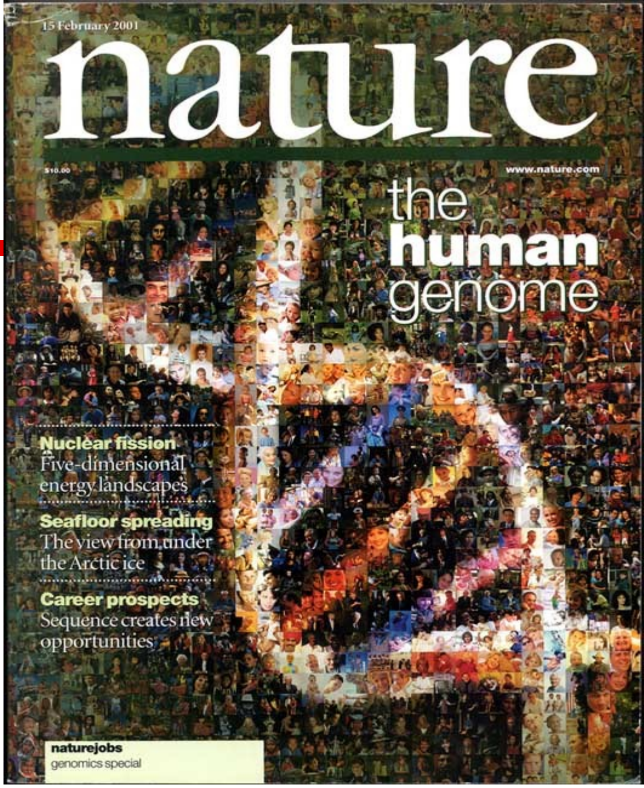
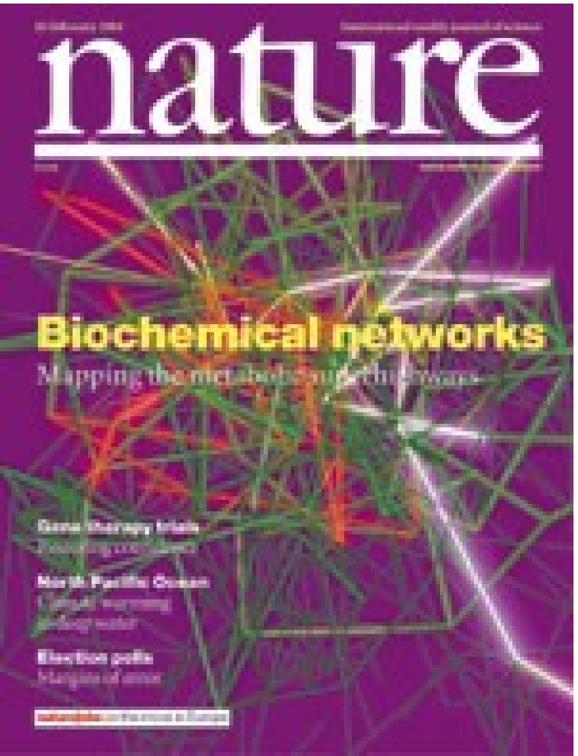
Mining, Indexing & Searching Graphs in Large Data Sets





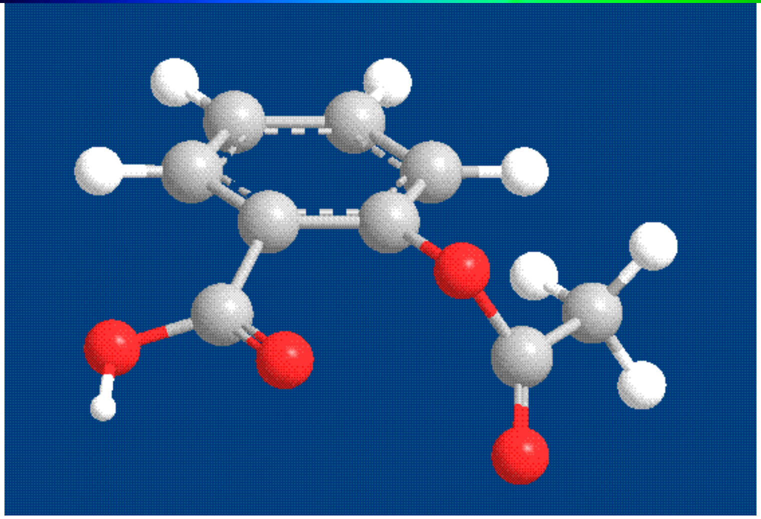
Jiawei Han

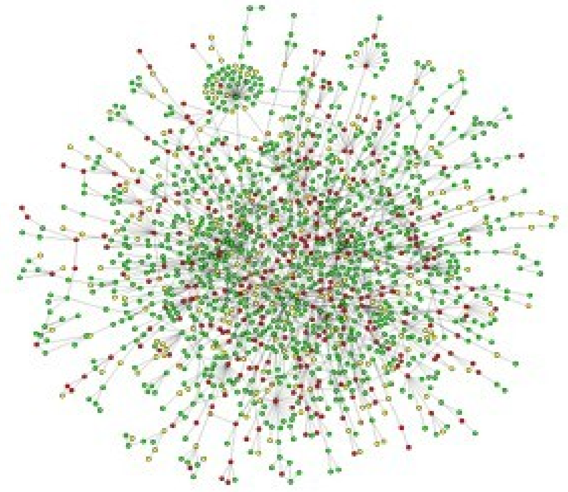
Department of Computer Science, University of Illinois at Urbana-Champaign

www.cs.uiuc.edu/~hanj

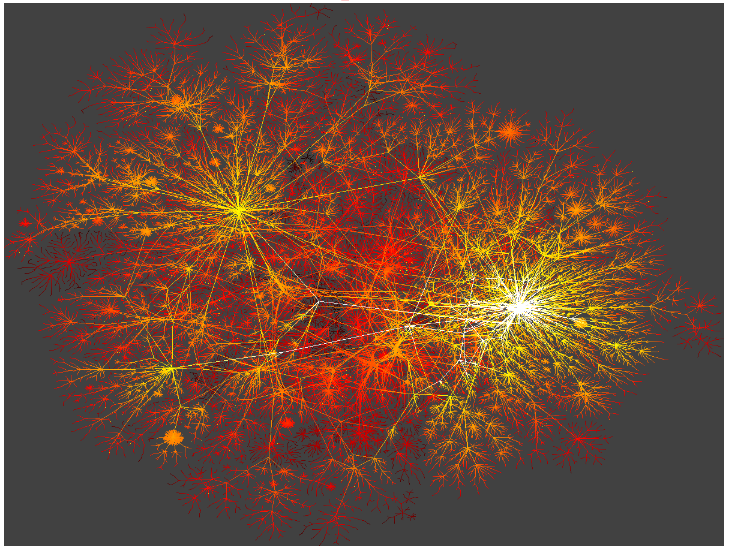
In collaboration with Xifeng Yan (IBM Watson), Philip S. Yu (IBM Watson), Feida Zhu (UIUC), Chen Chen (UIUC)

