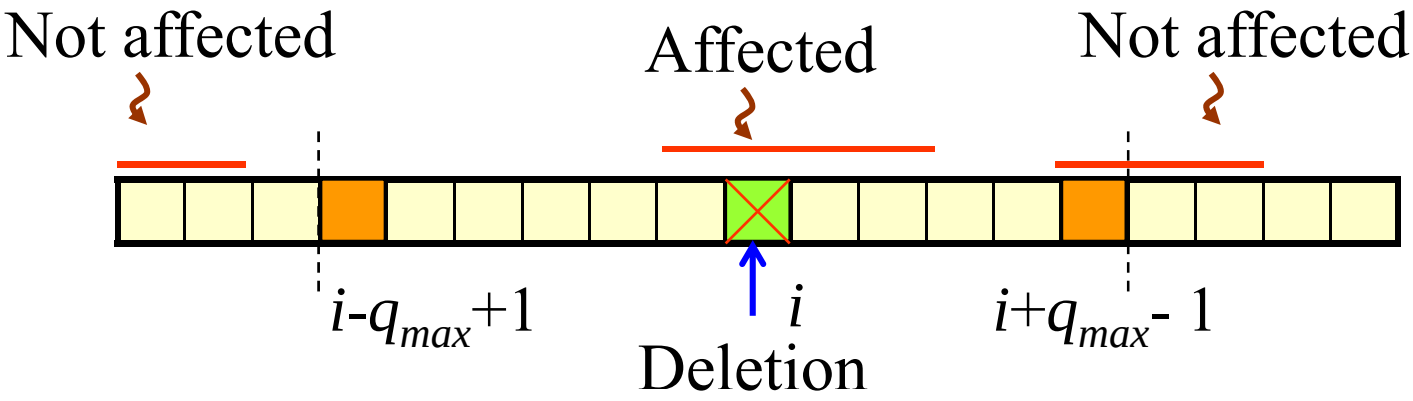
Challenge 3: Edit operation's effect on grams



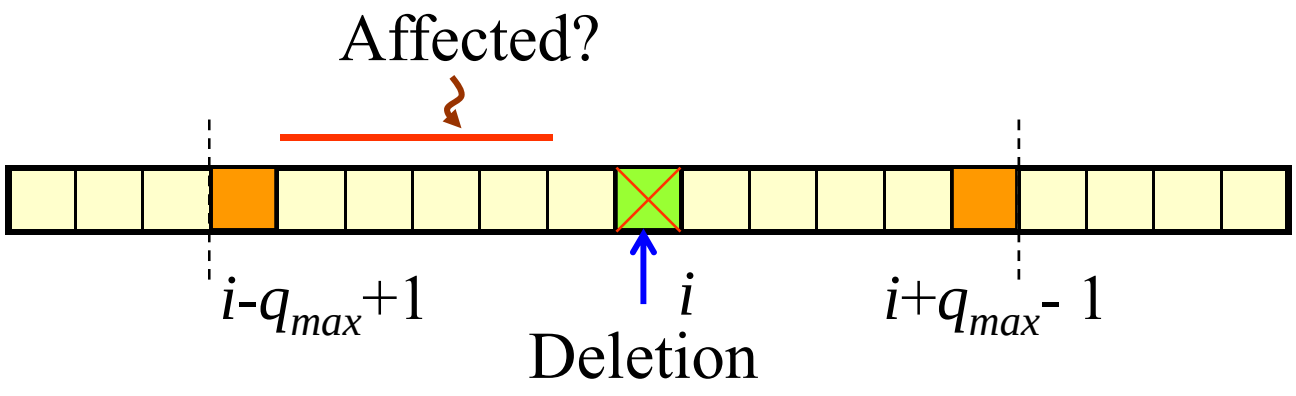
Fixed length: q

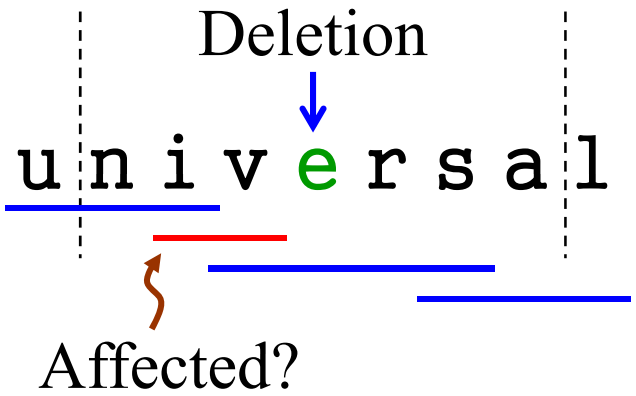
k operations could affect $k * q$ grams

Deletion affects variable-length grams



Grams affected by a deletion





[2,4]-grams

ni
ivr
sal
uni
vers

of grams affected by each operation

Deletion/substitution

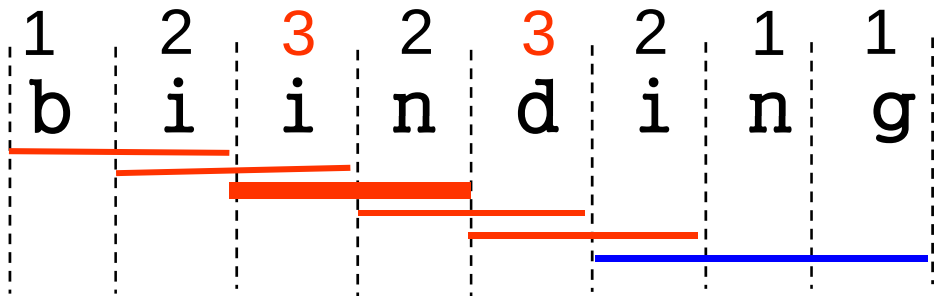
Insertion

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	_

of grams affected by k operations

- *k*-Max: Summation of *k* largest values

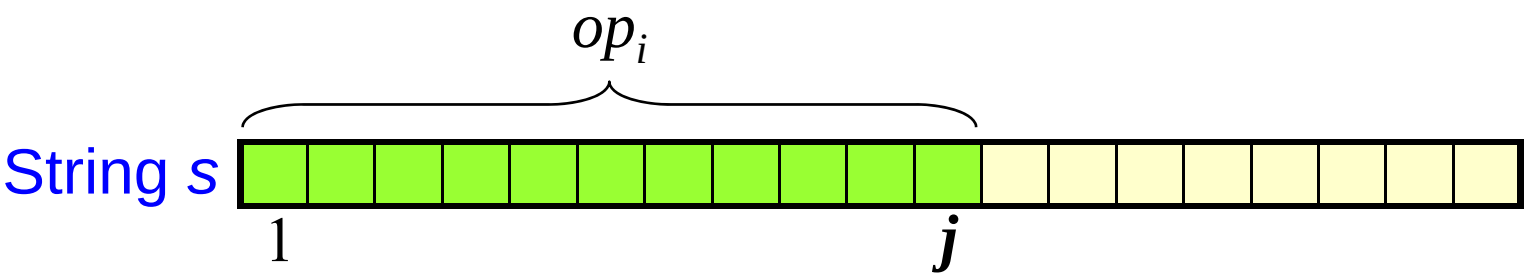
NAG(s,2)=3+3=6



Too pessimistic?

