Extending Q-Grams to Estimate Selectivity of String Matching with Low Edit Distance

Hongrae Lee, Raymond Ng, Kyuseok Shim
(U. of British Columbia) (Seoul National U.)

Introduction

- Suppose a user wants to
  - List members in Vienna city
  - List branches where member Sylvie (?) works

Member	*City*	*Country*	*Branch*	…
Silvia	Vancouver	Cana[illegible]	[illegible]	
Silvie	Viena ⊘	[illegible]	[illegible]	
Sylvie	Vienna	Austria	Liesing	
…	…	…	…	

1. Typos in the database

2. Similar names or Different spelling usage

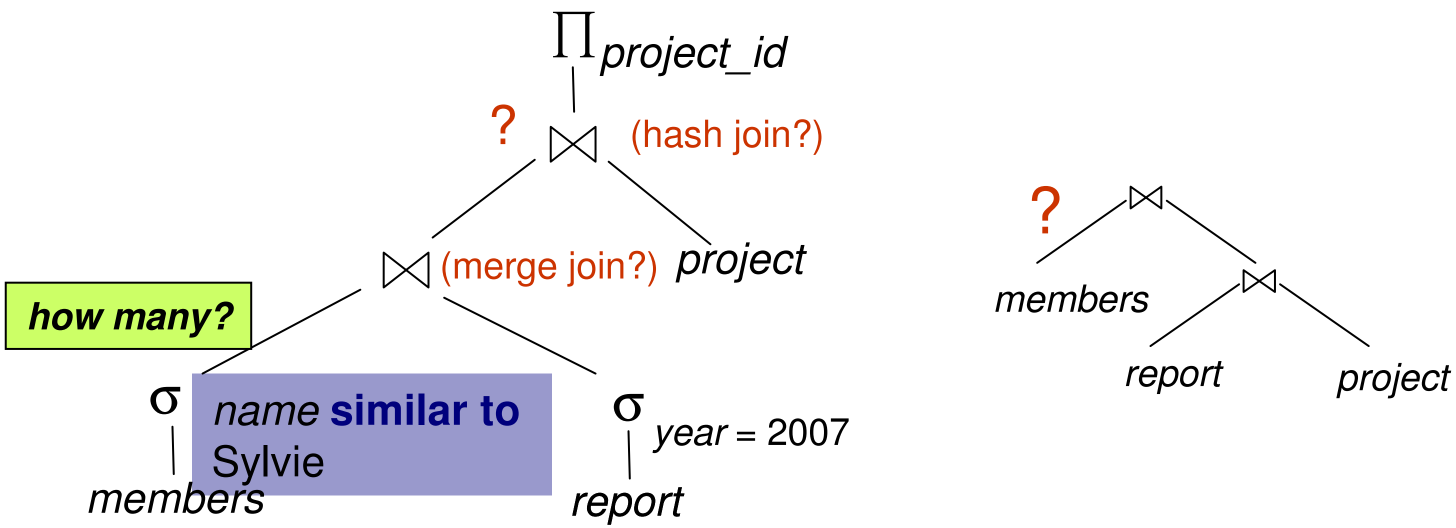
Introduction (cont.)

- Approximate string matching queries
  - Find cities **similar to** *Vienna*
  - Find names **similar to** *Sylvie*

- Approximate string matching is important in
  - Data cleaning, data integration
    - Pervasive errors or heterogeneity in the database
  - Searching
    - Uncertain query formulation (query correction)
    - Different spelling usages

Query Optimization of Approximate String Matching

- Optimization of approximate query processing
  - Join ordering, access method selection,…



- Estimating selectivity of approximate predicates
  - Important in making a good query execution plan

How Do We Define "Similar"?

- String similarity functions
  - Edit distance, Hamming distance, Jaccard coefficient,…
- Edit distance
  - The minimum # of edit operations (Insert, Delete, Replace) to convert one string to the other

ed (Vienna, Wiena) = 2

- Focus on low edit distance k, say k=1 ~ 3 or 4,5
  - Low edit distance offers a lot to database applications
    - E.g., [AGK06](data cleaning) employed k=1 ~ 3 for address
  - High edit distance can be error prone
    - E.g., Even k=2: Vienna → Vietnam

