Network Science : Lecture IX

Graph Query

Computer Science Department
Data Mining Research

Dec 2, 2014

Graph Queries

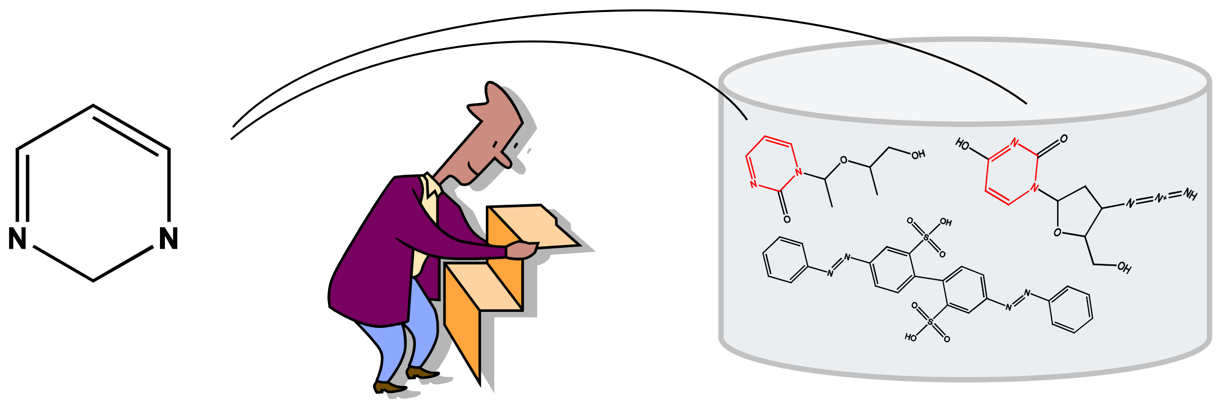
- Containment Query
  Retrieves all graphs from a graph database, such that they **contain** a given query graph (exact and approximate).

- Similarity Query
  Retrieves all graphs from a graph database, that are **similar** to the query graph (exact and approximate).

- Matching Query
  Find all **occurrences** of a query graph in a large target network (exact and approximate).

Containment Query

Find all of the graphs in a database that contain the query graph

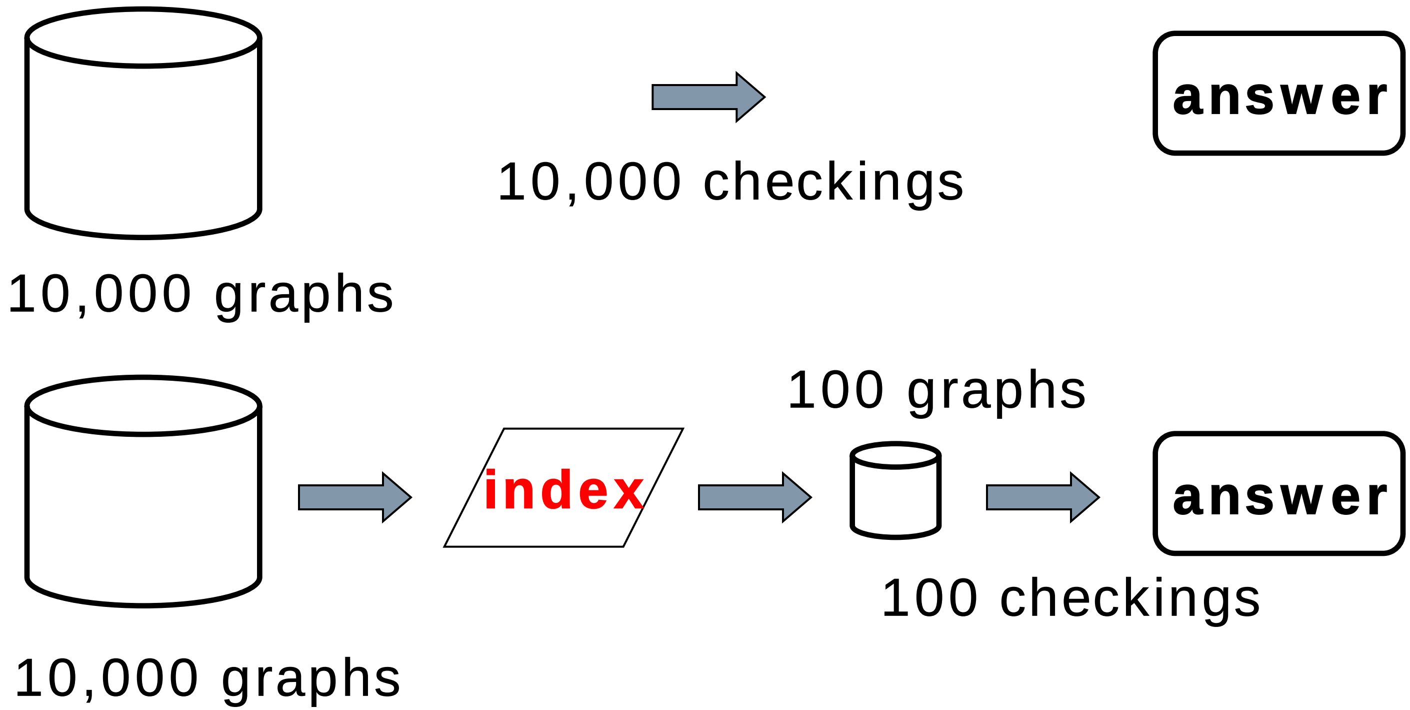


query graph

graph database

Indexing Graphs

- Indexing is crucial



Filtering and Verification

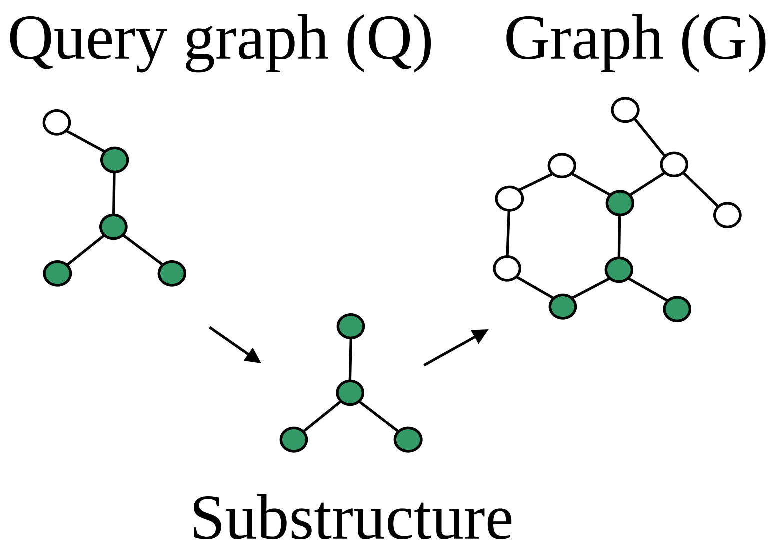
- Subgraph Isomorphism Problem is **NP-hard**.

Filtering and Verification

- **Filtering Phase:**
  Feature-based index is used to filter out the negative results and generate a candidate sets.

- **Verification Phase:**
  Precise Subgraph Isomorphism Testing to generate final results from the candidate set.

Indexing Strategy



If graph G contains query graph Q, G should contain any substructure of Q

Index substructures of a query graph to prune graphs that do not contain all of these substructures

Indexing Framework

- Two steps in processing graph queries

Step 1. Index Construction

- Enumerate **structures** in the graph database, build an inverted index between structures and graphs

Step 2. Query Processing

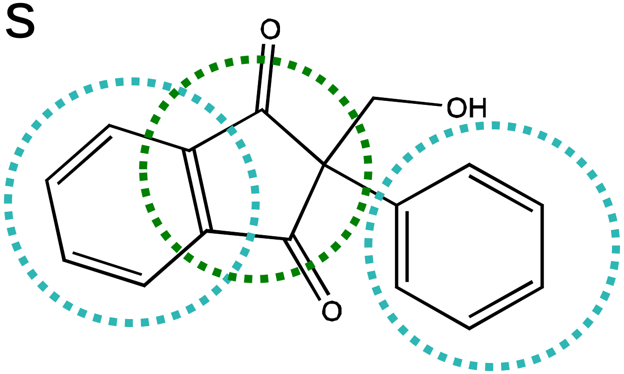
- Enumerate **structures** in the query graph
- Calculate the candidate graphs containing these structures
- Prune the false positive answers by performing subgraph isomorphism test

Feature-based Index

Question: What kind of substructures to index?

Options:

1. Node/edge labels
2. All of the substructures
3. Paths (Shasha et al. PODS'02)
4. Frequent graphs
5. Discriminative frequent graphs (Yan et al. SIGMOD'04)
6. Trees



Cost Analysis

QUERY RESPONSE TIME

$$T_{index} + \boxed{|C_q|} \times \left(T_{io} + T_{isomorphism\_testing}\right)$$

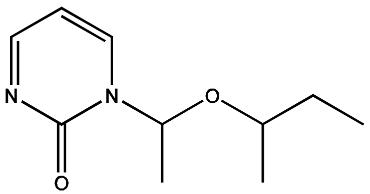
fetch index

number of candidates

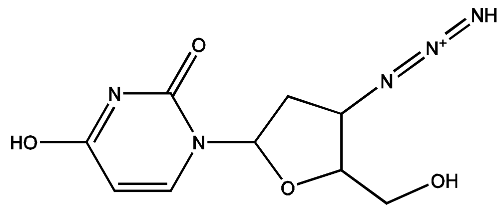
REMARK: make $|C_q|$ as small as possible

Path-based Approach

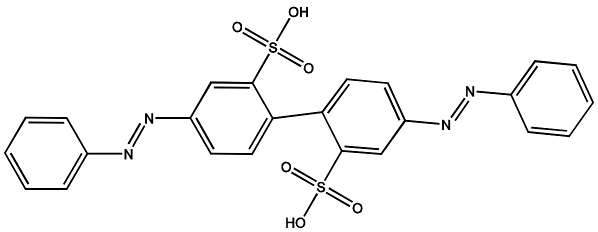
GRAPH DATABASE



(a)



(b)



(c)

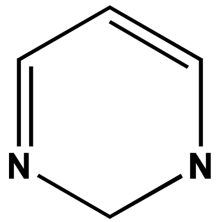
PATHS

0-length: C, O, N, S
1-length: C-C, C-O, C-N, C-S, N-N, S-O
2-length: C-C-C, C-O-C, C-N-C, ...
3-length: ...