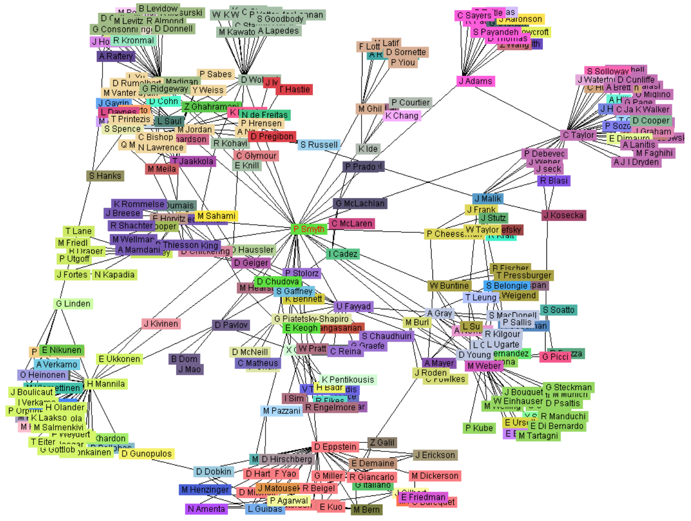
Graph, Graph, Everywhere


Aspirin


from H. Jeong et al Nature 411, 41 (2001)

Yeast protein interaction network


An Internet Web


Co-author network

Why Graph Mining and Searching?

- Graphs are ubiquitous
  - Chemical compounds (Cheminformatics)
  - Protein structures, biological pathways/networks (Bioinformactics)
  - Program control flow, traffic flow, and workflow analysis
  - XML databases, Web, and social network analysis
- Graph is a general model
  - Trees, lattices, sequences, and items are degenerated graphs
- Diversity of graphs
  - Directed vs. undirected, labeled vs. unlabeled (edges & vertices), weighted, with angles & geometry (topological vs. 2-D/3-D)
- Complexity of algorithms: many problems are of high complexity!

Outline

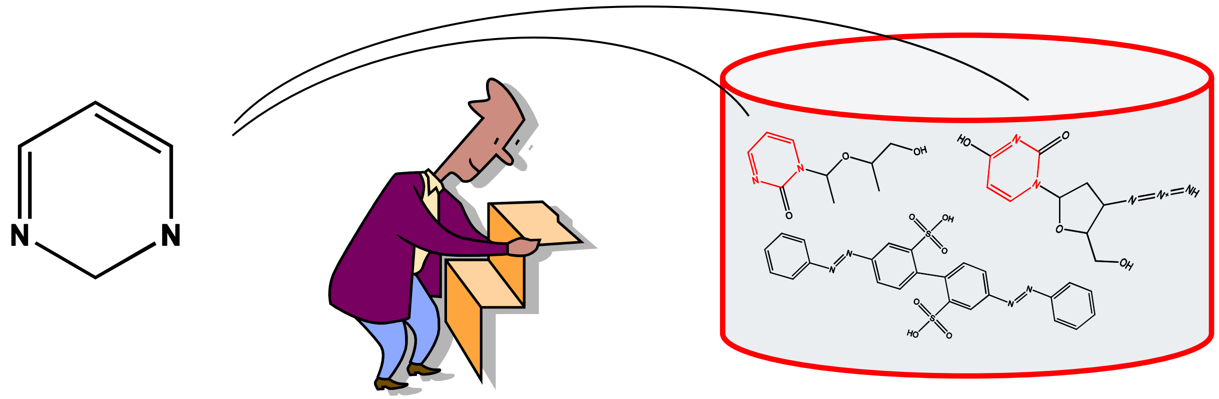
- Mining frequent graph patterns
- Constraint-based graph pattern mining
- Graph indexing methods
- Similairty search in graph databases
- Graph containment search and indexing

Research Papers Covered in this Talk

- X. Yan and J. Han, ***gSpan: Graph-Based Substructure Pattern Mining***, *ICDM'02*
- X. Yan and J. Han, ***CloseGraph: Mining Closed Frequent Graph Patterns***, *KDD'03*
- X. Yan, P. S. Yu, and J. Han, ***Graph Indexing: A Frequent Structure-based Approach***, *SIGMOD'04 (also in TODS'05, Google Scholar: ranked #1 out of 63,300 entries on "Graph Indexing")*
- X. Yan, P. S. Yu, and J. Han, "***Substructure Similarity Search in Graph Databases***", SIGMOD'05 *(also in TODS'06)*
- F. Zhu, X. Yan, J. Han, and P. S. Yu, "***gPrune: A Constraint Pushing Framework for Graph Pattern Mining***", PAKDD'07 (Best Student Paper Award)
- C. Chen, X. Yan, P. S. Yu, J. Han, D. Zhang, and X. Gu, "***Towards Graph Containment Search and Indexing***", VLDB'07, Vienna, Austria, Sept. 2007

Graph Search: Querying Graph Databases

- Querying graph databases:
  - Given a graph database and a query graph, find all graphs containing this query graph