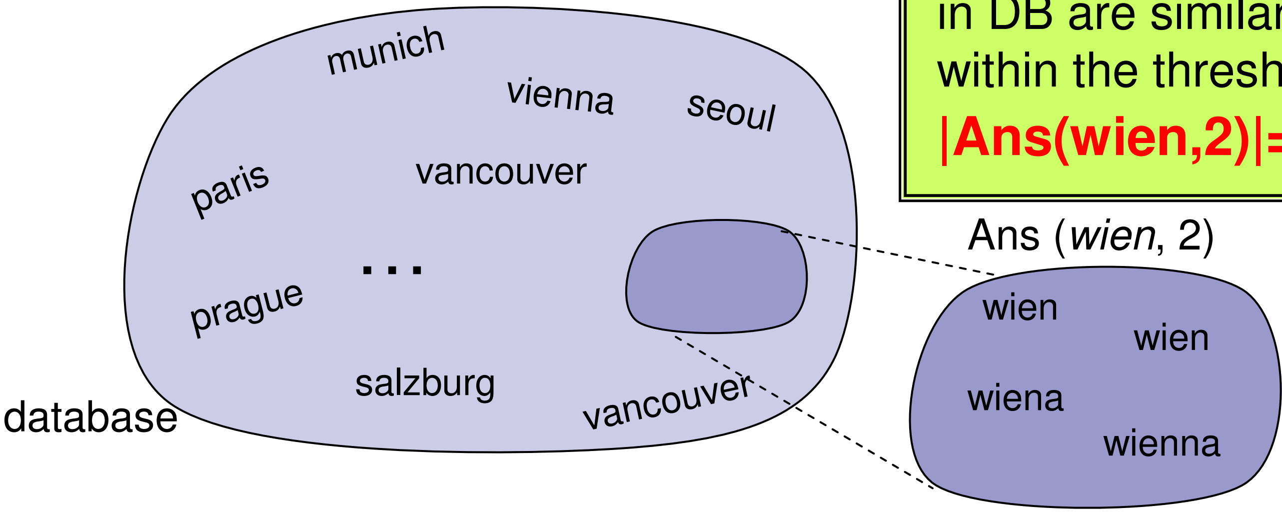
Problem Statement

- Given a query string s_q and an edit distance threshold *k*, estimate *the # of strings s* in the database that satisfy $ed(s_q, s) \leq k$.

Query ≡ (*wien*, 2)

How many strings in DB are similar to *wien* within the threshold *k*?

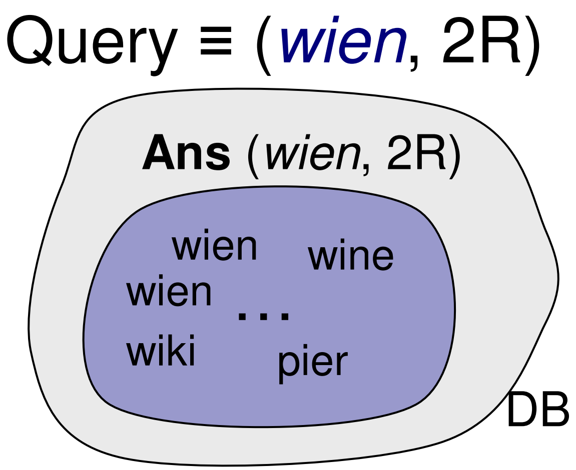
|Ans(wien,2)|=?



Overview

- Introduction
- Contributions
  - **Formulas for special cases**
    - **Replace only case**
    - **Delete only case**
    - **Insert only case**
  - Algorithm BasicEQ
  - Optimizations
  - Extended Q-grams
- Empirical evaluation
- Conclusion & future works

Replace Only Case



- Start with a restricted version of the problem
  - Only allow replace
- Want to estimate |Ans|
  - The # of strings in the DB that can be converted to *wien* with at most 2 replaces

Representing A Replace with ?

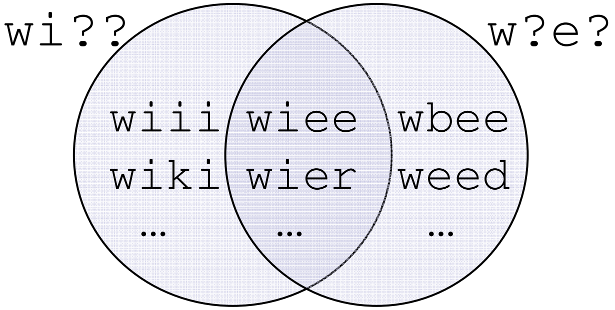
Strings in Ans (wien, 2R) can be acquired by replacing up to 2 characters from *wien*

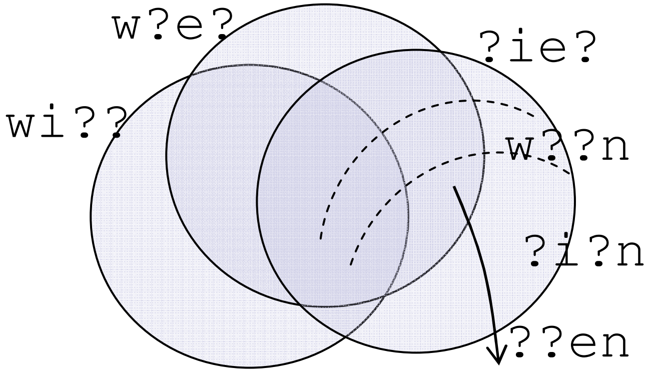
`wien` → `wi??`, `w?e?`, `?ie?`, `w??n`, `?i?n`, `??en`

$\binom{4}{2}$ = 6 possible cases

- The wildcard ? represents a replacement (or an insertion)
- Any string in the Ans is in at least one of the above 6 forms
  - E.g., `wiki` ⊂ `wi??`
    `teen` ⊂ `??en`
- |Ans(wien, 2R)| = # of strings in any of the 6 forms