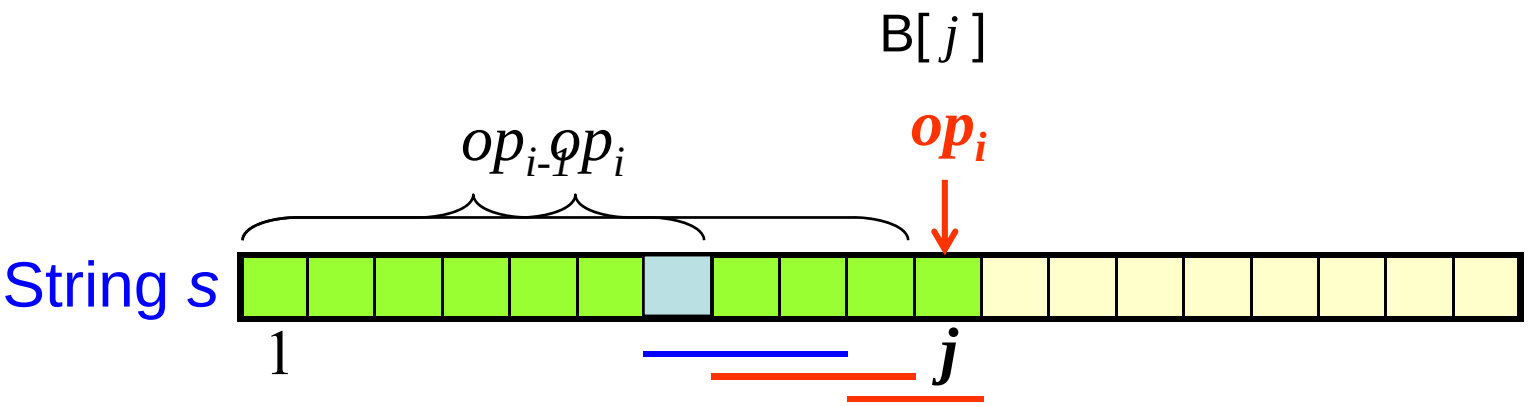
Tightening NAG(s,k) [SIGMOD'08]

- Dynamic programming: tightening NAG(s,k)
  - Subproblems: NAG(s[1,*j*], *i*)



Dynamic programming

- Recurrence function



Dynamic programming

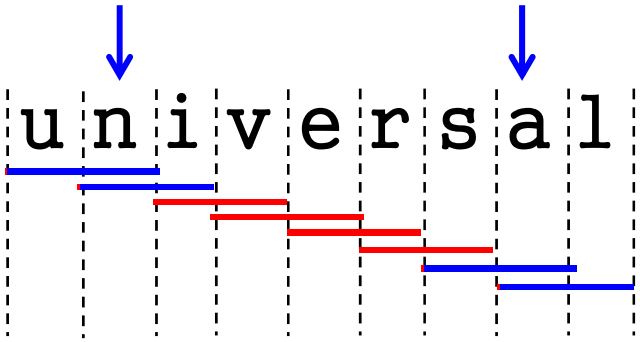
$$P(i,j) = \max \begin{cases} P(i,j-1), & \text{(no operation at } j\text{)} \\ P(i-1,R(j)) + B[j]. & \text{(operations at } j\text{)} \end{cases}$$

	1 b	2 i	3 i	2 n	3 d	2 i	1 n	1 g
k=0	0	0	0	0	0	0	0	0
k=1	0	1	2	3	3	3	3	3
k=2	0	1	2	3	4	5	5	5

NAG vector

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)

Challenge 4: adopting VGRAM

Easily adoptable by many algorithms

Basic interfaces:

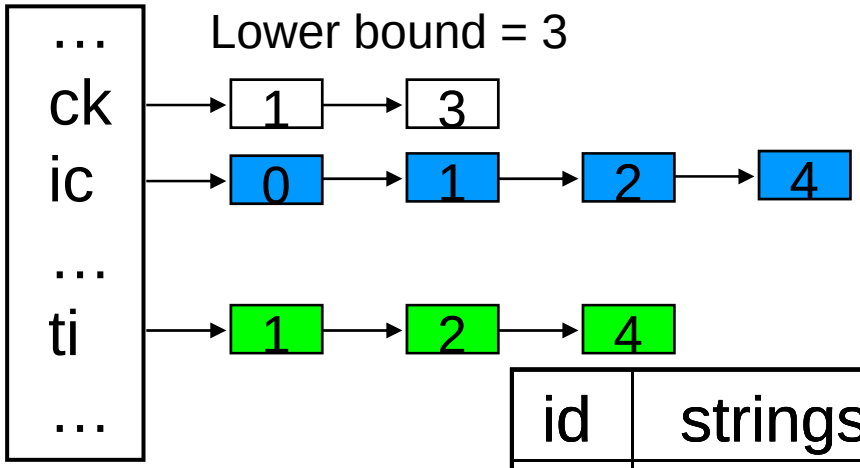
- String s → grams
- String s1, s2 such that ed(s1,s2) <= k
  → min # of their common grams

Example: algorithm using inverted lists

- Query: "shtick", ED(shtick, ?)≤1

sh ht tick

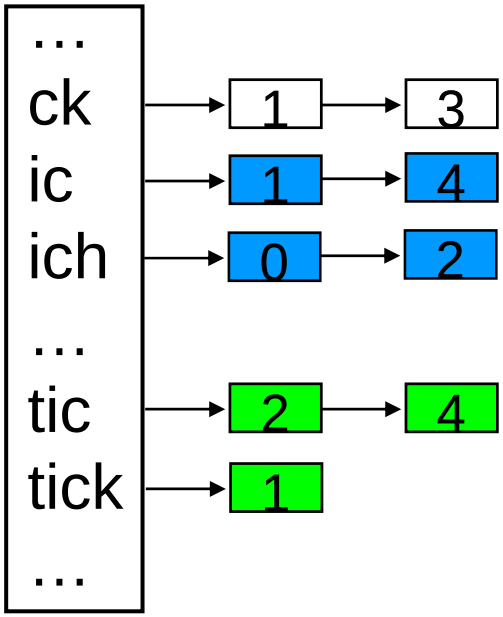
2-grams



Lower bound = 3

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

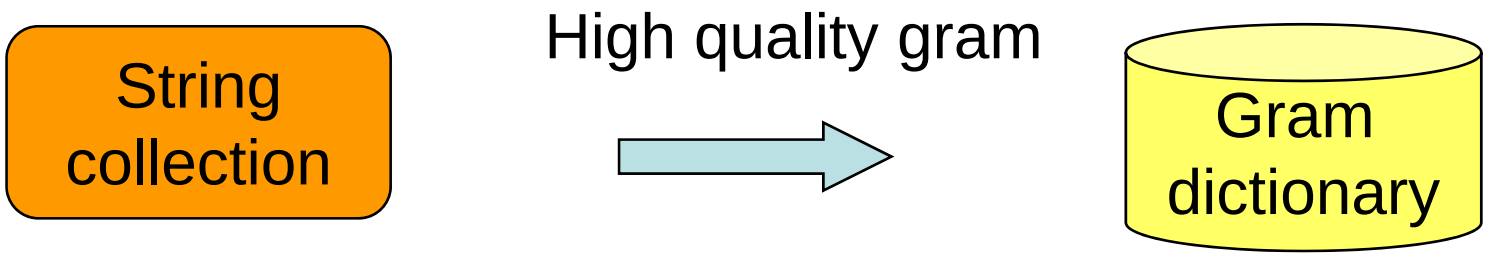
2-4 grams



Lower bound = 1

Gram selection [SIGMOD'08]