Built an inverted index between paths and graphs

Path-based Approach (cont.)

QUERY GRAPH



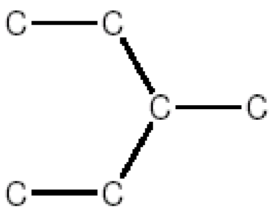
0-edge: S_C={a, b, c}, S_N={a, b, c}
1-edge: $S_{C\text{-}C}$={a, b, c}, $S_{C\text{-}N}$={a, b, c}
2-edge: $S_{C\text{-}N\text{-}C}$ = {a, b}, …
…

Intersect these sets, we obtain the candidate answers - graph (a) and graph (b) - which may contain this query graph.

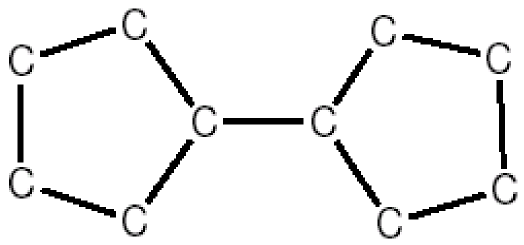
Problems: Path-based Approach

GRAPH DATABASE

(a)

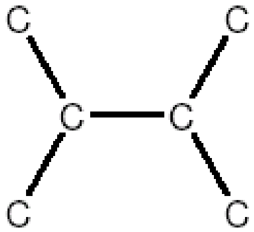


(b)



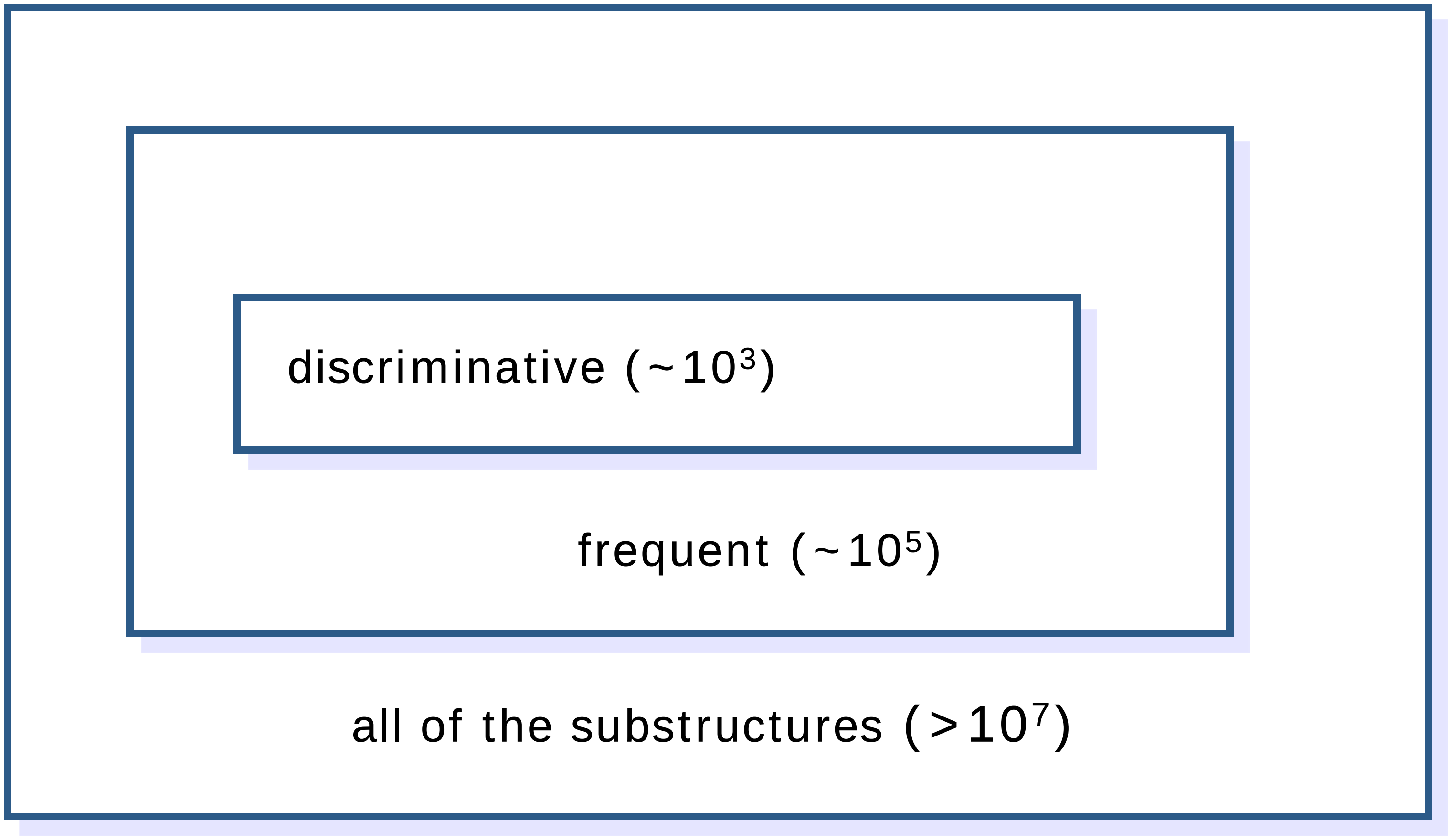
(c)

QUERY GRAPH



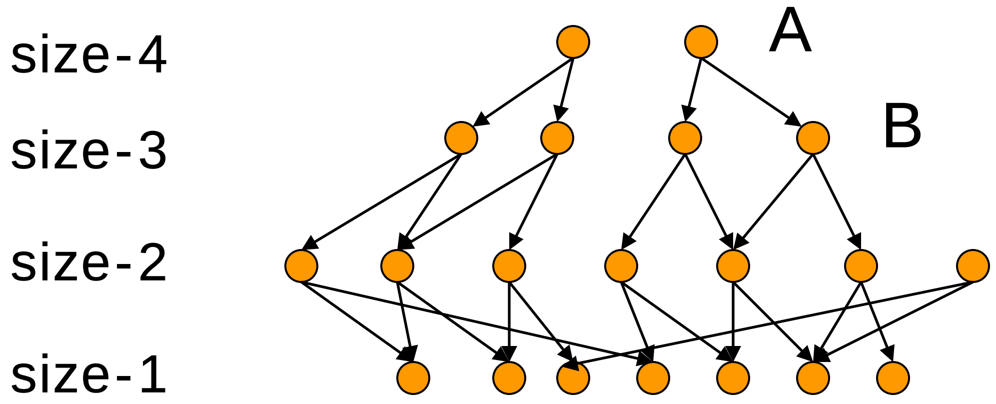
Only graph (c) contains this query graph. However, if we only index paths: C, C-C, C-C-C, C-C-C-C, we cannot prune graphs (a) and (b).

Using Frequent Patterns!!!



Discriminative Graphs

Remark: It is a kind of pattern post processing



patterns

Discriminative Graphs

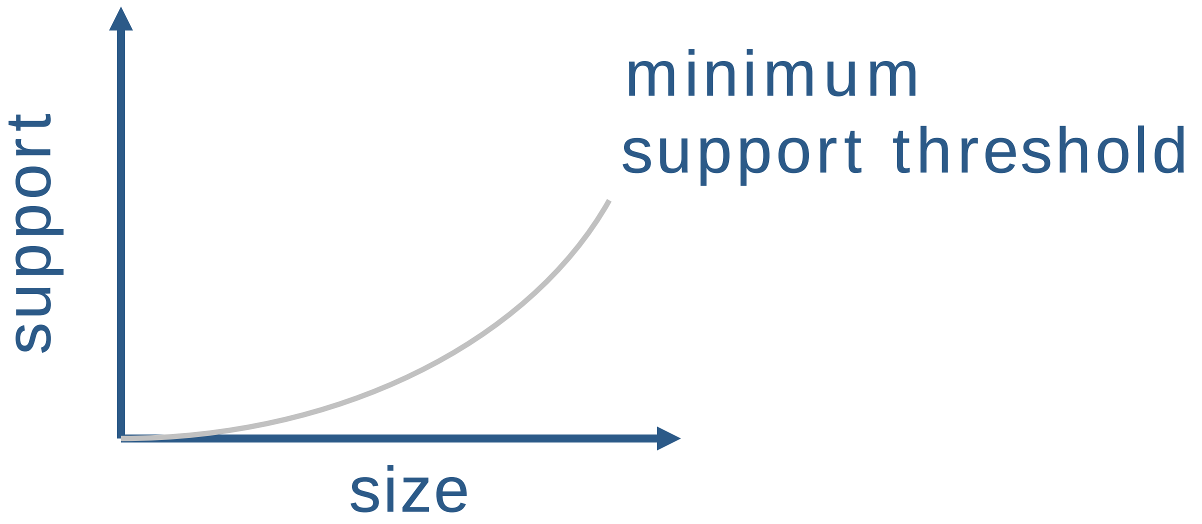
- Pinpoint the most useful frequent structures
  - Given a set of structures $f_1, f_2, \cdots f_n$ and a new structure x, we measure the extra indexing power provided by x ,

$$P\left(g \text{ contains } x \middle| g \text{ contains } f_1, f_2, \ldots f_n\right), f_i \subset x.$$

  When P is small enough, x is a discriminative structure and should be included in the index
- Index discriminative frequent structures only - Reduce the index size by an order of magnitude

Why Frequent Structures?