Finding |Ans(*wien*, 2R)|



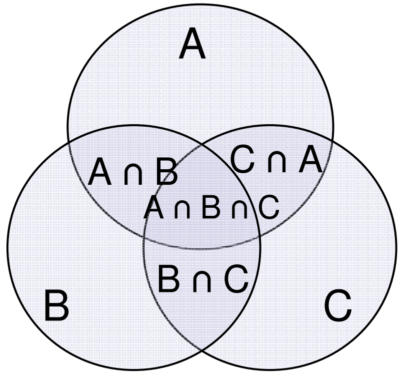


- Note that there are overlaps among the sets
  - E.g., `wi?? ∩ w?e? = wie?`
- The desired answer is

> |Ans(wien,2R)| =
> | wi?? ∪ w?e? ∪ ?ie? ∪ w??n ∪ ? i?n ∪ ??en |

Inclusion-Exclusion Principle

- Inclusion-Exclusion principle
  - The size of union of *n* sets is the sum of sizes of all possible intersections among *r* elements with sign of $(-1)^{r+1}, 1 \le r \le n$
  - E.g., |A ∪ B ∪ C|



$$= |A| + |B| + |C| - (|A \cap B| + |B \cap C| + |C \cap A|) + |A \cap B \cap C|$$

- |Ans(wien,2R)| =

| wi?? ∪ w?e? ∪ ?ie? ∪ w??n ∪ ?i?n ∪ ??en |

= |wi??|

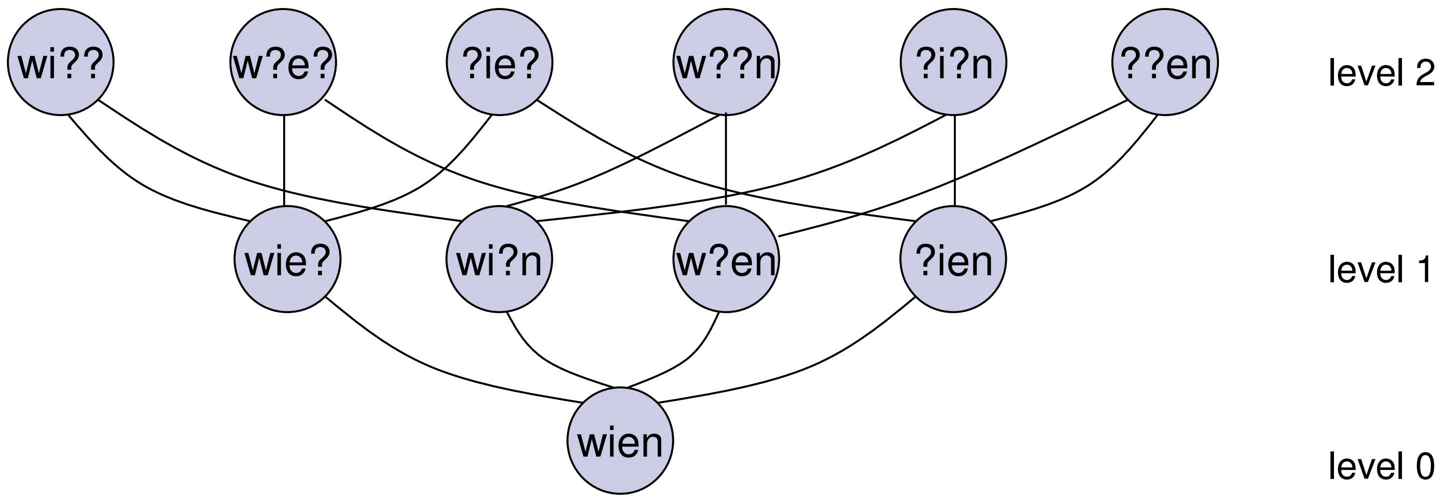
−(|wi?? ∩

+(|wi?? ∩

−(|wi?? ∩ w?e? ∩ … ∩ ??en|)

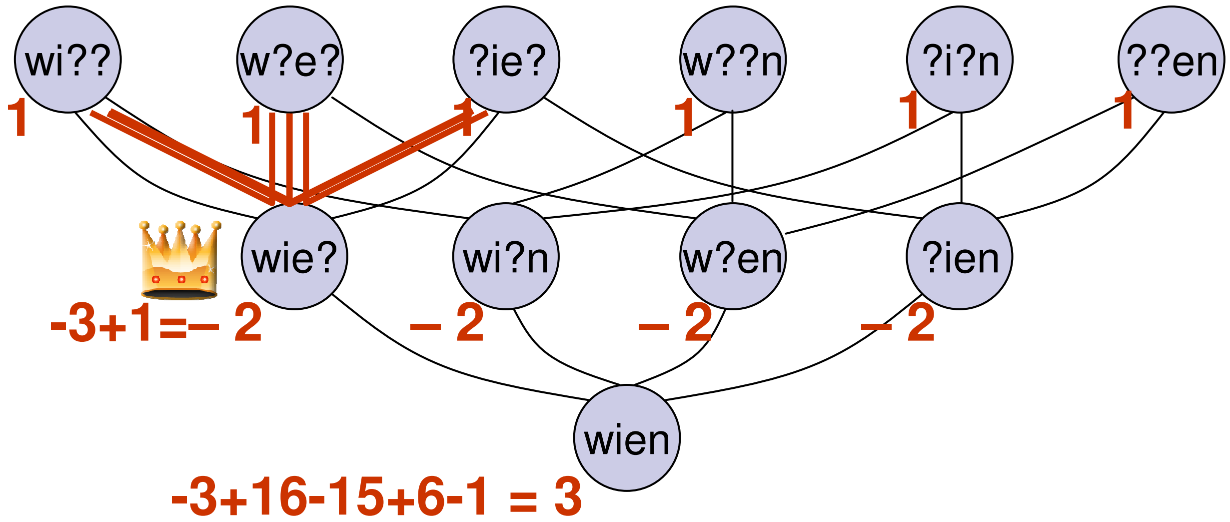
Exponential # of
- computing intersections (character level)
  e.g., `wi?? ∩ w?e? = wie?`
- getting frequency from the summary structure
  e.g., `|wie?|=` ?