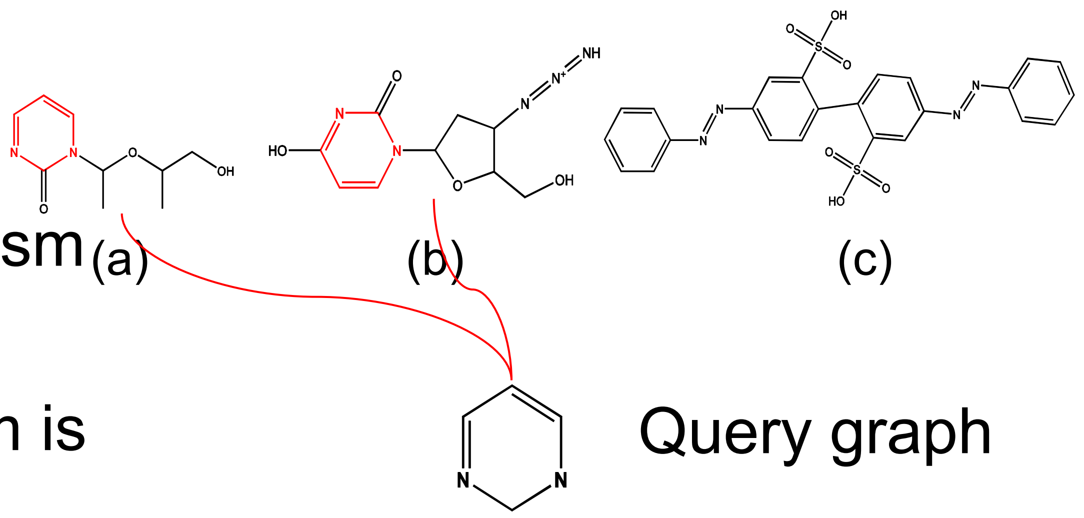


query graph

graph database

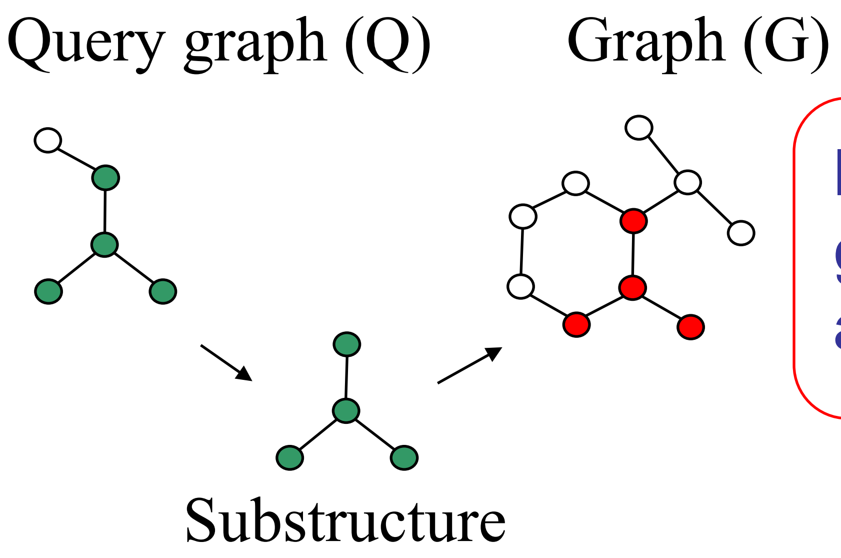
Scalability Issue

- Sequential scan
  - Disk I/O
  - Subgraph isomorphism testing
- An indexing mechanism is needed
  - DayLight: Daylight.com (commercial)
  - GraphGrep: Dennis Shasha, et al. PODS'02
  - Grace: Srinath Srinivasa, et al. ICDE'03



Sample database

Indexing Strategy



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

Remarks

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

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

Cost Analysis

Query Response Time

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

Graph index access time

Size of candidate answer set

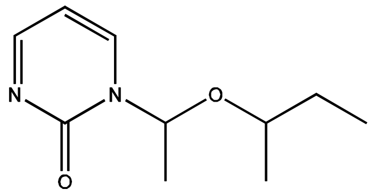
Disk I/O time

Isomorphism testing time

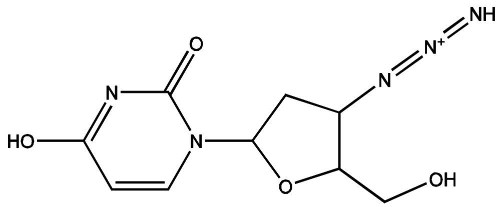
Remark: make $|C_q|$ as small as possible

Path-Based Approach

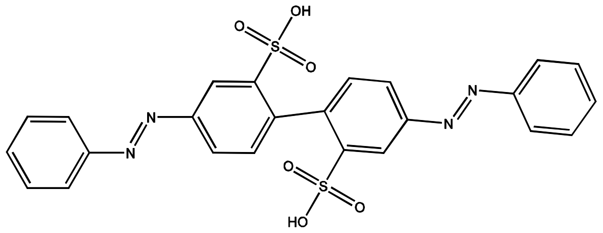
Sample database



(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: ...

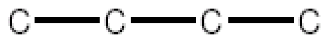
Built an inverted index between paths and graphs

GraphGrep – Path-based Approach

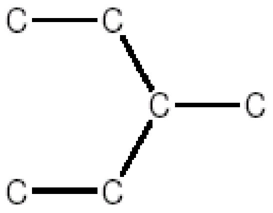
- Enumerate in each graph G_i all the existing paths upto length maxL.
- Build an inverted index based on the above paths. Note that as maxL is increased, the size of index increases considerably
- Enumerate the paths in the query upto maxL.
- Search in the inverted index all G_i that contain all paths contained in the query. Note that as maxL increases the number of sets in the index that need to be searched (or intersected) increases as well.
- On the other hand, if maxL is too small it wont characterize the query well!