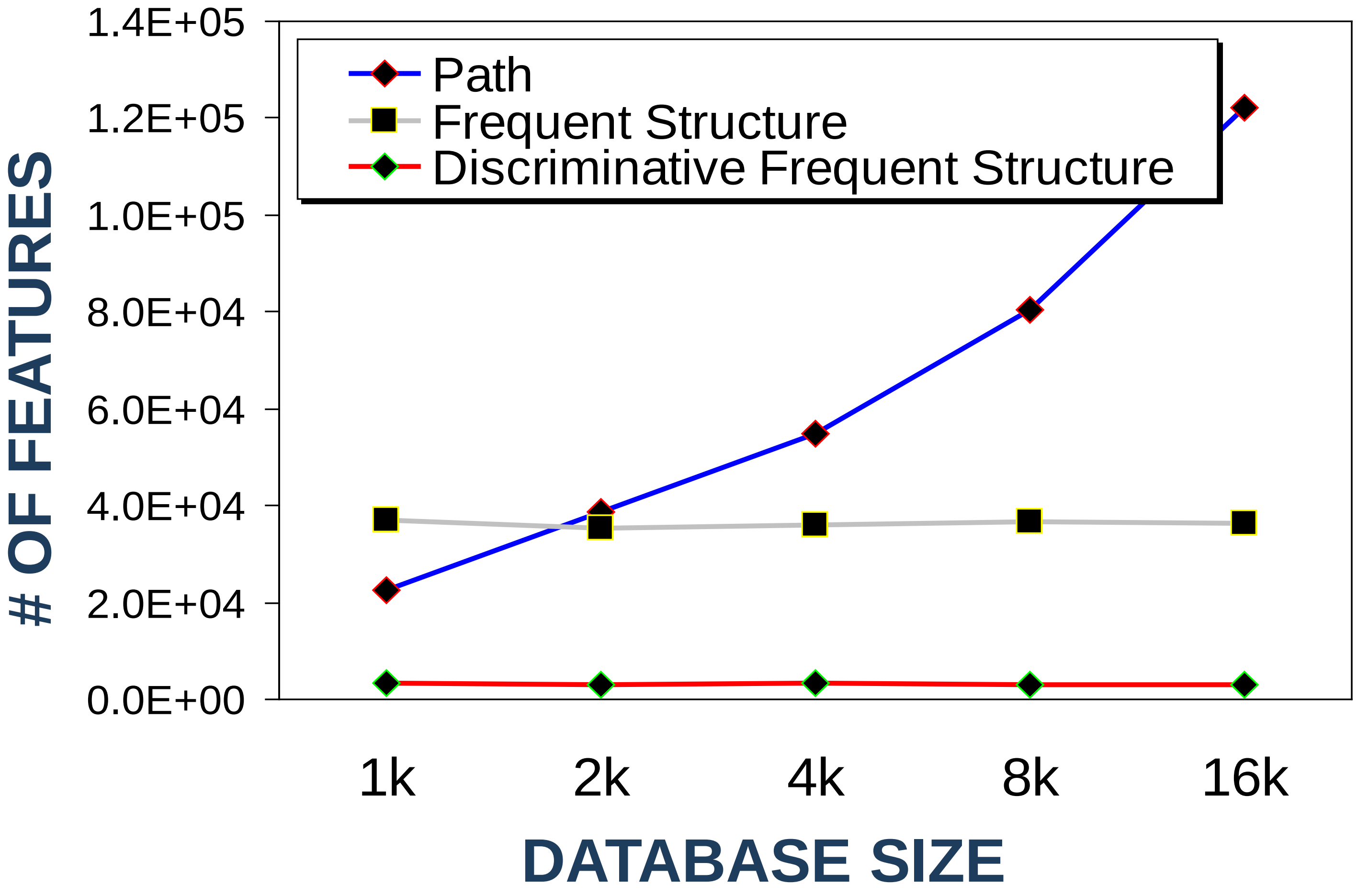
- We cannot index (or even search) all of substructures
- Large structures will likely be indexed well by their substructures
- Size-increasing support threshold



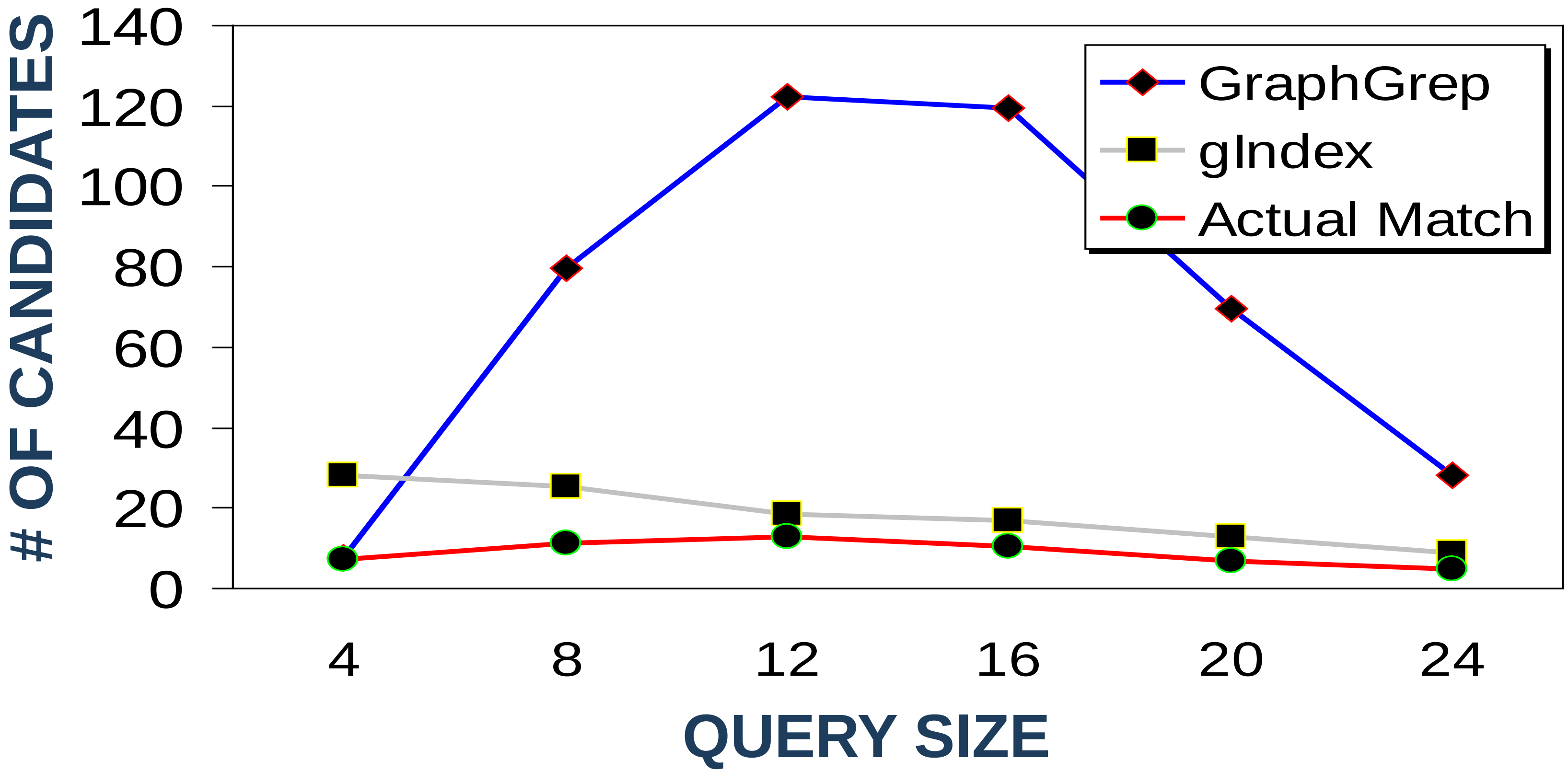
Index Graphs by Data Mining

- Identify frequent structures in the database
- Create a pattern lattice, Prune redundant frequent structures to obtain a small set of discriminative structures
- Create an inverted index between discriminative frequent structures and graphs in the database