Solution: Using A Semi-Lattice



- A Node represents the set of strings in DB in that form
- Start with leaf nodes of all possible 6 forms
- Generate nodes from intersections
- Layer nodes according to the # of wildcards (level)
- Draw edges for inclusion relationship

Using A Semi-Lattice (cont.)



- | wi?? ∪ w?e? ∪ ?ie? ∪ w??n ∪ ?i?n ∪ ?? en|

  = |wi??| + |w?e?| + … + |??en|

  − (|wie?| + |wie?| + |wie?| + …)

  \+ (|wie?| + …)

  …

  − | wi?? ∩ w?e? ∩ … ∩ ??en|

  − 3|wie?| +1 |wie?| = − 2|wie?|

Using A Semi-Lattice (cont.)

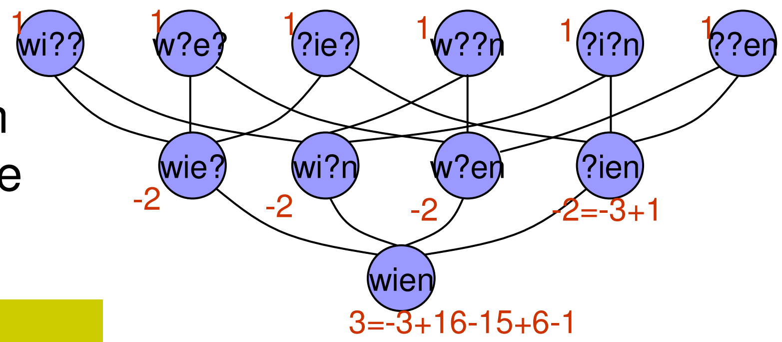
- Key observations
  - Many intersections may result in the same node
  - Regularity exists in the semi-lattice structure
- Key approach
  - Substitute an intersection with its result
  - Only need to count how many times a node participates in the I-E (inclusion-exclusion) formula
  - The coefficient of a node
    - # of times a node participates in the I-E formula
    - Minus sign if appears more in minus part in the I-E formula

Using A Semi-Lattice (cont.)

Original Inclusion-Exclusion process

| wi?? ∪ w?e? ∪ ?ie? ∪ w??n ∪ ?i?n ∪ ??en|
= |wi??| + |w?e?| + … + |??en|
− (|wi?? ∩ w?e?| + |wi?? ∩ ?ie?| + |w?e? ∩ ?ie?| + …)
+ (|wi?? ∩ w?e? ∩ ?ie?| + …)
…
− | wi?? ∩ w?e? ∩ … ∩ ??ne|