Problems of Path-Based Approach

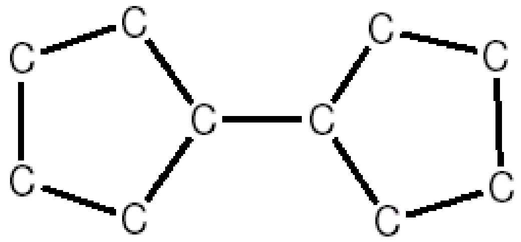
Sample database



(a)

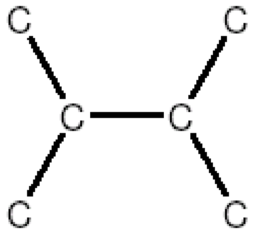


(b)



(c)

Query graph



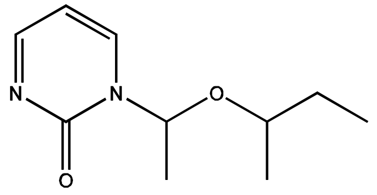
Only graph (c) contains this query graph. However, if we only index paths: C, C-C, C-C-C, C-C-C-C, we cannot prune graph (a) and (b).

gIndex: Indexing Graphs by Data Mining

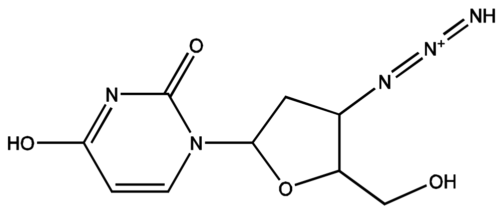
- Our methodology on graph index:
  - Identify frequent structures in the database, the frequent structures are subgraphs that appear quite often in the graph database
  - Prune redundant frequent structures to maintain a small set of discriminative structures
  - Create an inverted index between discriminative frequent structures and graphs in the database

Why Discriminative Subgraphs?

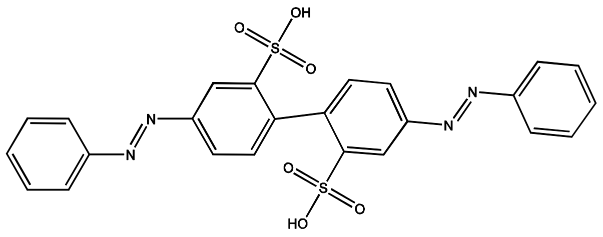
Sample database



(a)



(b)



(c)

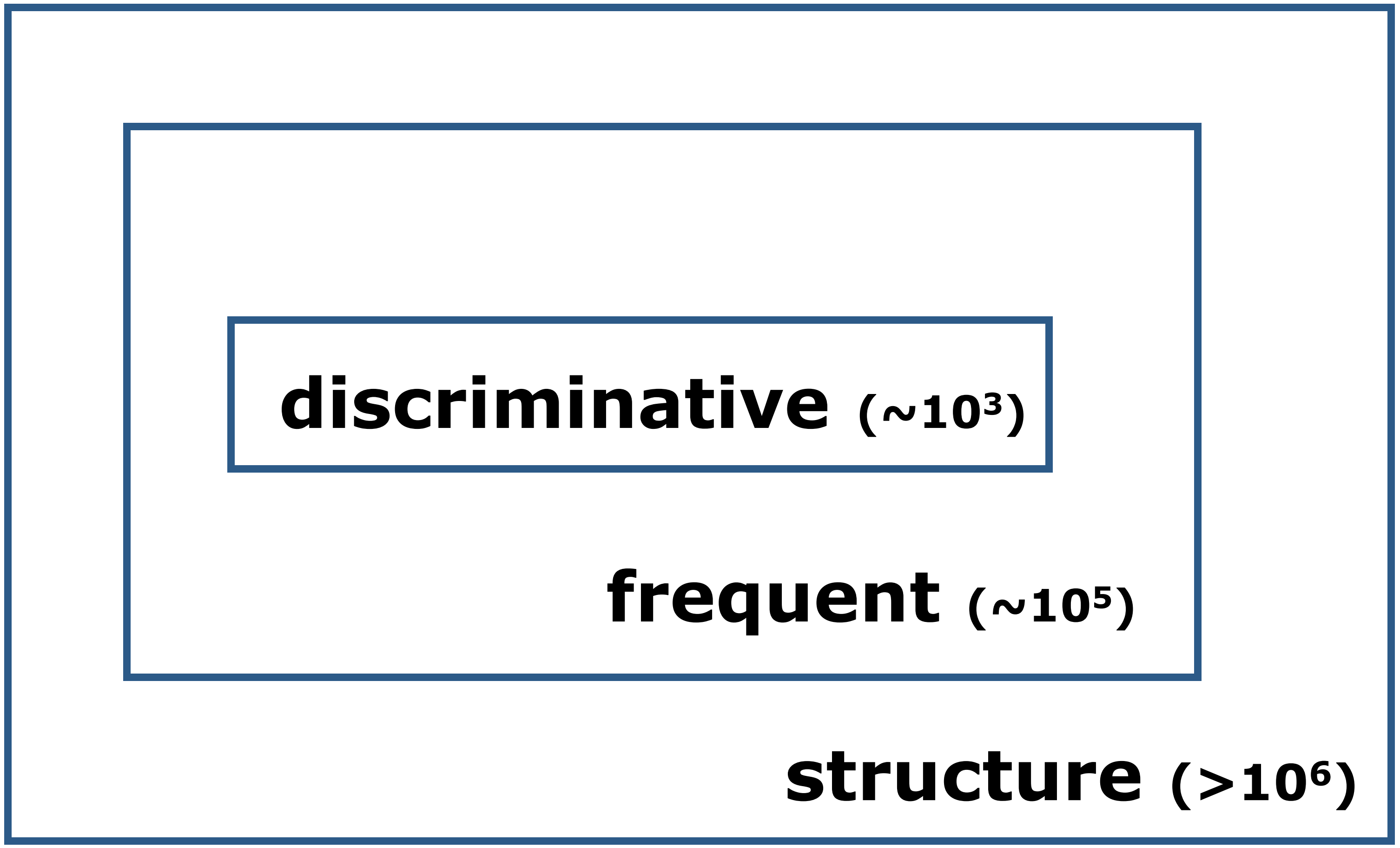
- All graphs contain structures: C, C-C, C-C-C
- Why bother indexing these redundant frequent structures?
  - Only index structures that provide more information than existing structures

Discriminative Structures

- Pinpoint the most useful frequent structures
  - Given a set of structures f_1, f_2, …, f_n and a new structure x, we measure the extra indexing power provided by x, $P\left(x \middle| f_1, f_2, \ldots f_n\right), f_i \subset x.$

    P (x|f_1, f_2, …, f_n), where f_i is contained in 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