Experiments: Index Size

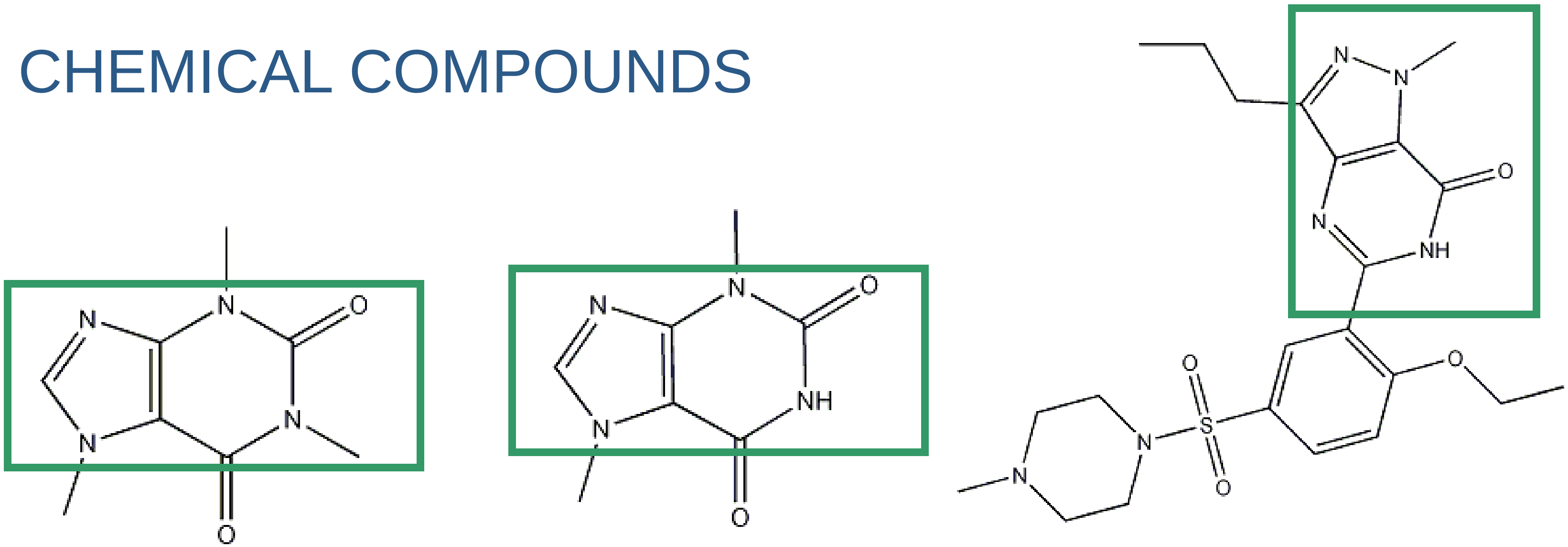


Experiments: Answer Set Size



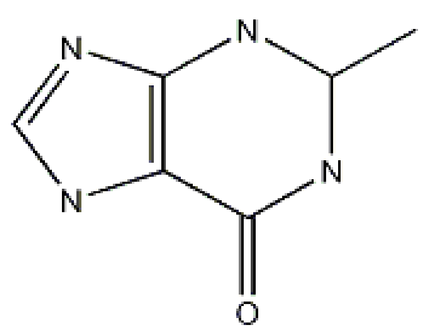
Structure Similarity Search

- CHEMICAL COMPOUNDS



(a) caffeine (b) diurobromine (c) viagra

- QUERY GRAPH



Similarity Measure

- Feature-based similarity measure
  - Each graph is represented as a feature vector

$$X = \{x_1, x_2, \ldots, x_n\}$$

  - The similarity is defined by the distance of their corresponding vectors
  - Advantages
    - Easy to index
    - Fast
    - Rough measure

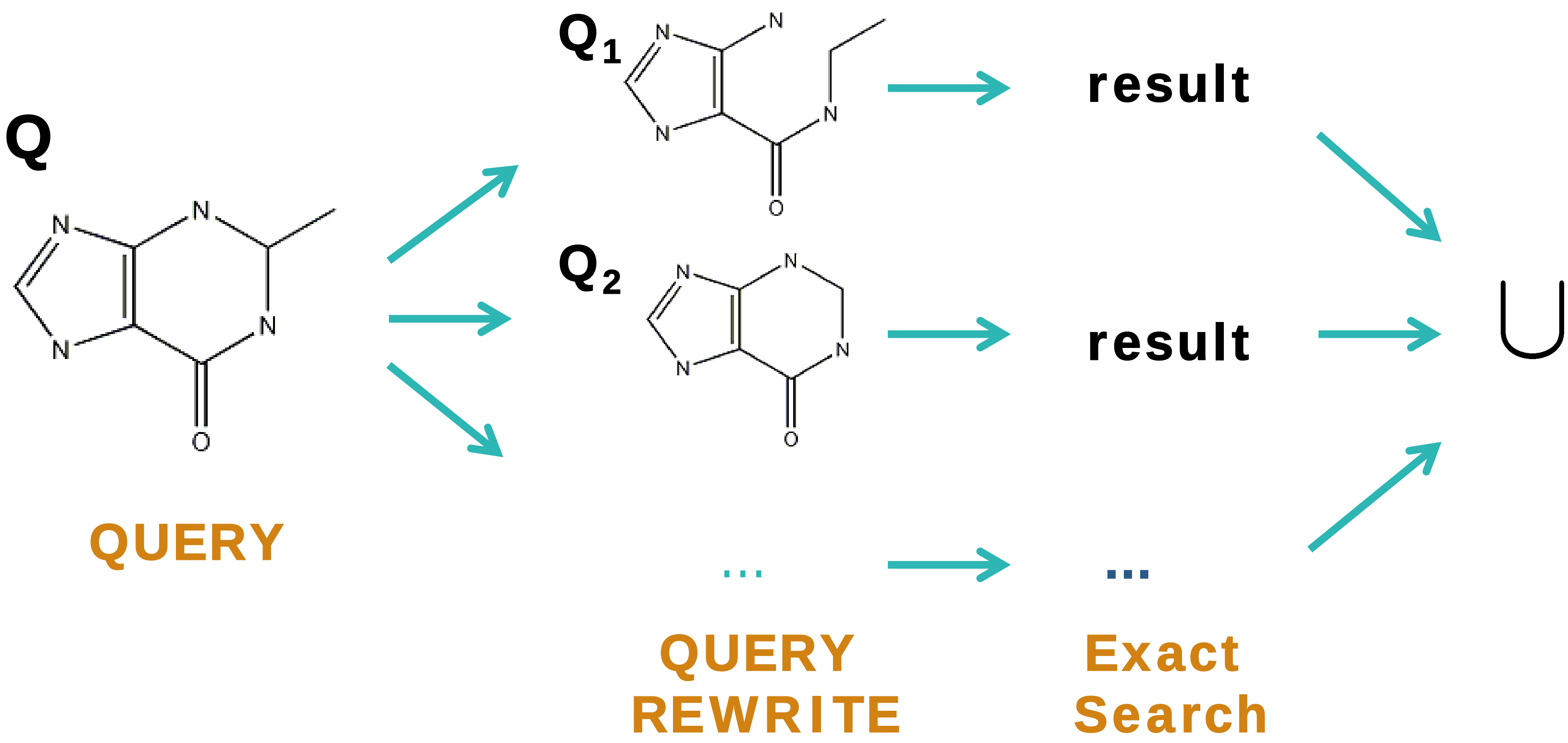
Similarity Measure

- Structure-based similarity measure
  - The maximum common subgraph (P) between query graph (Q) and target graph (G)

$$similarity = \frac{|P|}{|Q|}$$

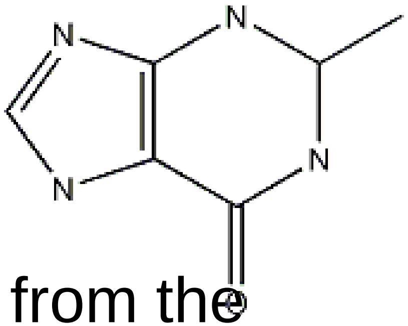
  - Similarity search: form P by deleting edges/nodes from Q; find graphs that contain P

Structure-based Similarity Measure



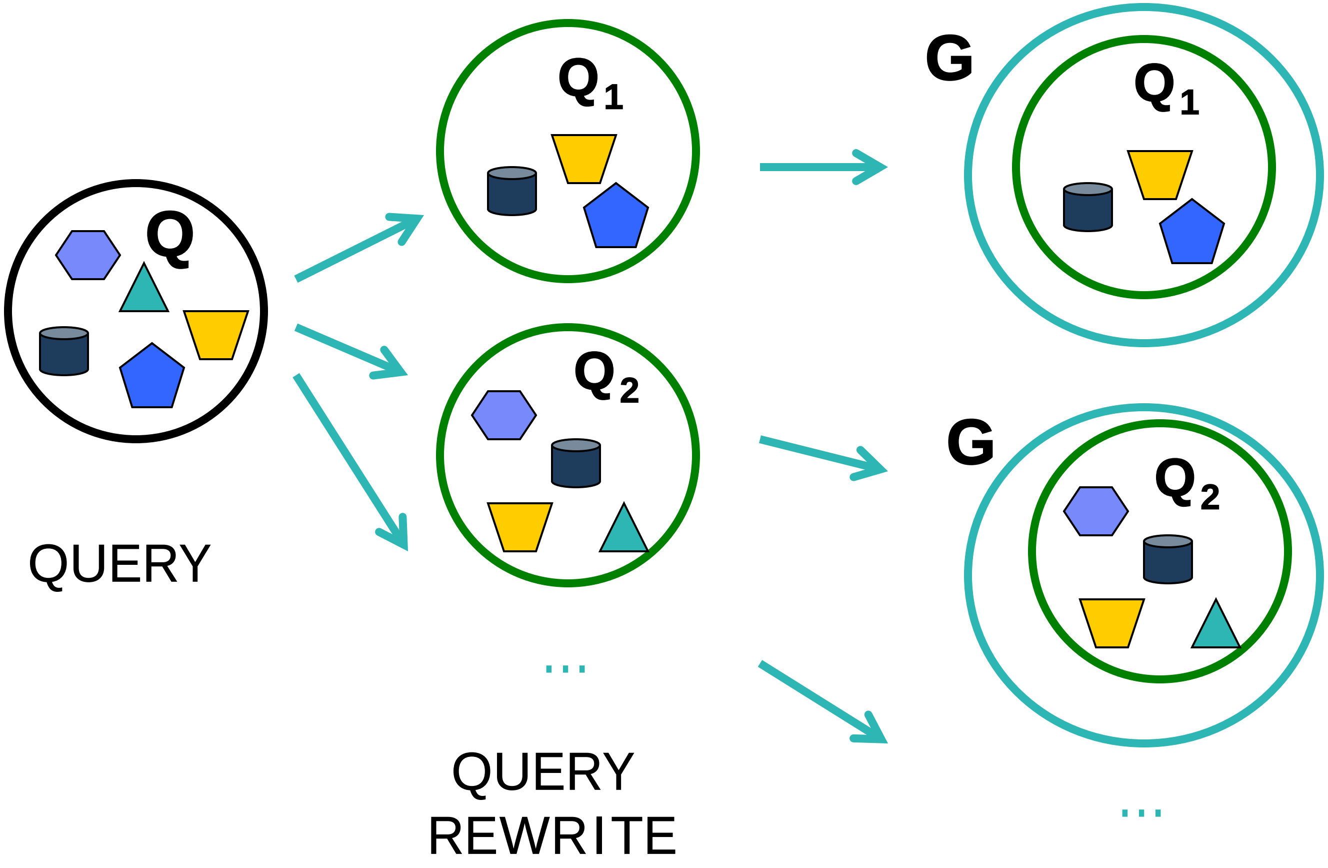
Some “Straightforward” Methods

- Method1: Directly compute the similarity between the graphs in the DB and the query graph
  - Sequential scan
  - Subgraph similarity computation