Simplify the equation
Using the semi-lattice



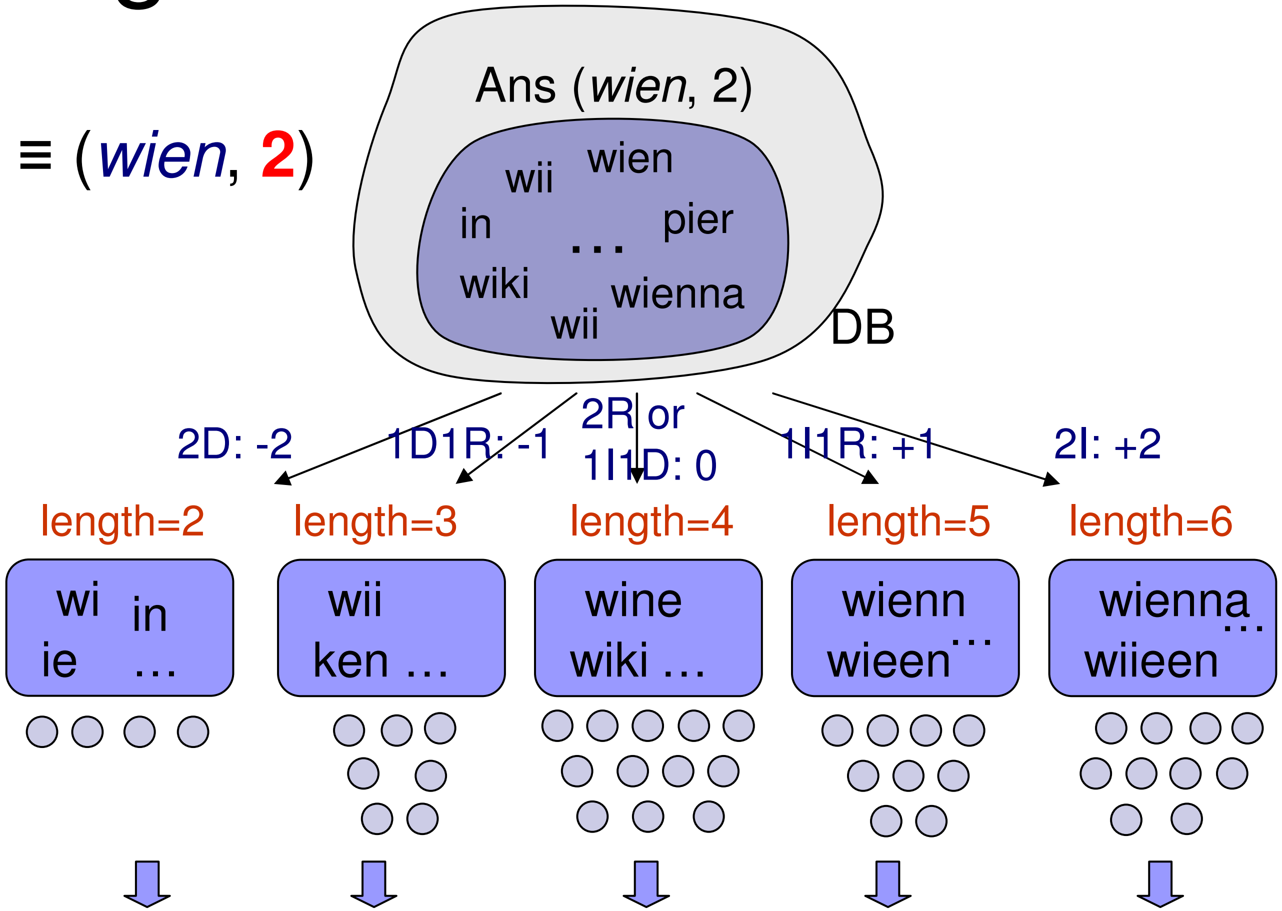
= |wi??| + |w?e?| + … + |??ne|
+ (– 3 + 1) (|wie?| + |wi?n| + |w?en| + |?ien|)
+ (– 3 + 16 – 15 + 6 – 1) |wien|

Overview

- Introduction
- Contributions
  - Formulas for special cases
  - **BasicEQ Algorithm**
  - Optimizations
  - Extended Q-grams
- Empirical evaluation
- Conclusion & future works

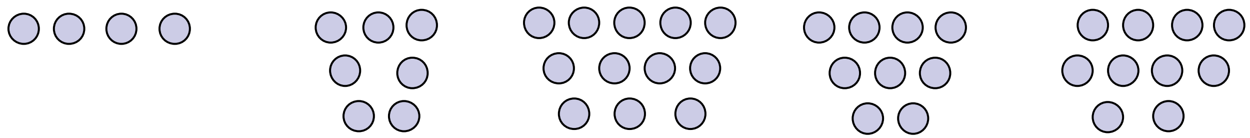
The BasicEQ Algorithm: Returning to the General Problem

Query ≡ (*wien*, **2**)

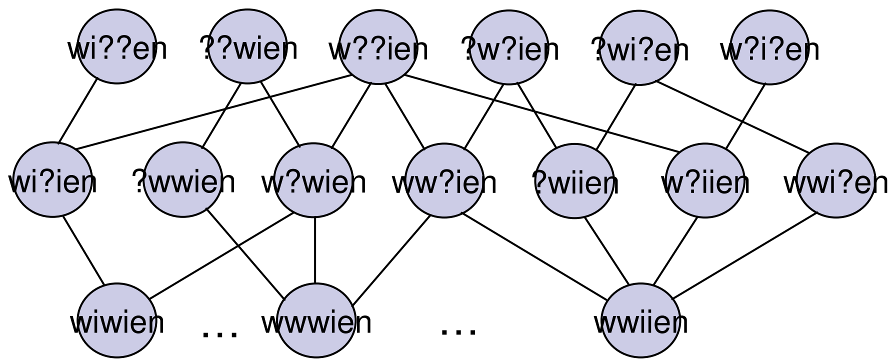


$|\mathbf{Ans(wien,2)}| = freq_{len=2} + freq_{len=3} + freq_{len=4} + freq_{len=5} + freq_{len=6}$

String Hierarchies



Do not have the formulas for all string hierarchies!
E.g.) **1I1R, 2I1D + 1I2R**



An example of general string hierarchy

- General string hierarchy: not so regular (closed form fomular is hard)
- Need a general algorithm to handle arbitrary combinations of edit operations. e.g.)1I1R