Comparing Index size



Determining the right Support Value

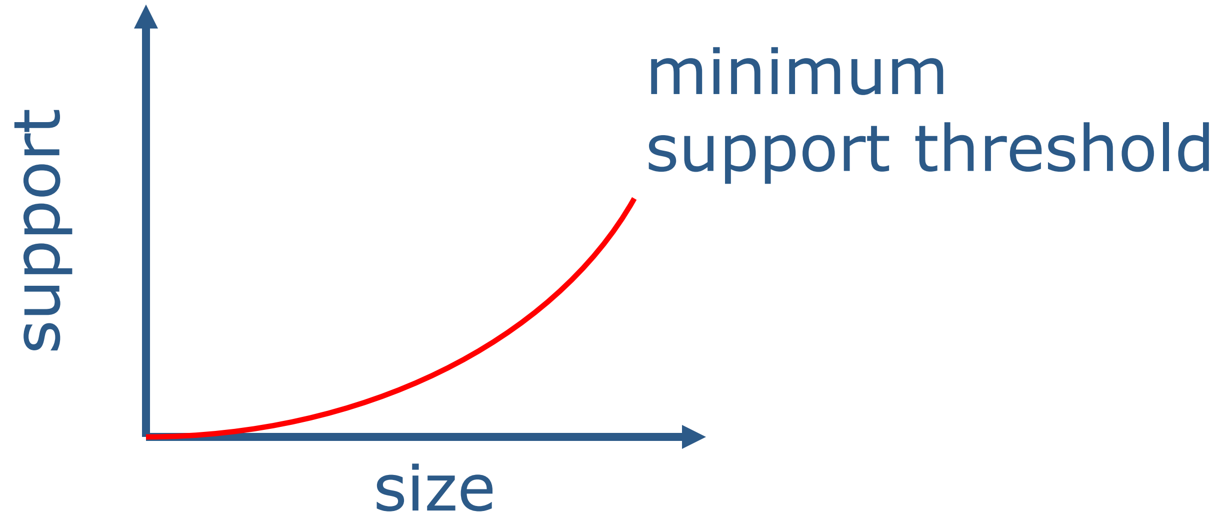
- The candidate set for a query Q is defined as
  - C_q = the intersection of the sets D_i where D_i is the set of graphs containing the frequent subgraph g_i (features) for each g_i which appears in the query.
- Now, if there are many such features, then there will be many Di and the intersection is likely to be small, which is good! However, index size will be very large.
- On the other hand if min-support is high, there will be few D_i thus C_q will be large which is bad but index size will be smaller
- In other words: *If minSup is set too high, the size of Cq may be too large. If minSup is set too low, it is too difficult to generate all the frequent fragments because there may exist an exponential number of frequent fragments under low support.*
- The solution – Varying minSupport!

Determining the right Support Value (Cont.)

- Let's examine a simple example: a *completely connected graph with 10 vertices, each of which has a distinct* label.
- There are 45 1-edge subgraphs, 360 2-edge ones, and more than 1*,814,400 8-edge ones*
- *As one can see, in or*der to reduce the overall index size, it is appropriate for the index scheme to have *low minimum support on small* Fragments and *high minimum support on large fragments (for compactness).*
- *This criterion on the se*lection of frequent fragments for effective indexing is called *size-increasing support constraint.*

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



Defining Discriminant features

- If two features appear in exactly (or approximately) the same set of graphs, then one of this feature is redundant. Conclusion – never include a graph and its sub-graph…
- A redundant feature is a feature f whose corresponding D_f is very close to the intersection of the sets D_{fi} where fi is a sub-graph of f.
- A discriminative feature is defined as:
- *Fragment x is discriminative with respect to F if*
  $Dx << \cap D_{fi}$
- Let us examine the query example . carbon chains, c - c, c - c - c, and c - c - c - c, are redundant and should not be used as indexing features in this dataset. The carbon ring is a discriminative fragment since only graph (c) in contains it while graphs (b) and (c) in Figure 1 have all of its sub-graphs.
- *The paper presents an algorithm to find the discriminative features using the above definition and the size-increasing support function (Alg. 1)*

Building the Gindex tree

- One uses again the Canonical labeling notation of gSpan
- The tree is built by levels where each level corresponds to the size of the sub-graph. The nodes contain pointers to the sets Di containing the pattern