- Method 2: Form a set of subgraph queries from the original query graph and use the exact subgraph search
  - Costly: If we allow 3 edges to be missed in a 20-edge query graph, it may generate 1,140 subgraphs

From Edge Misses To Feature Misses



At least 3 of 5 features should be retained

Feature-based Pruning

Feature-Graph Matrix

features

	G_1	G_2	G_3	G_4	G_5
f_1	0	1	0	1	1
f_2	0	1	0	0	1
f_3	1	0	1	1	1
f_4	1	0	0	0	1
f_5	0	0	1	1	0

Assume a query graph has 5 features;
At least 3 features should be retained

Feature Miss Estimation

- Connection to maximum coverage
  - If we allow *k* edges to be relaxed (relabel or deletion), *J* is the maximum number of features to be hit by *k* edges - maximum coverage problem

- **maximum coverage problem:** Given several sets and a number, the sets may have some elements in common. You must select at most k of these sets such that the maximum number of elements are covered, i.e. the union of the selected sets has maximal size.

- NP-complete

Feature-Edge Matrix

features

edges

	f_1	f_2	f_3	f_4	f_5
~~e_1~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~
e_2	0	1	0	0	0
e_3	1	0	1	0	0
e_4	1	0	0	0	1
e_5	0	0	1	1	0

- A greedy algorithm exists

$$J_{greedy} \geq (1 - (1 - \frac{1}{k})^k) \cdot J$$

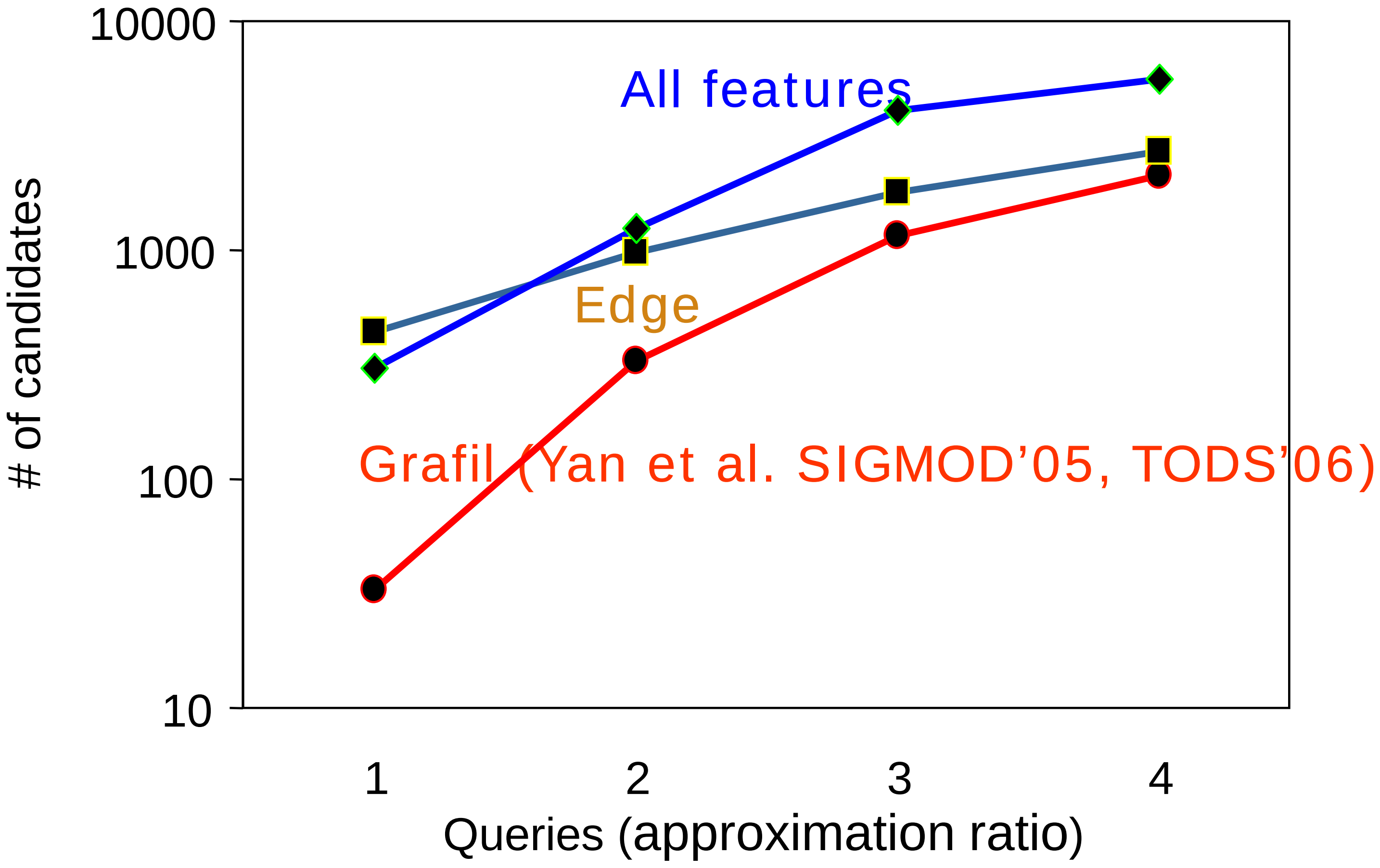
Feature Selection

Should we use all the features in a query graph?

- Features differentiate with selectivity and size
- How to select a good feature set?
  - features with similar properties: clustering
  - enough number of features

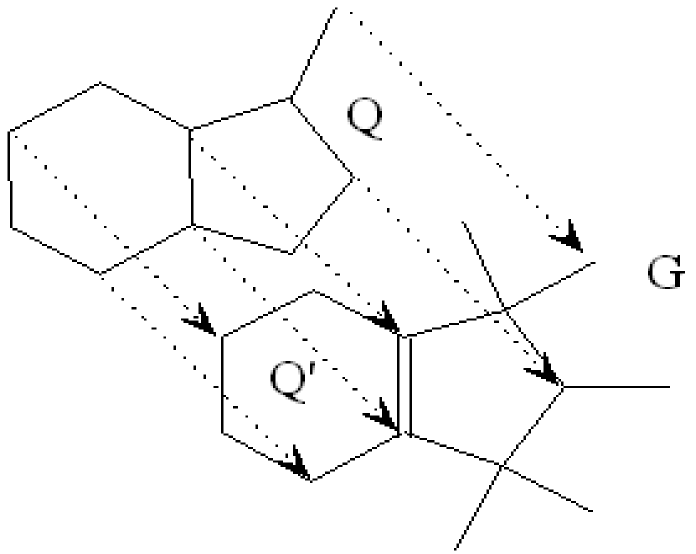
Remark: another kind of pattern post processing

Feature Selection Works



Superimposed Distance

Same Topological Structure
But different Labels



$$\mathbf{MD} = \sum_{v'=f(v)} \mathbf{D}(l(v), l'(v')) + \sum_{e'=f(e)} \mathbf{D}(l(e), l'(e'))$$

Minimum Superimposed Distance

Given two graphs, Q and G, let M be the set of subgraphs in G that are isomorphic to Q. The minimum superimposed distance between Q and G is the minimum distance between Q and Q' in M.

$$d(Q, G) = \min_{Q' \in M} d(Q, Q'),$$

where $d(Q, Q')$ is a distance function of two isomorphic graphs Q and Q'.