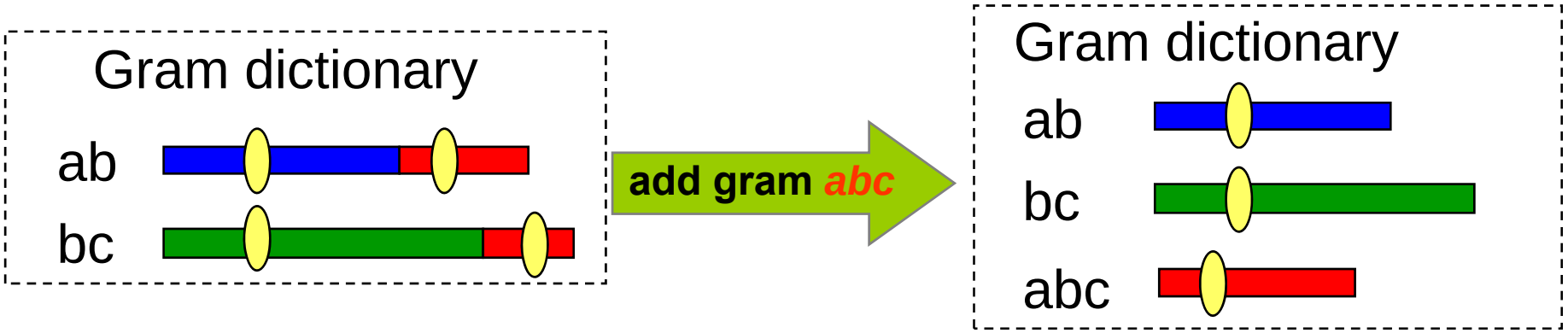
- Cost-based quantitative approach
  - Analyze and estimate query performance when adding each gram
  - Automatically find high-quality grams



Outline

- Effects of adding a gram on index and queries
- Cost-based construction of gram dictionary
- Experiments

Effects on inverted lists

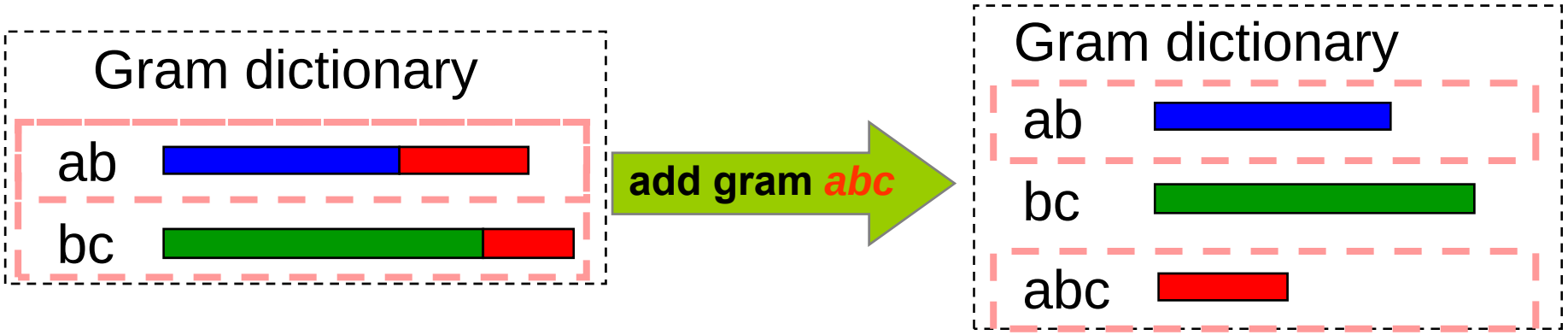


string --bc--

Effects on query performance

- Decrease query's inverted list
- Change lower bound
- Change # of candidates

Effects on query's inverted lists



Query *Q*

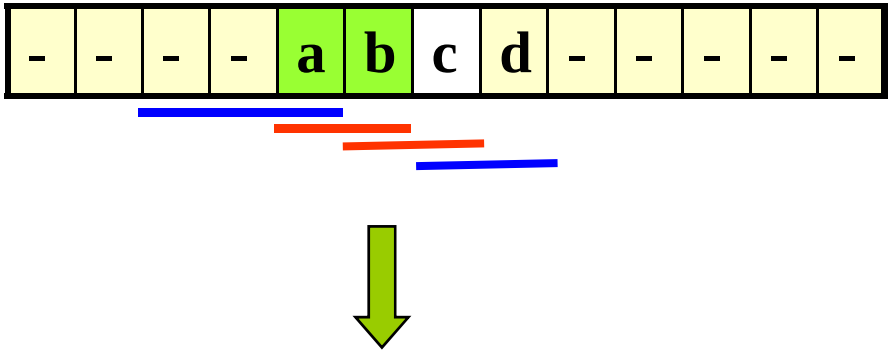
-	-	-	-	-	a	b	c	-	-	-	-	-

➔Adding a new gram ***abc*** will not change or decrease the query's inverted lists