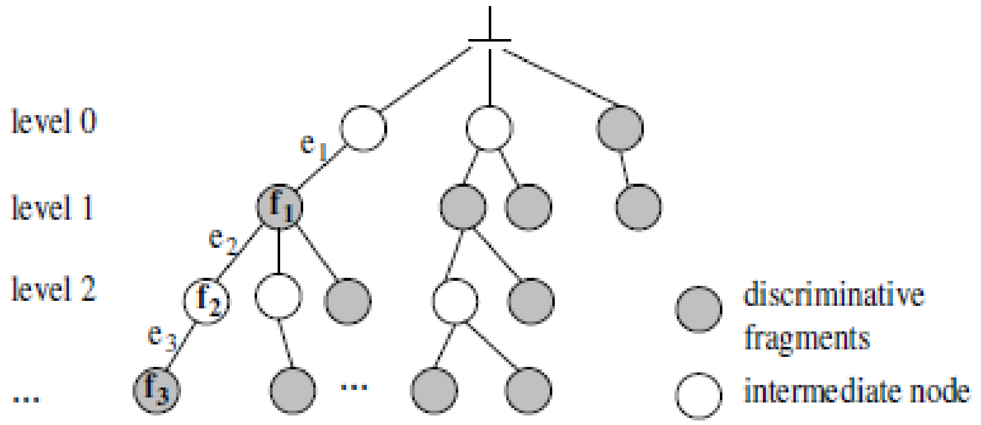
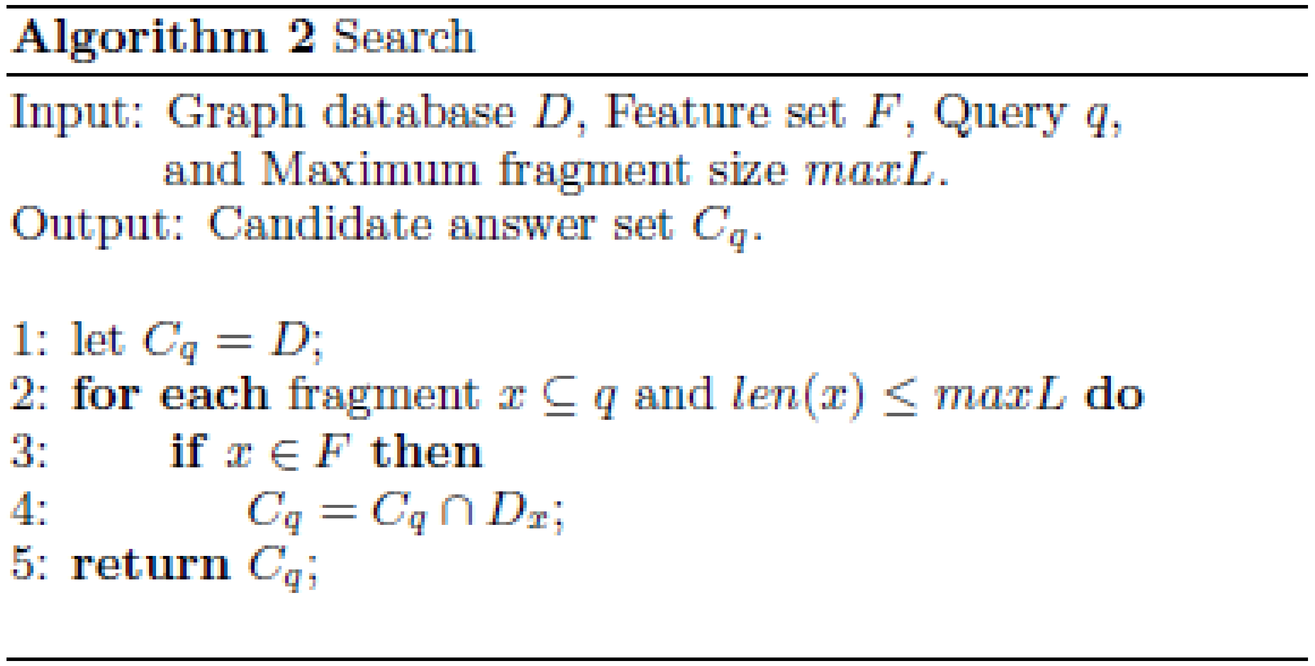


Figure 7: gIndex Tree

EXAMPLE 6. *Figure 7 shows a gIndex tree, where each node represents a fragment (a DFS code). For example, two discriminative fragments $f_1 = \langle e_1 \rangle$ and $f_3 = \langle e_1\ e_2\ e_3 \rangle$ are stored in the gIndex tree (for brevity, we use e_i to represent edges in the DFS codes). Although fragment $f_2 = \langle e_1\ e_2 \rangle$ is not a discriminative fragment, we have to store f_2 in order to connect fragments f_1 and f_3.*

Searching using the index

- The search uses the algorithm below but applies also the Apriori principle
- If a fragment is not in the Gindex, we need not search its super-graphs
- On Cq we perform the subgraph isomorphism



Algorithm 2 Search

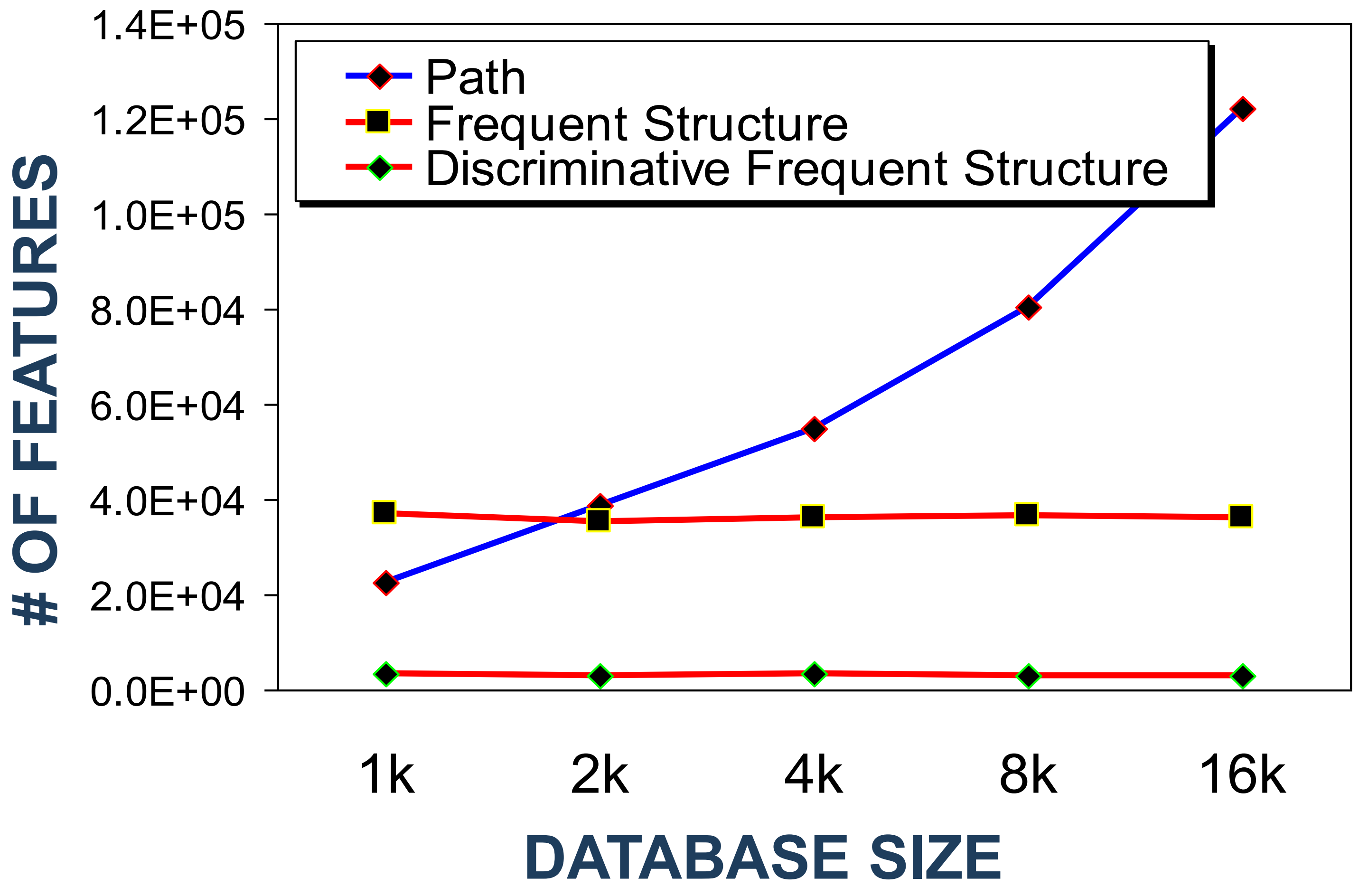
```
Input: Graph database $D$, Feature set $F$, Query $q$,
       and Maximum fragment size $maxL$.
Output: Candidate answer set $C_q$.

1: let $C_q = D$;
2: for each fragment $x \subseteq q$ and $len(x) \leq maxL$ do
3:     if $x \in F$ then
4:         $C_q = C_q \cap D_x$;
5: return $C_q$;
```