Substructure Search With Superimposed Distance

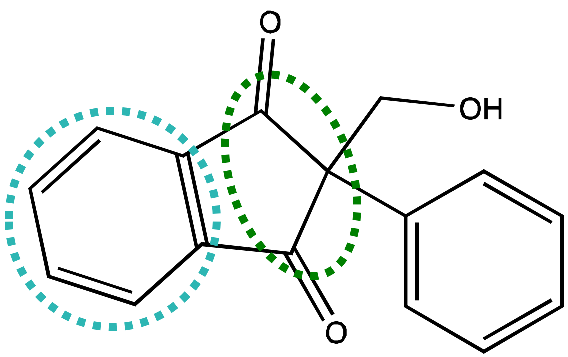
Given a set of graphs $D=\{G_1, G_2, ..., G_n\}$
and a query graph Q,
SSSD is to find all G_i in D such that

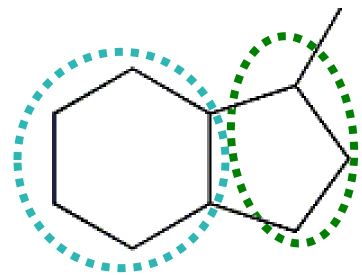
$$d(Q, G_i) \leq \sigma$$

Feature Partitions

Target graph G

Query graph Q

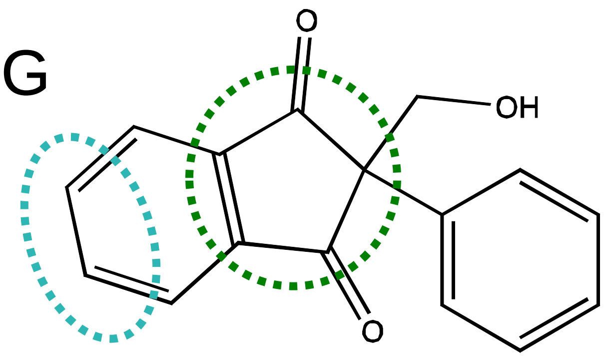




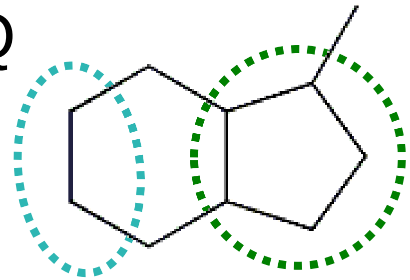
Partition I

Hexagon + Path

G



Q



Partition II

Pentagon + Path

Partition-Based Search

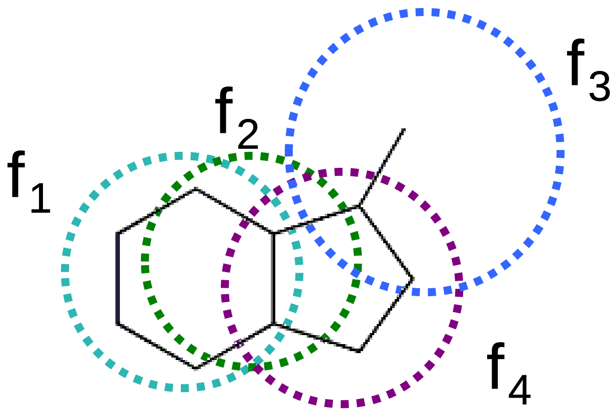
- We partition a query graph Q into non-overlapping indexed features f_1, f_2, ..., f_m, and use them to do pruning. If the distance function satisfies the following inequality,

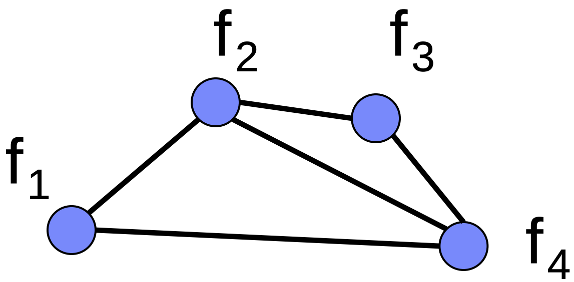
$$\sum_{i=1}^{m} d(f_i, G) \leq d(Q, G)$$

We can get the lower bound of the superimposed distance between Q and G by adding up the superimposed distance between f_i and G.

Overlapping Relation Graph

Query graph Q



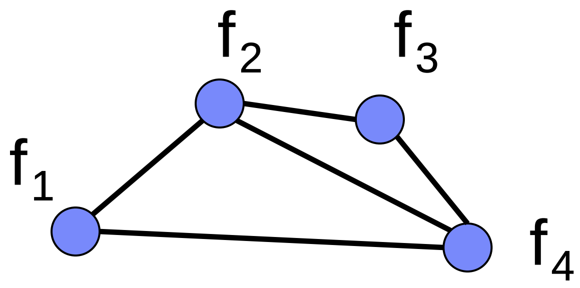


node: feature

edge: overlapping

node weight: minimum distance between f_i and G, $d(f_i, G)$

Across Multiple Graphs



node weight is redefined

Using average minimum distance between a feature f and the graphs G_i in the database, written as

$$w(f) = \frac{\sum_{i=1}^{n} d(f, G_i)}{n}$$

Structured Query Language

“find all patients diagnosed with eye tumor”

```
WITH Traversed (cls,syn) AS (
    (SELECT R.cls, R.syn
    FROM XMLTABLE ('Document("Thesaurus.xml")
      /terminology/conceptDef/properties
      [property/name/text()="Synonym" and
      property/value/text()="Eye Tumor"]
      /property[name/text()="Synonym"]/value'
      COLUMNS
      cls CHAR(64) PATH './parent::*/parent::*
                         /parent::*/name',
      tgt CHAR(64) PATH'.') AS  R)
UNION ALL
    (SELECT CH.cls,CH.syn
    FROM Traversed PR,
      XMLTABLE ('Document("Thesaurus.xml")
      /terminology/conceptDef/definingConcepts/
      concept[./text()=$parent]/parent::*/parent::*/
      properties/property[name/text()="Synonym"]/value'
      PASSING  PR.cls AS "parent"
      COLUMNS
      cls CHAR(64) PATH './parent::*/
                         parent::*/parent::*/name',
      syn CHAR(64) PATH'.') AS CH))
SELECT DISTINCT V.*
FROM Visit V
WHERE V.diagnosis IN
  (SELECT DISTINCT syn FROM Traversed)
```

NCI thesaurus

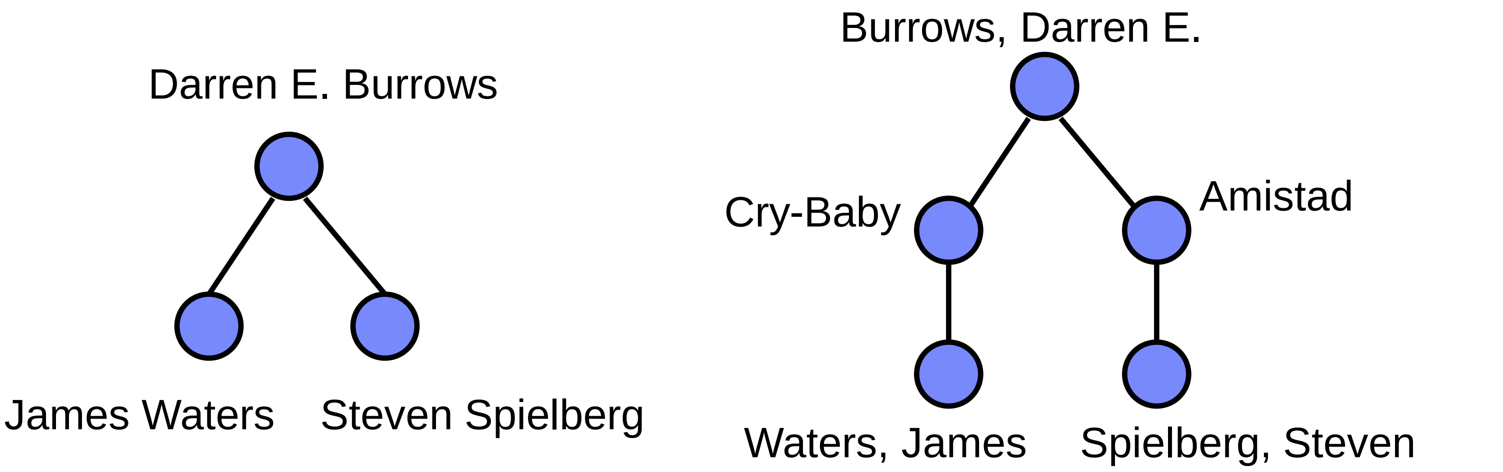
“Semantic queries by example”, Lipyeow Lim et al., EDBT 2014

Graph Search 2.0

- Large Scale Graph

	# of Entities	# of Links
Facebook	>1.2B	X 300
Twitter	270 M	500M tweets /day
Wikipedia	38 M	3B

- Lack of fixed representation



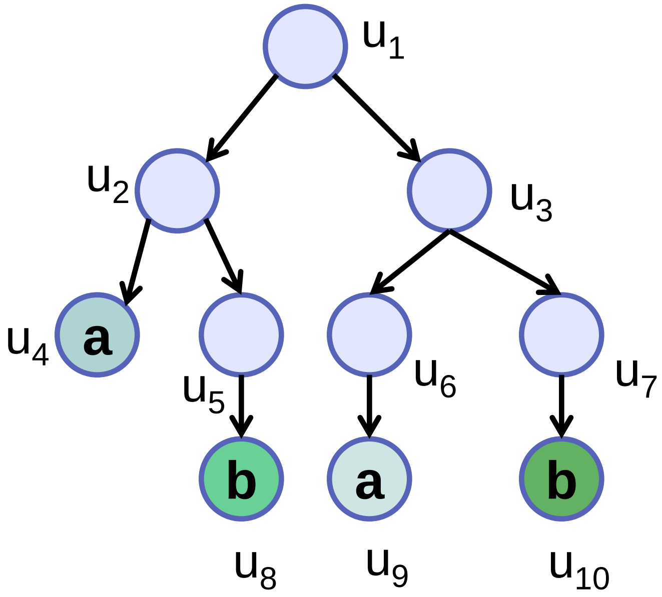
IMDB Network

FreeBase

Challenges

- Novel graph queries are emerging, that integrate both structure and content information.
- Traditional graph algorithms do not scale well for large scale graphs.
- Standard SQL or SPARQL queries cannot be applied to graph data that are lack of fixed schema, label and type information.