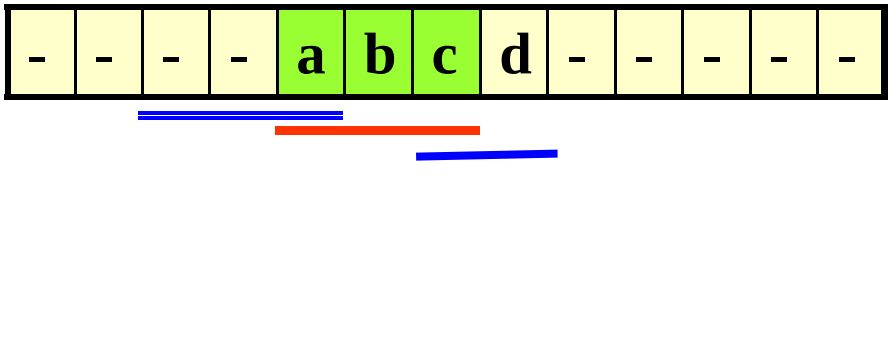
Effects on lower bound

- Query: Q, ED(Q, ?)≤1

Query *Q*



Query *Q*



Effects on lower bou

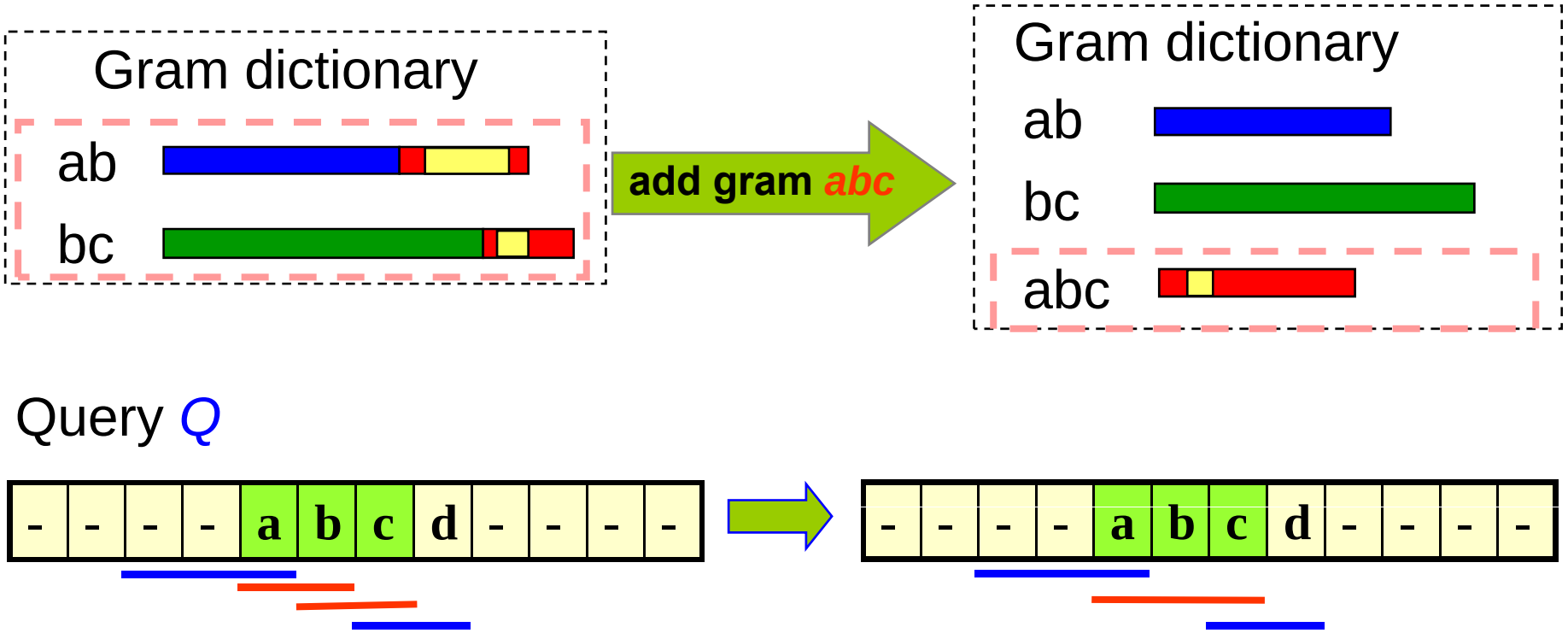
id	strings
1	bingo
2	bioinng
3	bitingin
4	biting
5	boing
6	going

- Query: "bingon", ED(bingon, ?)≤1

Dictionary	Gram set: VG(Q)	|VG(Q)|	NAG(Q,1)	Lower bound
D_0	{bi, in, ng, go, on}	5	2	3
D_1 (+ ing)	{bi, ing, go, on}	4	2	2
D_2 (+ bin)	{bin, ing, go, on}	4	2	2

Effects on # of candidates

- Change lower bound ➔ change # of candidates



Effects of queries

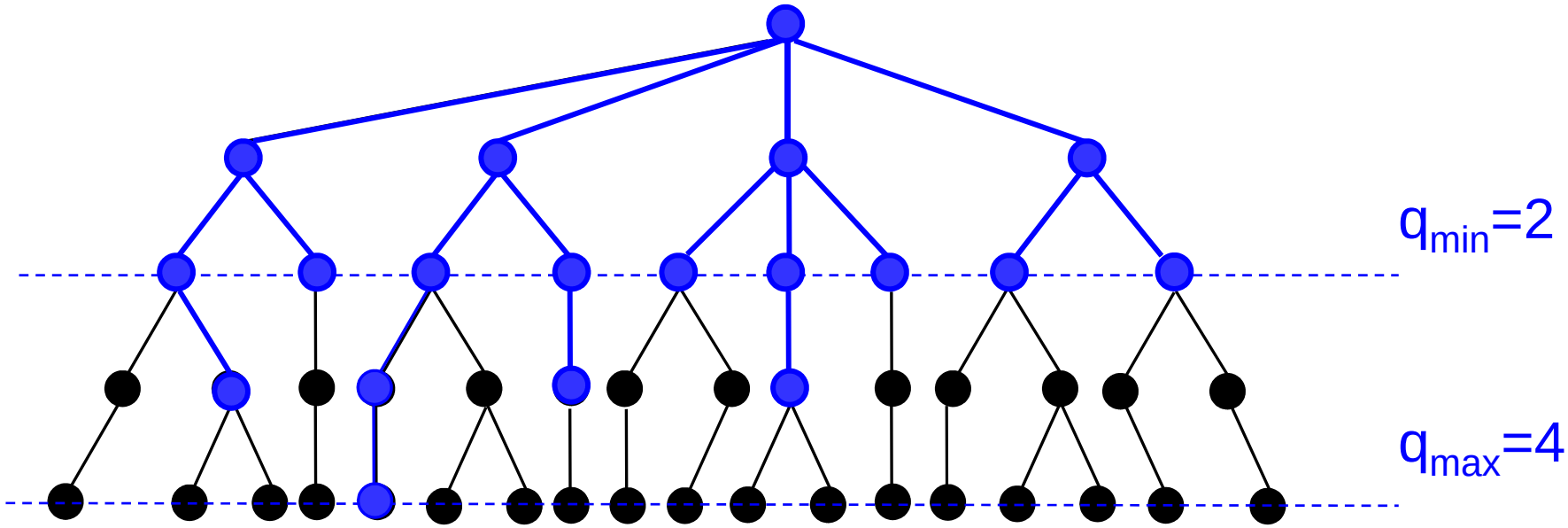
id	strings
1	bingo
2	bioinng
3	bitingin
4	biting
5	boing
6	going

Query Q	Dictionary	Gram set VG(Q)	Scanned list size	Candidates
bingon	D_0	{bi, in, ng, go, on}	19	1,2,3,4,6
	D_1 (+ ing)	{bi, ing, go, on}	11	1,3,4,6
	D_2 (+ bin)	{bin, ing, go, on}	8	1,6
bitting	D_0	{bi, it, tt, ti, in, ng}	21	3,4
	D_1 (+ ing)	{bi, it, tt, ti, ing }	13	3,4
	D_2 (+ bin)	{bi, it, tt, ti, ing }	12	1,3,4

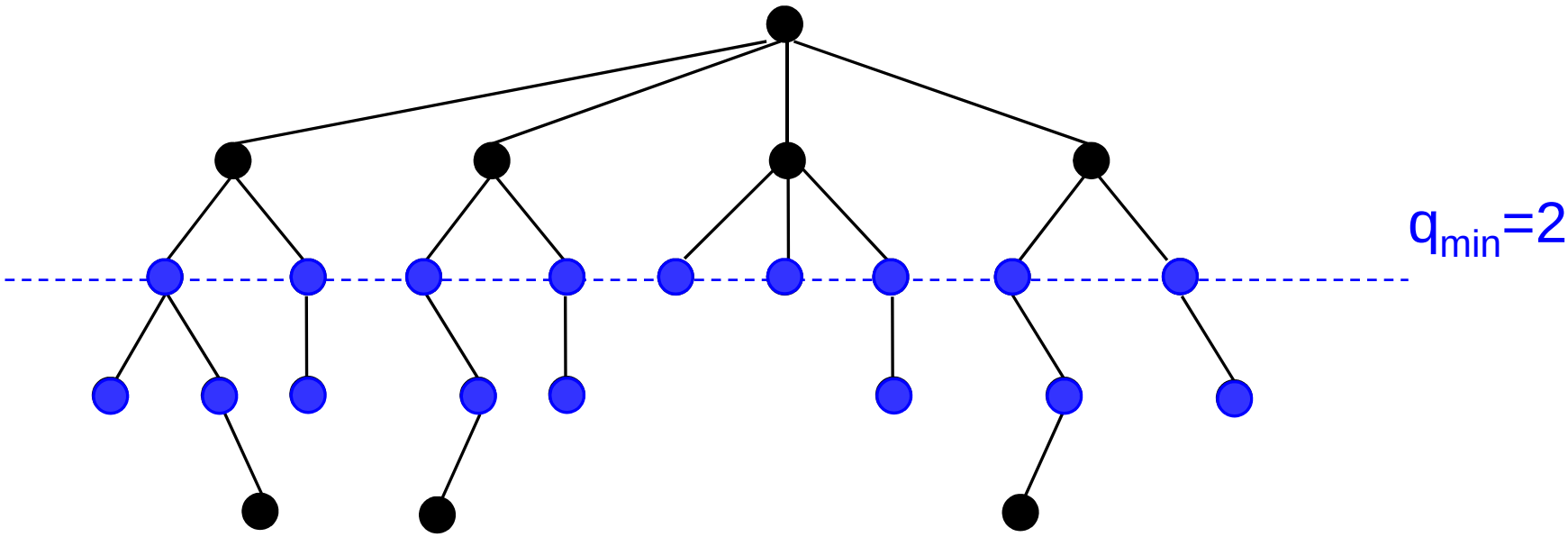
Outline

- Effects of adding a gram on index and queries
- →Cost-based construction of gram dictionary
- Experiments