Experimental Setting

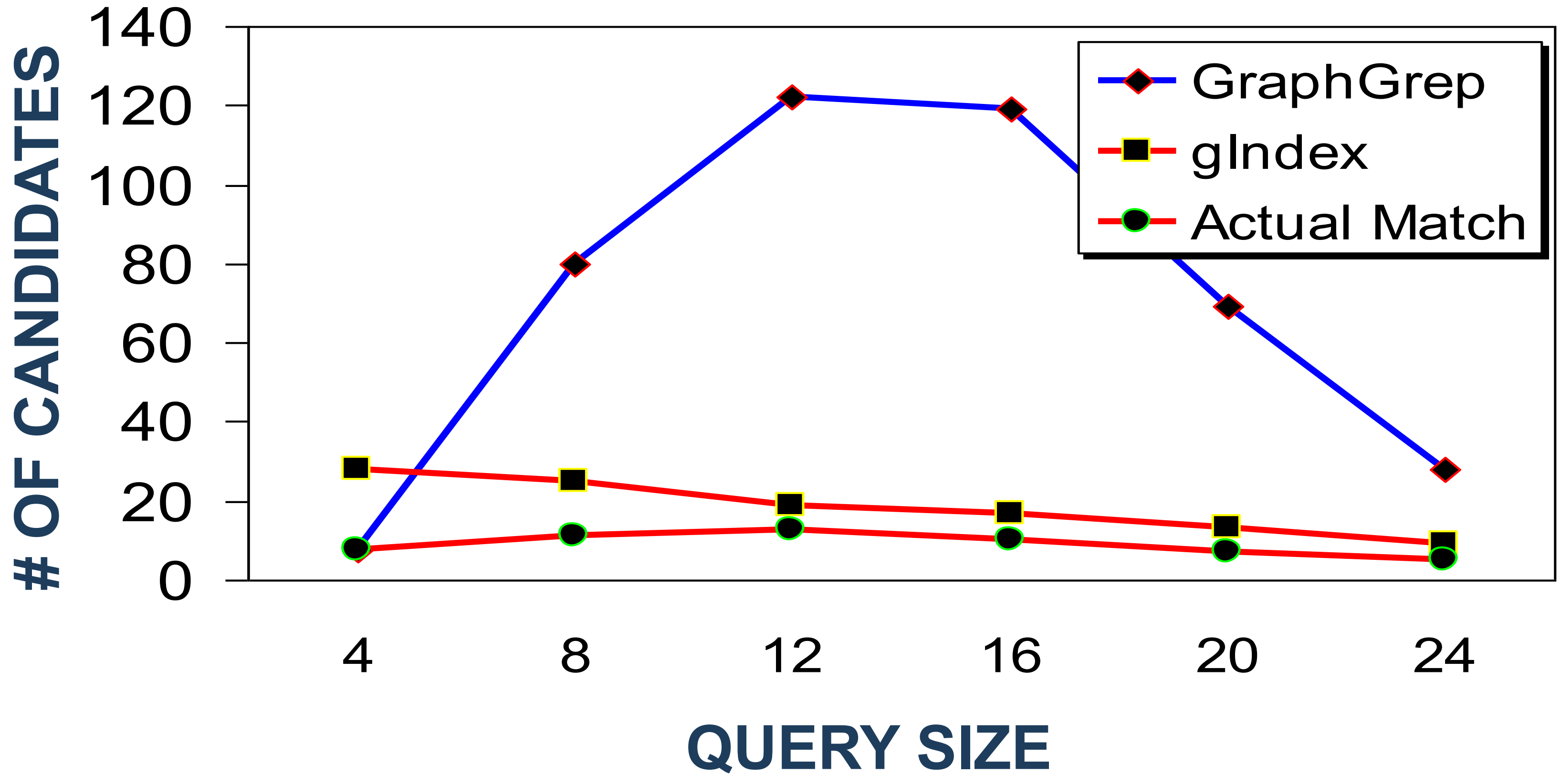
- The AIDS antiviral screen compound dataset from NCI/NIH, containing 43,905 chemical compounds
- Query graphs are randomly extracted from the dataset.
- GraphGrep: maximum length (edges) of paths is set at 10
- gIndex: maximum size (edges) of structures is set at 10