Keyword Search



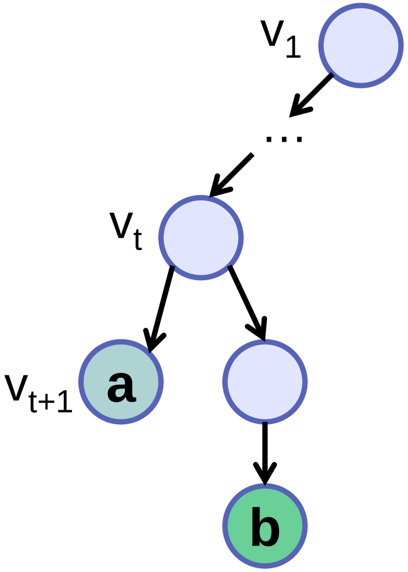
graph data

Query Keywords = {a, b}

Semantics:

- Directed graph: A node whose descendants containing the keywords (XRANK, BANKS, …)
- Undirected graph: A structure that contains the keywords

XRank (SIGMOD'03)



- Given a query

$$Q = (k_1, k_2, \ldots k_n)$$

- Raking respect to one keyword

$$r(v_1, k_i) = ElemRank(v_t) \times decay^{t-1}$$

  - v_{t+1} is the node that directly contains the keyword k_i
  - t is the level distance between v_1 and v_t
  - *Decay* is the a parameter penalizing the distance
  - *ElemRank* is the importance of a node in the graph
  - *ElemRank(v_t)* is in fact related to *ElemRank(v_1)* due to certain properties of containment edges.

XRank (cont.)

- Raking respect to all the keywords:

$$R(v_1, Q) = (\sum_{1 \le i \le n} \hat{r}(v_1, k_i)) \times p(v_1, k_1, k_2, ..., k_n)$$

$p(v_1, k_1, k_2, \ldots, k_n)$ is the keyword proximity function, which can be any function that ranges from 0 (keywords are very far apart in v1) to 1 (keywords occur right next to each other in v1

Tree Based Approach

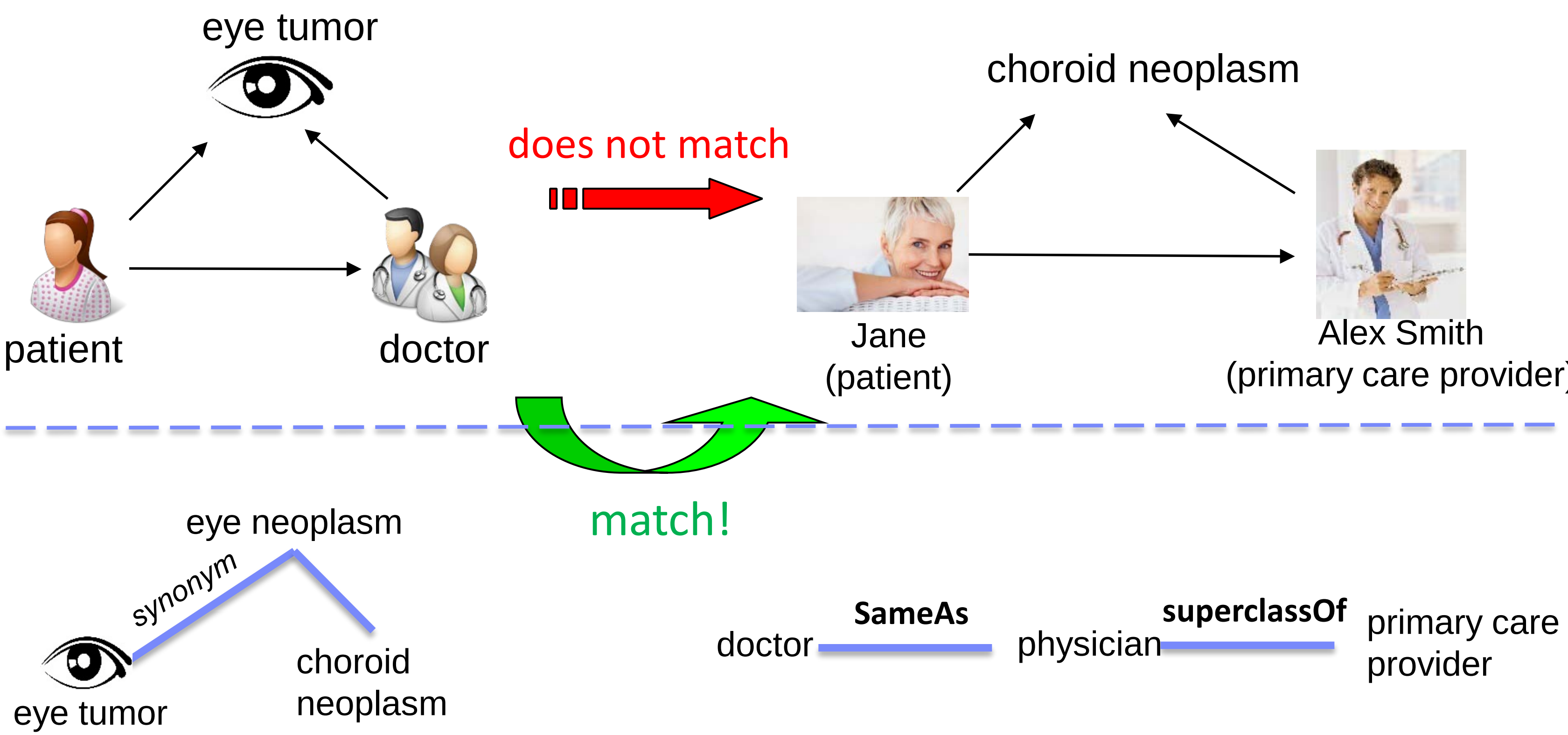
- Result is a connected tree containing all query keywords
- Score function:
  - (i) sum of all edge weights in the tree, or
  - (ii) sum of all path weights from root to each keyword in the tree
- Algorithm: Find top-k result trees with minimum score
  - Bidirectional Search [Kacholia et. al., *VLDB '05*]
  - BLINKS [He et. al., *SIGMOD '07*]
  - Dynamic Programming [Ding et. al., *ICDE '07*]
  - External Memory [Dalvi et. al, *VLDB '08*]

Graph Based Approach

- Result is a connected ***graph*** containing all query keywords
- Score function:
  - (i) sum of all edge weights in the graph, or
  - (ii) maximum pairwise distance, or
  - (iii) min-max pairwise distance.
- Algorithm: Find top-k result graphs with minimum score
  - EASE [Li et. al., *SIGMOD '08*]
  - r-Clique [Kargar et. al., *KDD '11*]
  - Team Formation [Lappas et. al., *KDD '09*]

Ontology-based Graph Query

"find information about the patients with eye tumor, and doctors who cured them."

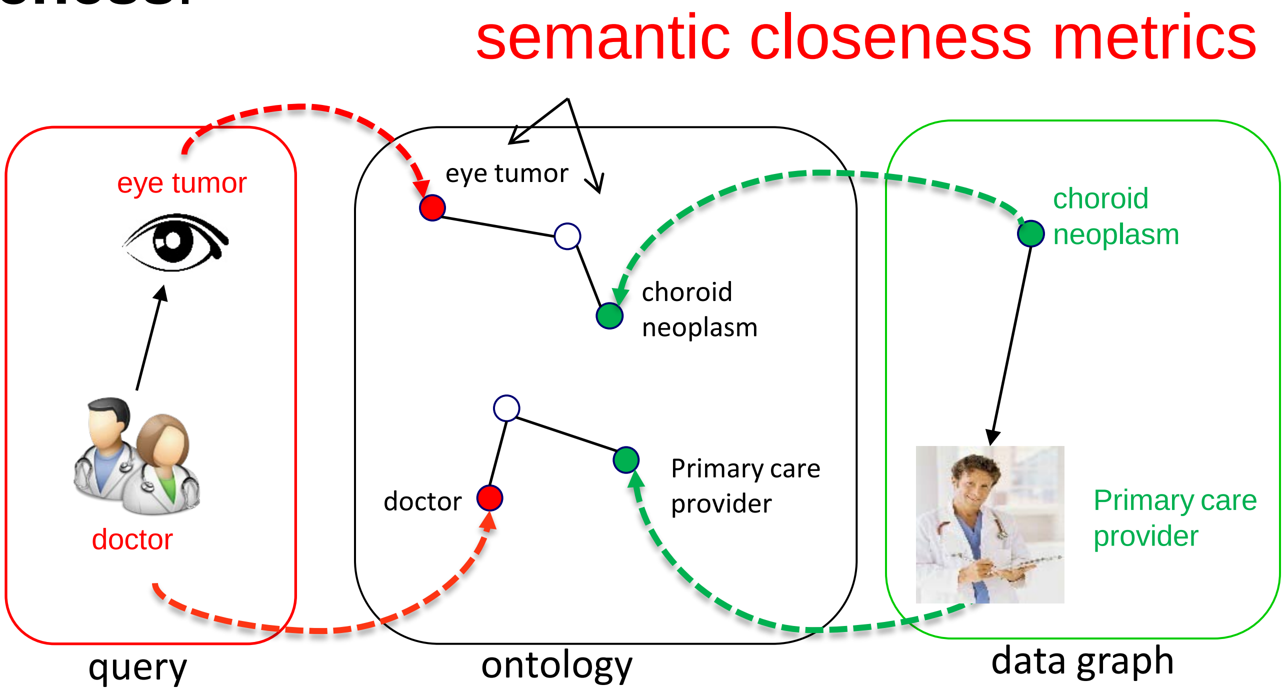


Using ontologies to capture semantically related matches

46 46

Ontology-based Graph Querying

- Given a data graph, a query graph and an ontology graph, identify K best matches with **minimum semantic closeness**.