Construct a gram dictionary [VLDB'07]

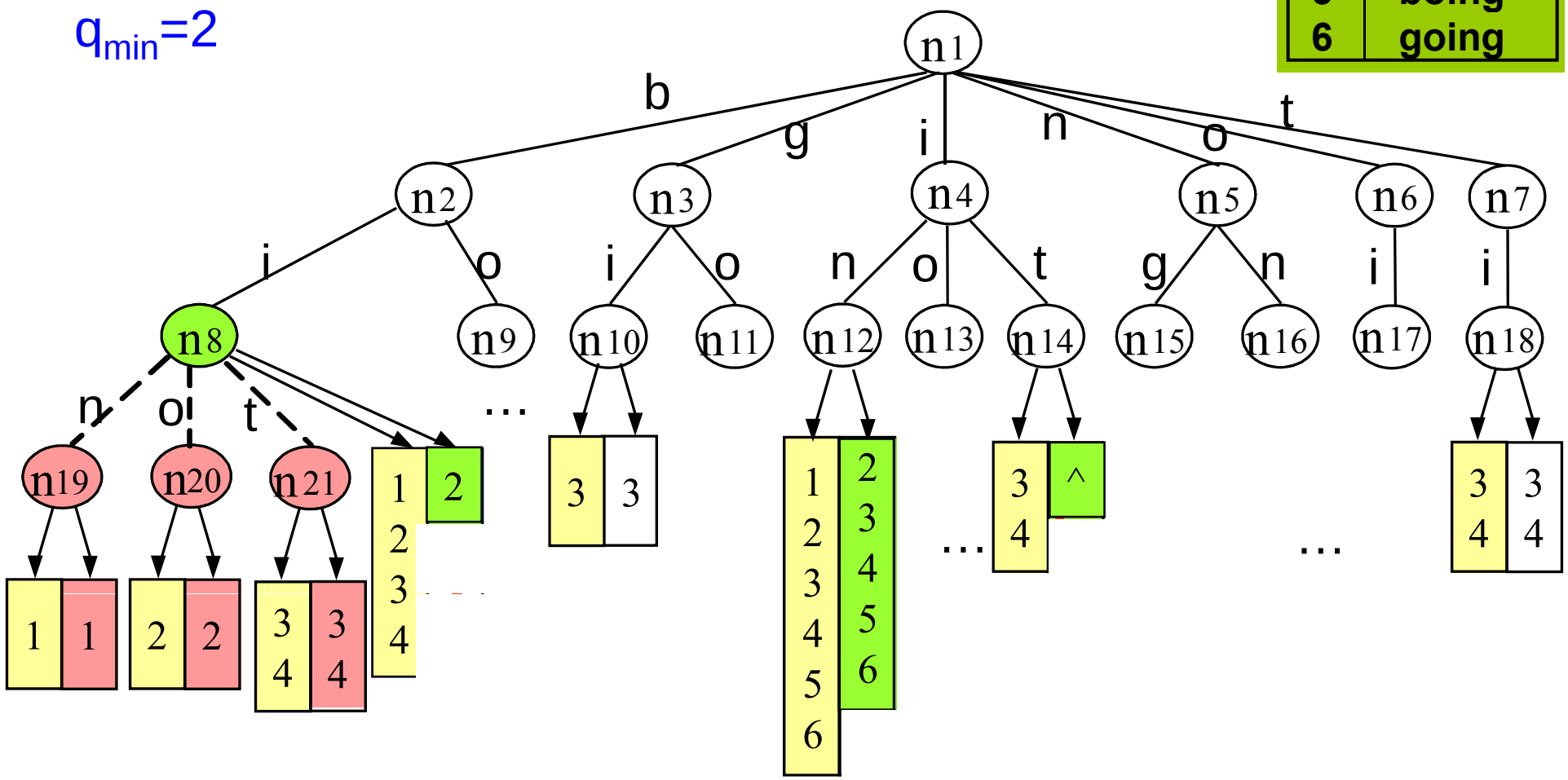


Cost-base construction [SIGMOD'08]



Construct a gram dict

id	strings
1	bingo
2	bioinng
3	bitingin
4	biting
5	boing
6	going



Experiments -- Data sets

Data set	String #	Length			Range of # of injected edit operations
		Min	Max	Avg	
Article Titles	277,000	6	207	66	[1,6]
Movie Titles	855,000	8	249	35	[1,3]
Actor Names	1,200,000	4	74	17	[1,2]

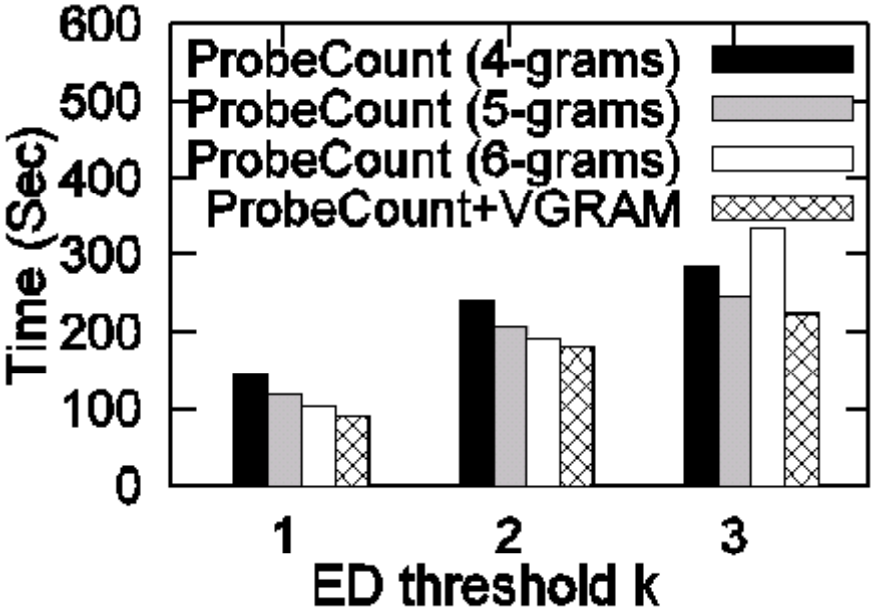
Environment:

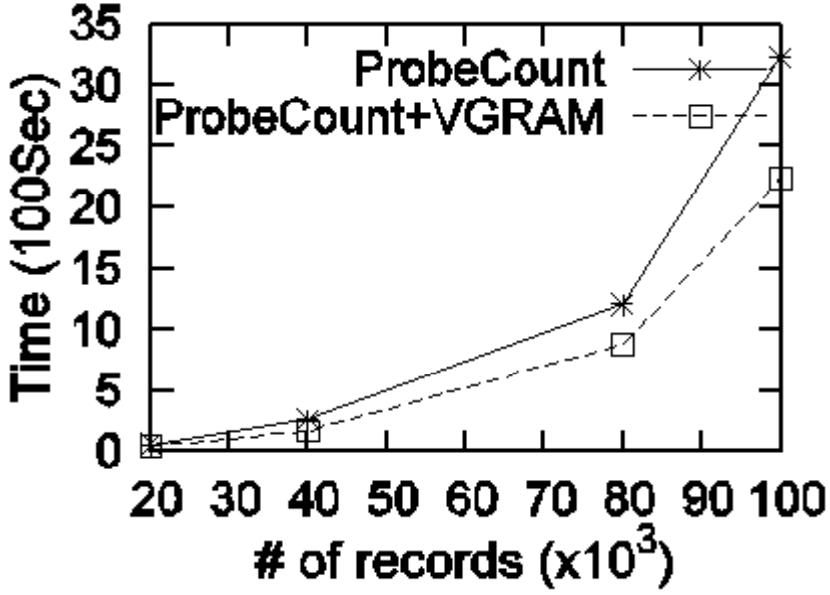
GNU C++, Dell GX620 PC with an Intel Pentium 2.40Hz Dual Core CPU, 2GB memory, 250GB disk, Ubuntu (Linux) O.S.

Index structure were assumed to be in memory

Improving algorithm ProbeCount

[VLDB'07]

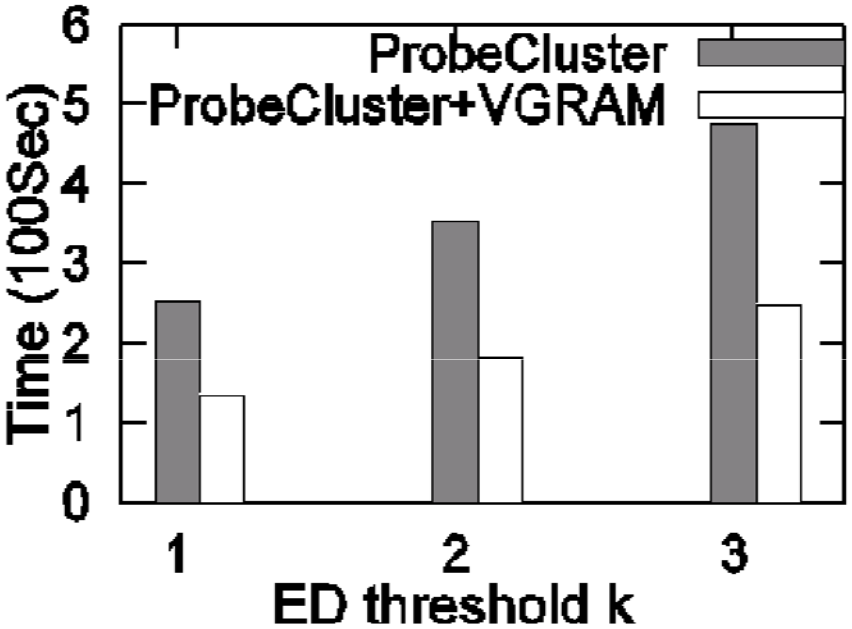


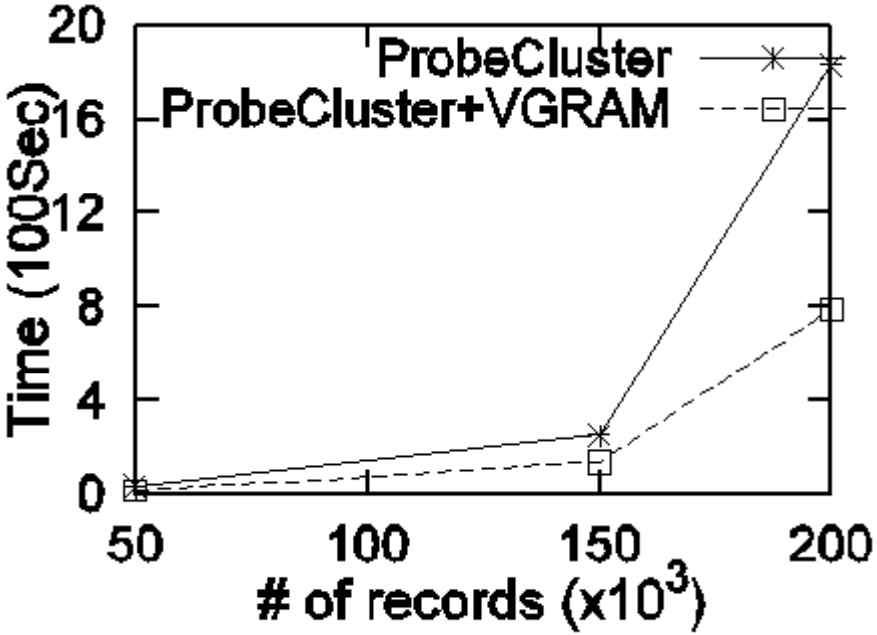


Dataset 1: Person name

Improving algorithm ProbeCluster

[VLDB'07]