Computing Frequency from A String Hierarchy

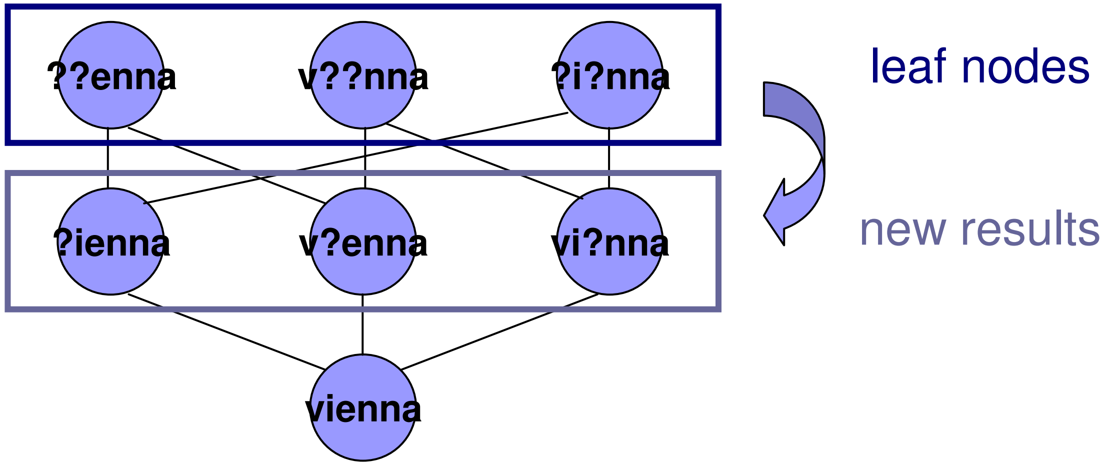
Answer set cardinality = sum of the frequencies of nodes multiplied by the coefficients

Key steps

1. Build the string hierarchy
2. Compute the coefficients of nodes
3. Estimate selectivity of each node and compute the simplified inclusion-exclusion formula

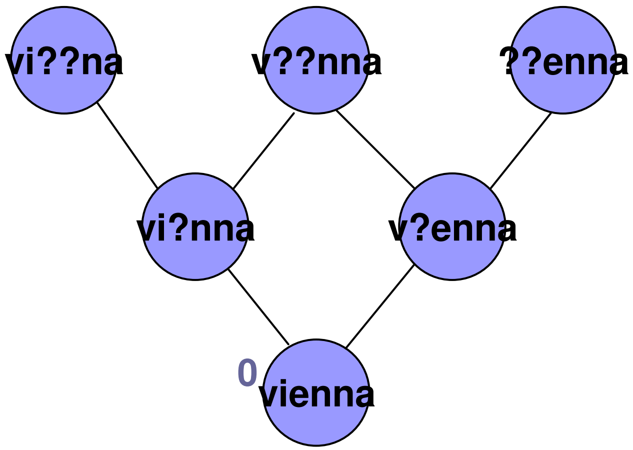
BasicEQ
Step 1: Building The String Hierarchy

- An Apriori-Style algorithm
  - Start from leaf nodes
  - Generate an intersection of r nodes by extending intersection of (r-1) nodes
  - Two observations are crucial
    - **Only *newly formed results*** need to be considered at each round
    - **Only** the nodes with ***at least one wildcard*** need to be considered



BasicEQ
Step 2: Computing Coefficients of Nodes

- For each node, add the number of intersections among r nodes that result in that node with the sign of $(-1)^{r+1}$



# of 2-intersection results in vienna:1→ −1
# of 3-intersection results in vienna:1→ +1
The coefficient of vienna → −1+1=0

Overview

- Introduction
- Contributions
  - Formulas for special cases
  - Algorithm BasicEQ
  - **Optimizations**
  - Extended Q-grams
- Empirical evaluation
- Conclusion & future works

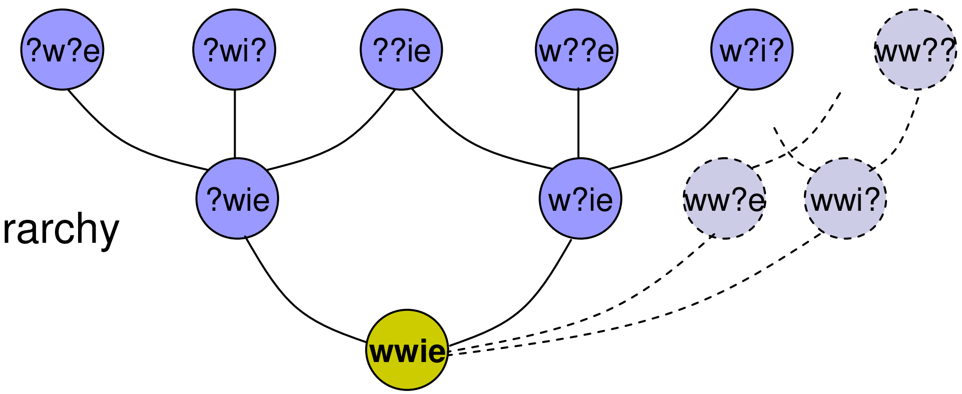
Three Optimizations

- BasicEQ is not scalable
  - Coefficient computation step is a major bottleneck

1. Node partitioning
   - Compute coefficients just once for each partition
2. Coefficient approximation
   - Use replace-only formula to approximate coefficients
3. Fast intersection test by grouping
   - Avoid test of intersections that are guaranteed to produce the empty result