What If

Data Graph **Query**

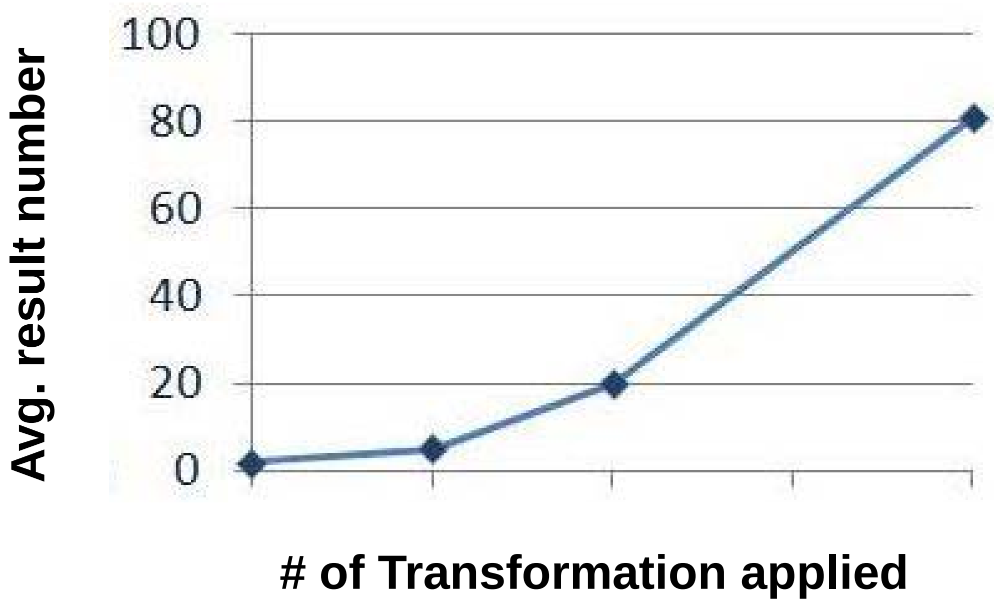
Transformation	Category	Example
First/Last token	String	“Barack Obama” > “Obama”
Abbreviation	String	“Jeffrey Jacob Abrams” > “J. J. Abrams”
Prefix	String	“Doctor” > “Dr”
Acronym	String	"International Business Machines" > "IBM"
Synonym	Semantic	“tumor" > “neoplasm"
Ontology	Semantic	"teacher" > "educator"
Range	Numeric	“1980” > “~30”
Unit Conversion	Numeric	"3 mi" > "4.8 km"
Distance	Topology	"Pine" - "M:I" > "Pine" - "J.J. Abrams" - "M:I"
…	…	…

User Scenario

- Users want to freely post queries, without possessing any knowledge of the underlying data.
- The querying system should automatically find the matches through a set of **transformations** including ontology.

✓**Acronym transformation** matches ‘UCB’ to ‘University of California, Berkeley’
✓**Abbreviation transformation** matches ‘M : I’ to ‘Mission: Impossible’
✓**Numeric transformation** matches ‘~30’ to ‘1980’.
✓**Structural transformation** matches ‘an edge’ to ‘a path’.

Transformation-based Graph Matching



- Transformation-based graph matching produces more results
- Suggest only the “best” results to the users

Ranking Function (I)

- Different kinds of transformations, equal weight?

Example: Given a single node query, “Chris Pine”

Nodes with “C. Pine” (Abbreviation)
shall be ranked higher than
Nodes with “Pine” (Last token)

- Different weights! How to determine them?
- Weights shall be learned

Ranking Function

- With a set of transformations $\{f_i\}$, given a query Q and its match result R, our ranking model considers
  - the node matching: from a query node v to its match

$$F_V(v,\phi(v)) = \sum_i \alpha_i f_i(v,\phi(v)) \qquad \phi(v)$$

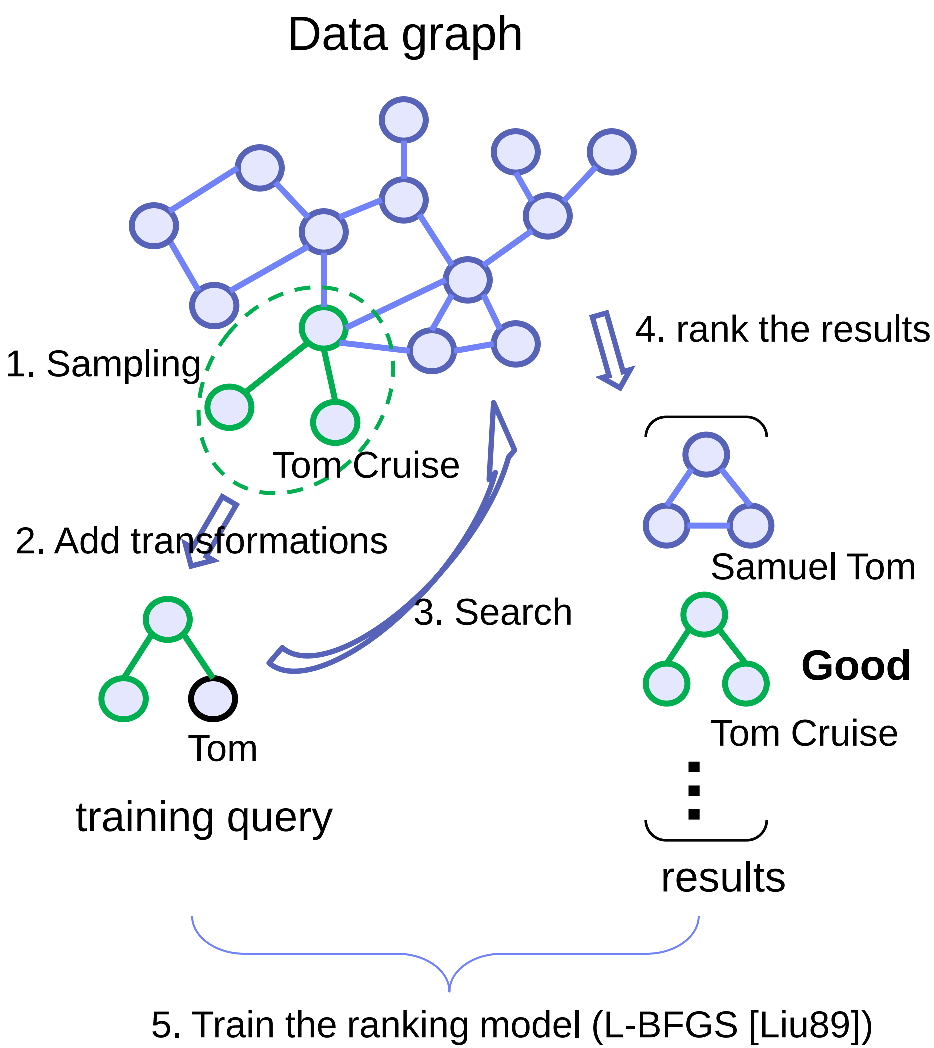
  - the edge matching: from query edge e to its match

$$F_E(e,\phi(e)) = \sum_i \beta_i f_i(e,\phi(e)) \qquad \phi(e)$$

- Overall ranking model:

$$P(\phi(Q) \mid Q) \propto \exp\Big(\sum_{v \in V_Q} F_V(v,\phi(v)) + \sum_{e \in E_Q} F_E(e,\phi(e))\Big)$$

Automatically Generate Training Data



- **Sampling**: a set of subgraphs are randomly extracted from the data graph.
- **Query generation**: the queries are generated by randomly adding transformation on the extracted subgraphs.
- **Searching**: search the generated queries on the data graph
- **Labeling**: the results are labeled based on the original subgraph.
- **Training**: the queries, with the labeled results, are then used to estimate the parameters of the ranking model.

Write a Graph Query?

User Scenario:

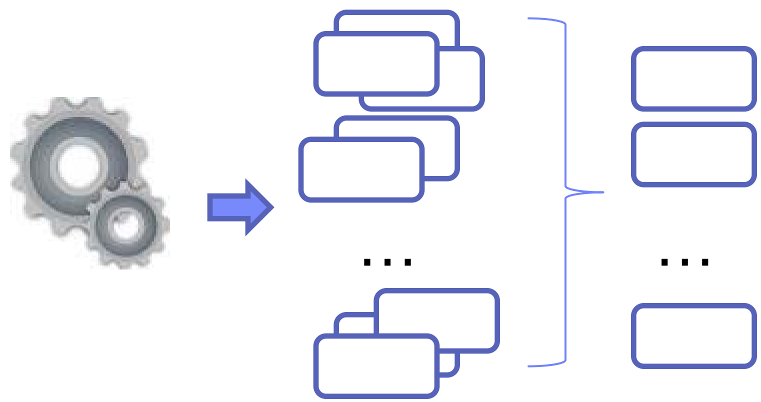
I have no idea about schema/data specification; yet I still want to query.

several keywords

draw a graph

one example (ICDE’14, Li etc.)

Natural Language Query (SIGMOD’14, Zou, etc.)



answers

summary answers (VLDB’14)

References

- Schemaless and Structureless Graph Querying, by S. Yang, Y. Wu, H. Sun, X. Yan, VLDB 2014
- Ontology-based Subgraph Querying, by Y. Wu, S. Yang, X. Yan, ICDE 2013
- Towards Graph Containment Search and Indexing, by C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang and X. Gu. VLDB 2007.
- Graph Indexing: A Frequent Structure-based Approach, by X. Yan, P. S. Yu, and J. Han, SIGMOD 2004.
- Substructure Similarity Search in Graph Databases, by X. Yan, P. S. Yu, and J. Han, SIGMOD 2005.
- D. Shasha, J.T-L Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. PODS 2002.
- XRANK: Ranked Keyword Search over XML Documents, by L. Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram, SIGMOD 2003