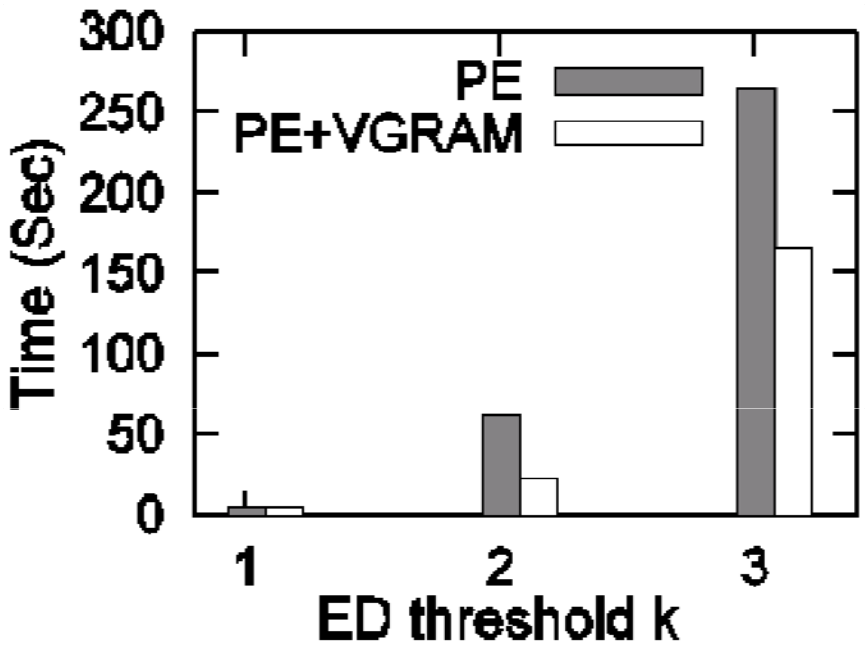


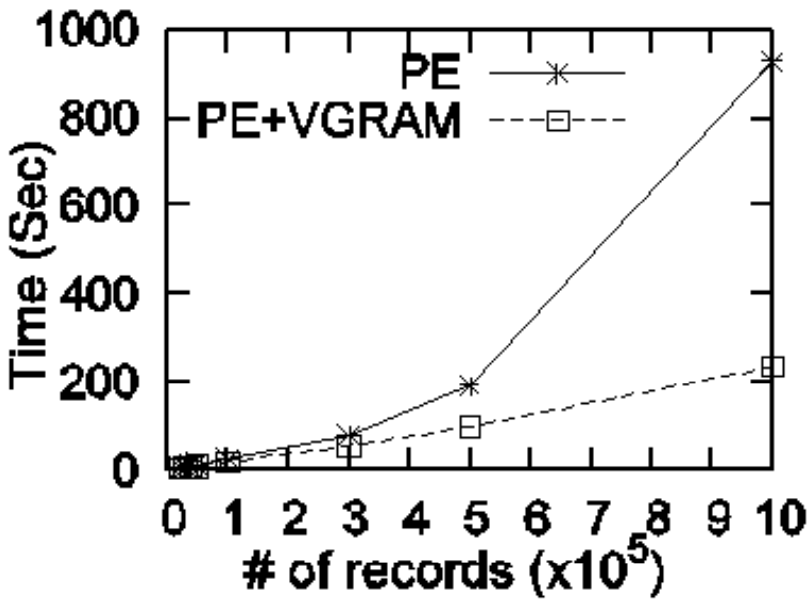


Dataset 1: Person name

Improving algorithm PartEnum

[VLDB'07]

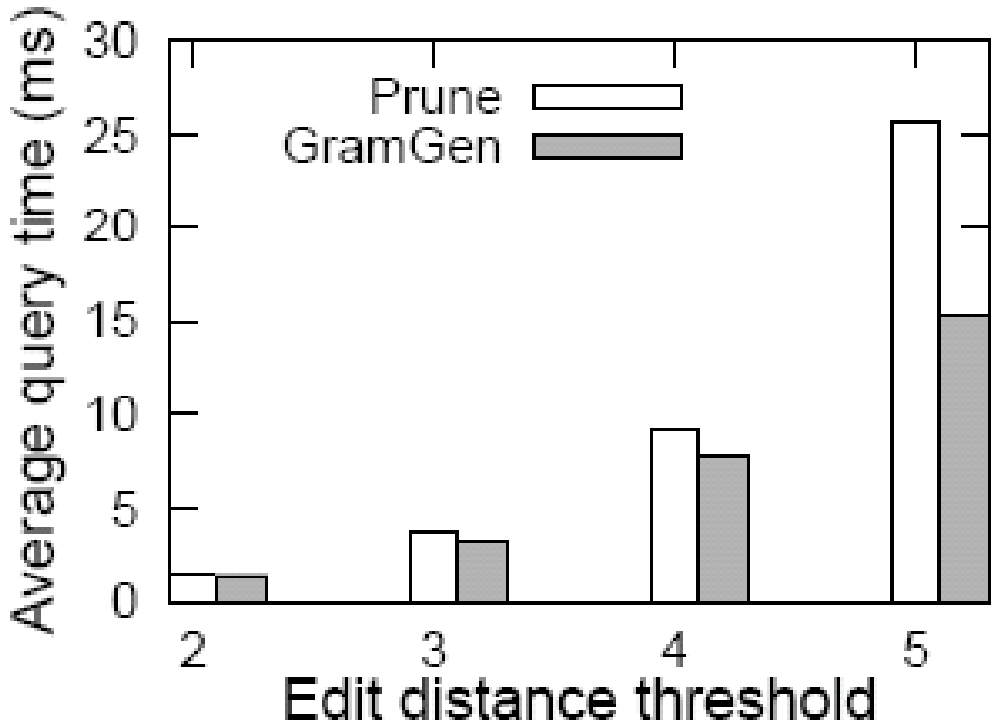




Dataset 1: Person name

Comparison with algorithm Prune

[SIGMOD'08]

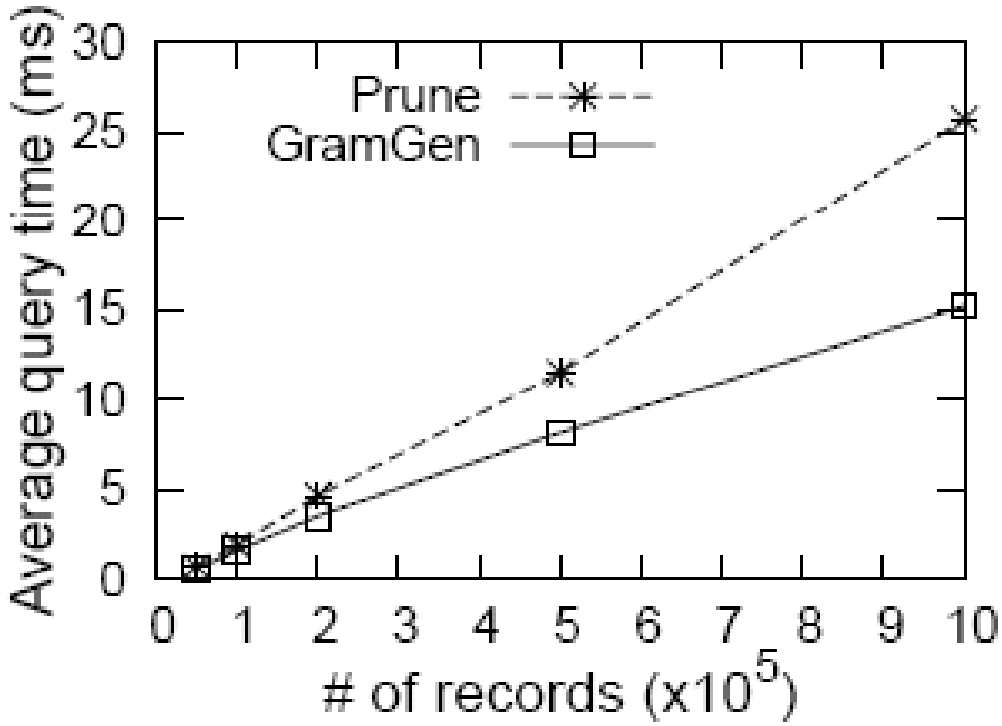


(b) Queries with different edit distance thresholds.

Dataset: 1M article titles
Prune: qmin=5, qmax=7, T=2000, LargeFirst policy
GramGen: 1% sampling ratio, 2000 queries, (qmin=5 automatically determined)

Comparison with algorithm Prune

[SIGMOD'08]



(a) Overall query performance.