Coefficient Approximation

- Approximate coefficients using the replace-only formula
  - Motivation is that we have a formula for coefficients



Part of the string hierarchy for Ans(wie,1I1R)

  - Complete the lattice to the full replacement lattice
  - Scale terms in the formula assuming everything is proportional to the possible choices

Overview

- Introduction
- Contributions
  - Formulas for special cases
  - Algorithm BasicEQ
  - Optimizations
  - **Extended Q-grams**
- Empirical evaluation
- Conclusion & future works

Estimating Selectivity of Each Node

|Ans(wien,2R)| =
1 (|wi??| + … + |??ne|) – 2 (|wie?| + |wi?n| + |w?en| + |?ien|) + 3 |wien|

|wien|=freq(wien)=# of *wien* in the database

- Q-grams
  - Any string of length q in ∑
  - *vienna* → 3-grams: *vie, ien, enn, nna*
- Q-gram table [Chaudhuri, Ganti & Gravano 04]
  - N-grams of length q or less
  - with their frequency

Q-gram	Frequency
wien	**9**
wie	12
ien	10
ein	56
ei	1,205
e	24,503
…	…

Extended Q-Gram Table

- Extended q-grams
  - *Extend* q-gram with wildcard ? (not in ∑)
  - Speed up the frequency computation of string forms
    - Example using just simple q-gram tables
      - |wie?| = |wiea| + |wieb| + |wiec| + ….

Q-gram	Frequency
wien	9
wie?	**89**
wiea	1
ien	10
i??	**4,213**
…	…

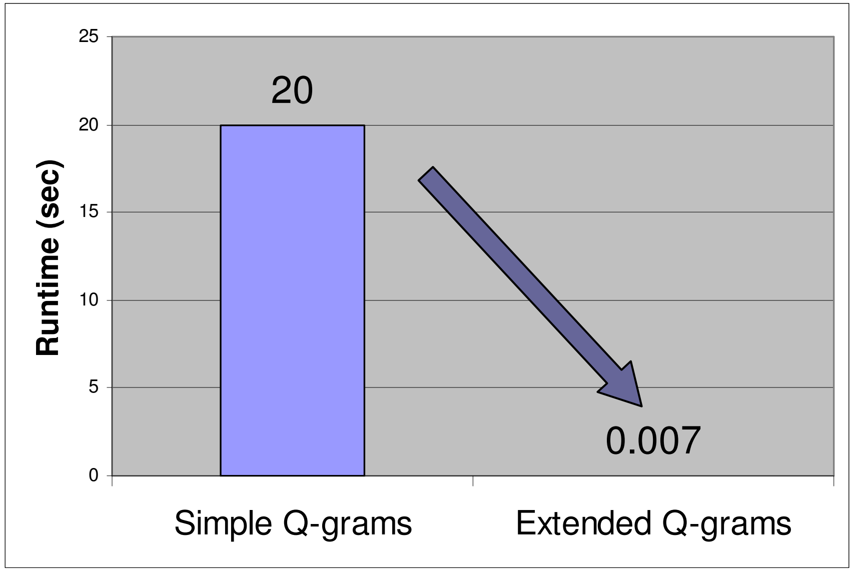
Overview

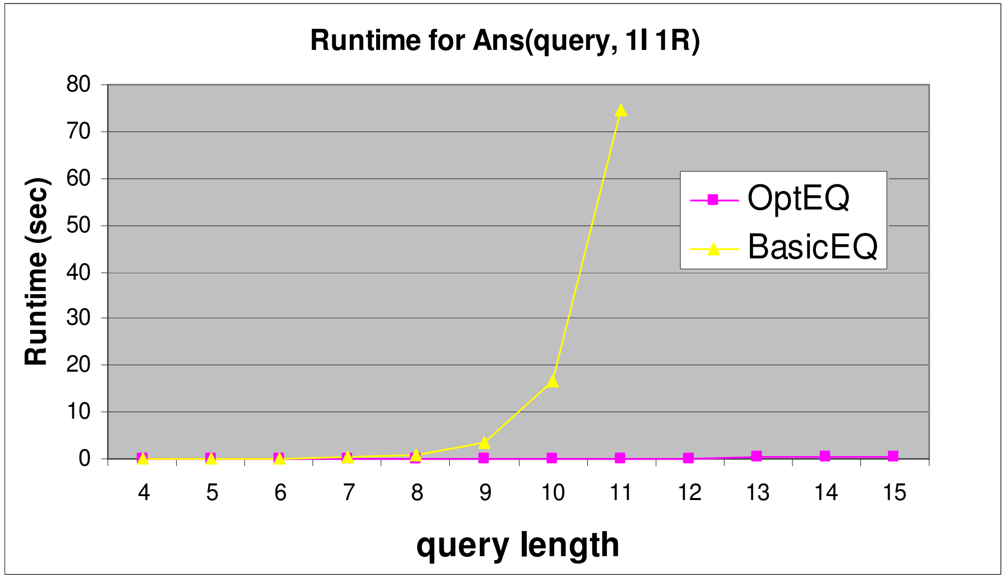
- Introduction
- Contributions
  - Formulas for special cases
  - Algorithm BasicEQ
  - 3 Optimizations
  - Extended Q-grams
- **Empirical evaluation**
  - **Settings**
  - **Effectiveness of optimizations**
  - **Estimation accuracy**
- Conclusion & future works