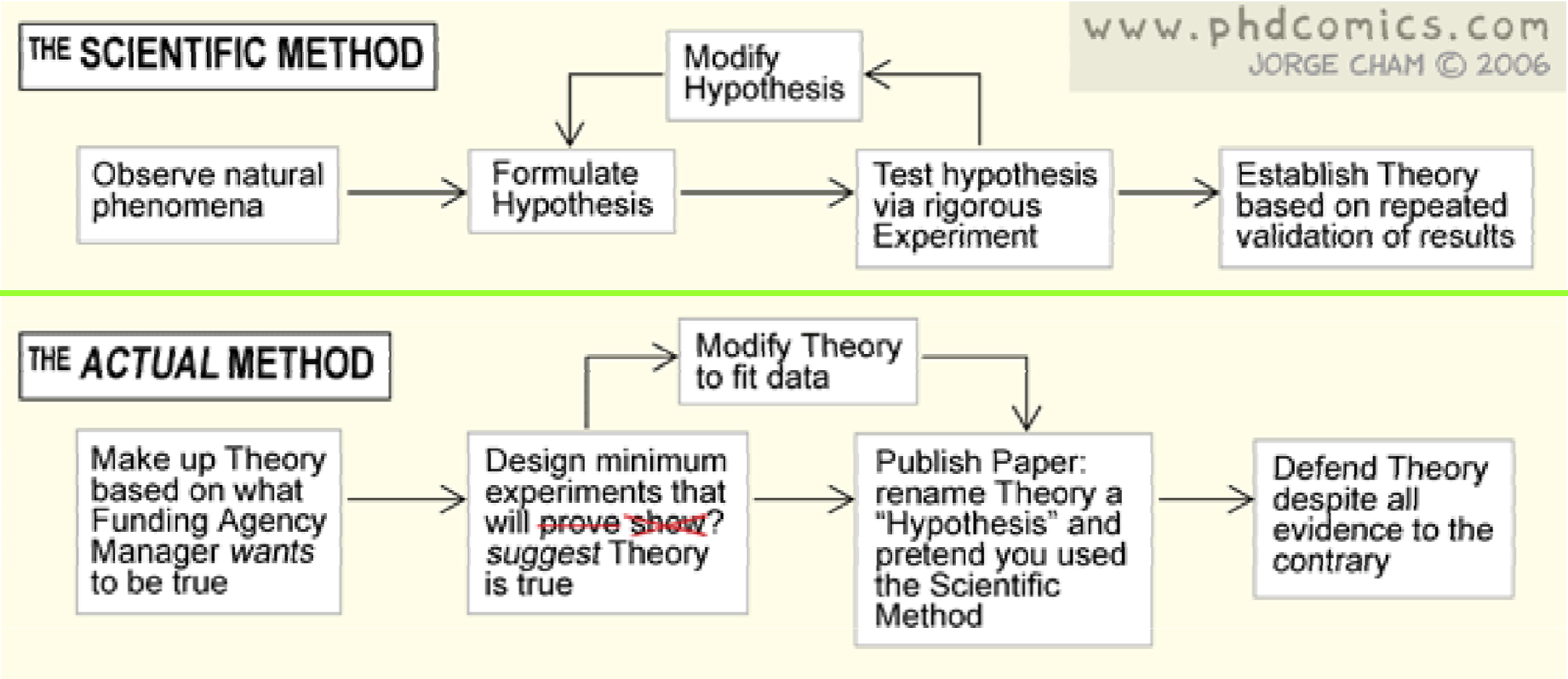
Prune: qmin=5, qmax=7, T=2000, LargeFirst policy
GramGen: 1% sampling ratio, 2000 queries, (qmin=5 automatically determined)

Conclusions

- VGRAM + gram selection
  - variable-length
  - high-quality
- Adoptable in existing algorithms
  - Reduce index size
  - Reduce running time

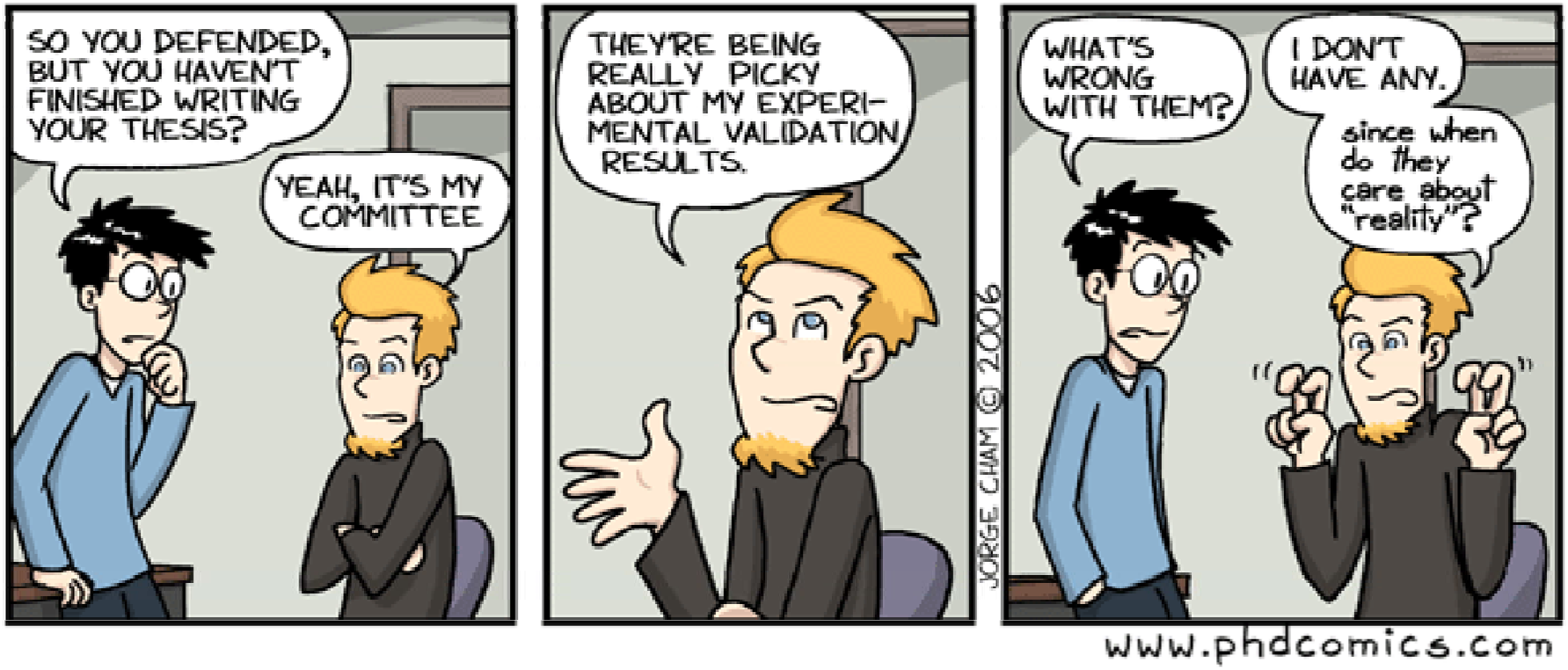
How to do research

- Use scientific method



How to do research

- Real application -- Make your approach useful

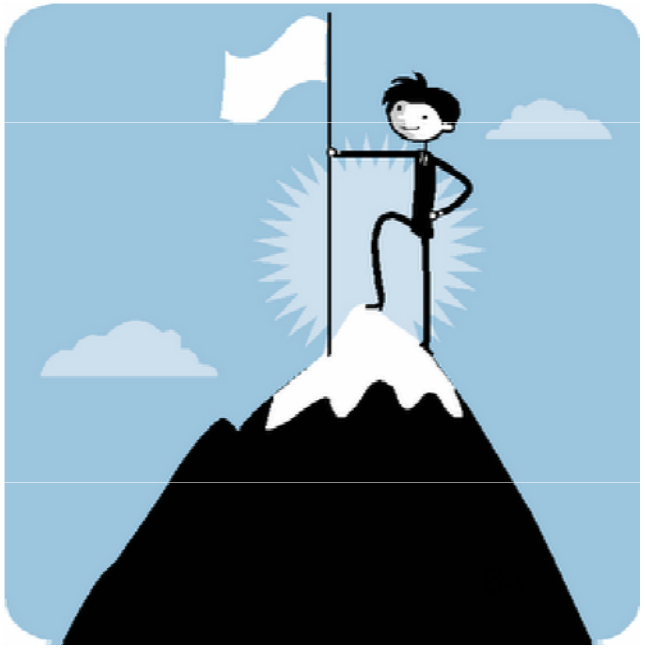


How to do research

Enjoy your work

Work hard

Make a difference



How to do research

- Focus on detail

- Be efficient

- **Beat the problem to death!**

Can we run faster?

Thank you