Empirical Evaluation

- Data set
  - 392,132 IMDB actresses' last names
  - 699,198 DBLP Authors full names
  - 53,365 DBLP Paper titles
- Compared technique
  - SEPIA [Jin & Li 05]
- Settings
  - SEPIA: 2000 clusters, 5% sampling
  - OptEQ: BasicEQ + optimizations
  - Coefficients are pre-computed (not data dependent)
  - Intel P4 3GHz PC with 1 GB Memory

Effectiveness of Optimizations



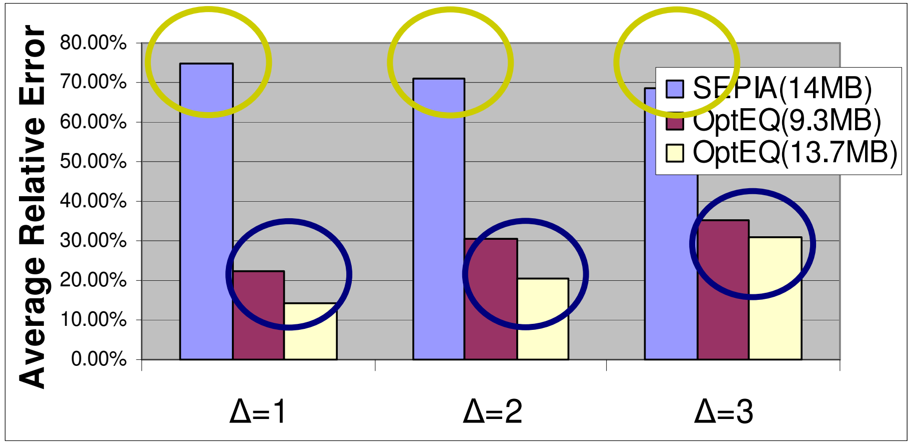


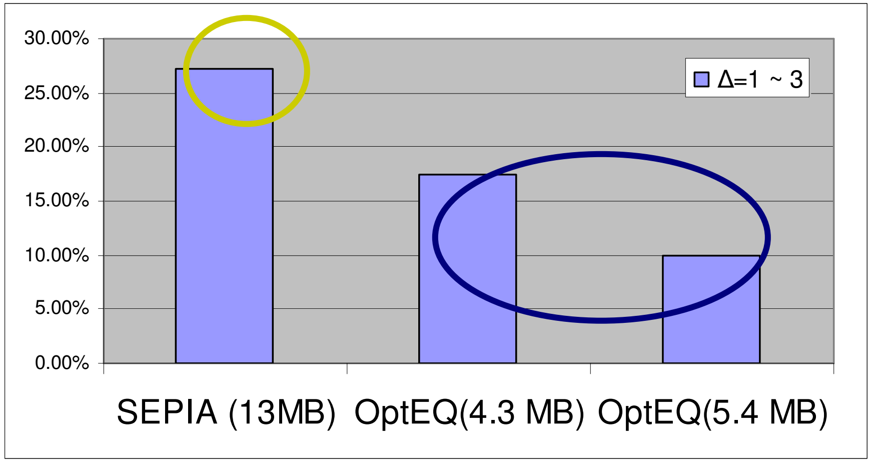
Extended q-gram vs. simple q-gram

BasicEQ vs. OptEQ

- Extended q-grams enable faster computation
- OptEQ's optimizations improve the performance of BasicEQ by orders of magnitudes

Estimation Accuracy





DBLP Author names

DBLP Paper titles

- Relative error: $|freq_{est} - freq_{real}|/freq_{real}$
- OptEQ delivers more accurate estimation
- OptEQ is able to utilize additional space showing clear trade-off between space and accuracy

Other Experimental Results

- Error distribution characteristics
- Scalability
- Higher edit distance threshold with sampling

- See the paper for details

Related Work

- Substring selectivity estimation
  - Exact string match
  - MO [Jagadish, Ng & Srivastava 99]
  - CRT [Chaudhuri, Ganti & Gravano 04]
- Approximate string selectivity estimation
  - SEPIA [Jin & Li 05]

Conclusion

- Contribution
  - Extended q-grams with the wildcard
  - New lattice-based algorithm for estimating selectivity of approximate string matching
  - Performance study shows that OptEQ delivers accurate selectivity estimation
- Future work
  - Handling longer string with higher edit distance threshold as in genomic applications

Any Questions?

Danke schön!

Node Partitioning

- Coefficients only depend on the lattice structure
- We *partition* nodes according to the local lattice structure to each node and *compute the coefficients just once per each partition*
  - Approximate isomorphism test is developed