Experiments: Index Size

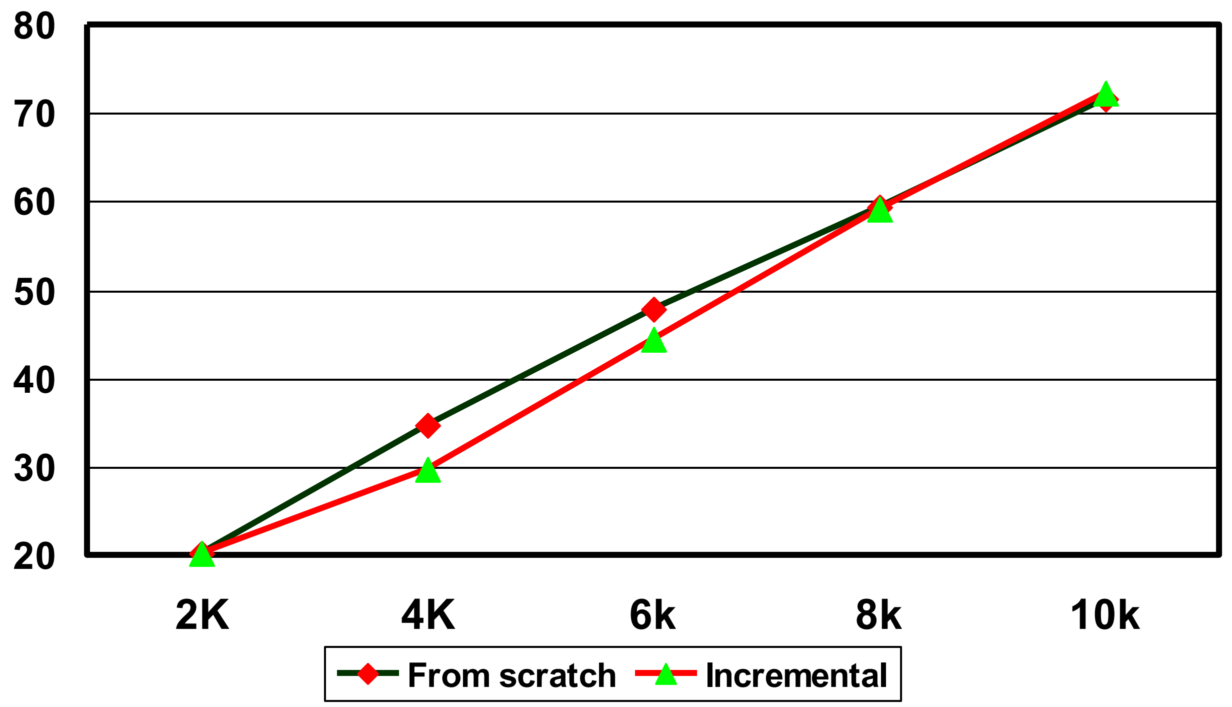


Note Gindex is very stable!

Experiments: Answer Set Size



Experiments: Incremental Maintenance



Frequent structures are stable to database updating

Index can be built based on a small portion of a graph database, but be used for the whole database

Final conclusions

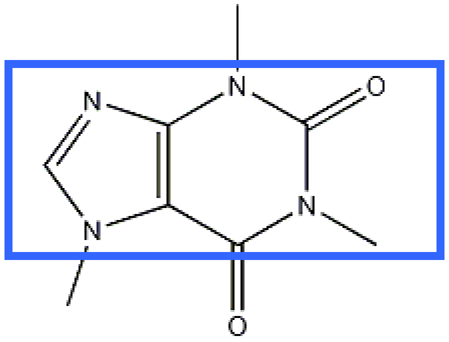
- The index size of gIndex is more than 10 times smaller than that of GraphGrep
- gIndex outperforms GraphGrep by 3 to 10 times in various query loads
- The index returned by the incremental maintenance algorithm is effective: it performs as well as the index computed from scratch provided the data distribution does not change much.

Outline

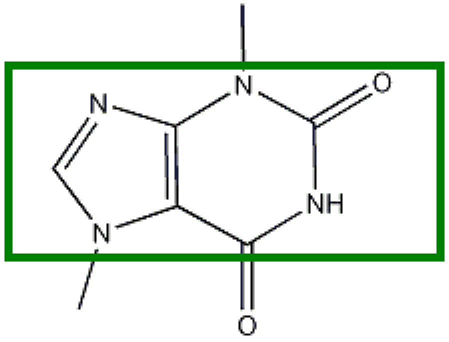
- Mining frequent graph patterns
- Constraint-based graph pattern mining
- Graph indexing methods
- Similairty search in graph databases
- Graph containment search and indexing

Structure Similarity Search

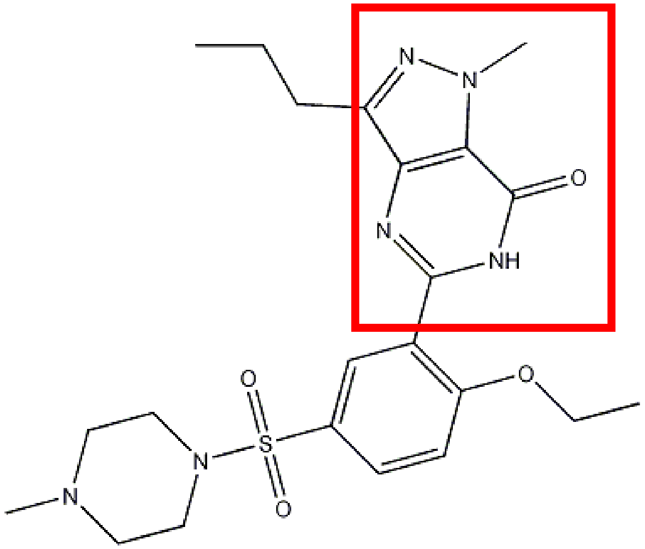
- **CHEMICAL COMPOUNDS**



(a) caffeine

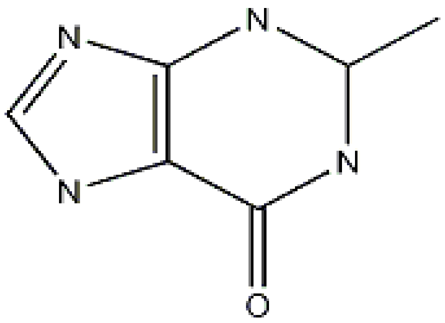


(b) diurobromine



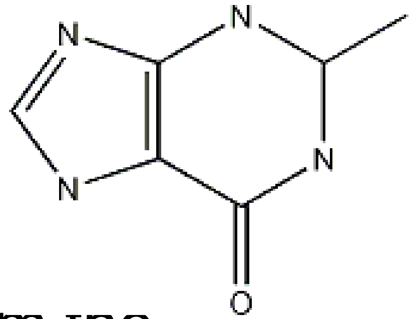
(c) viagra

- **QUERY GRAPH**



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

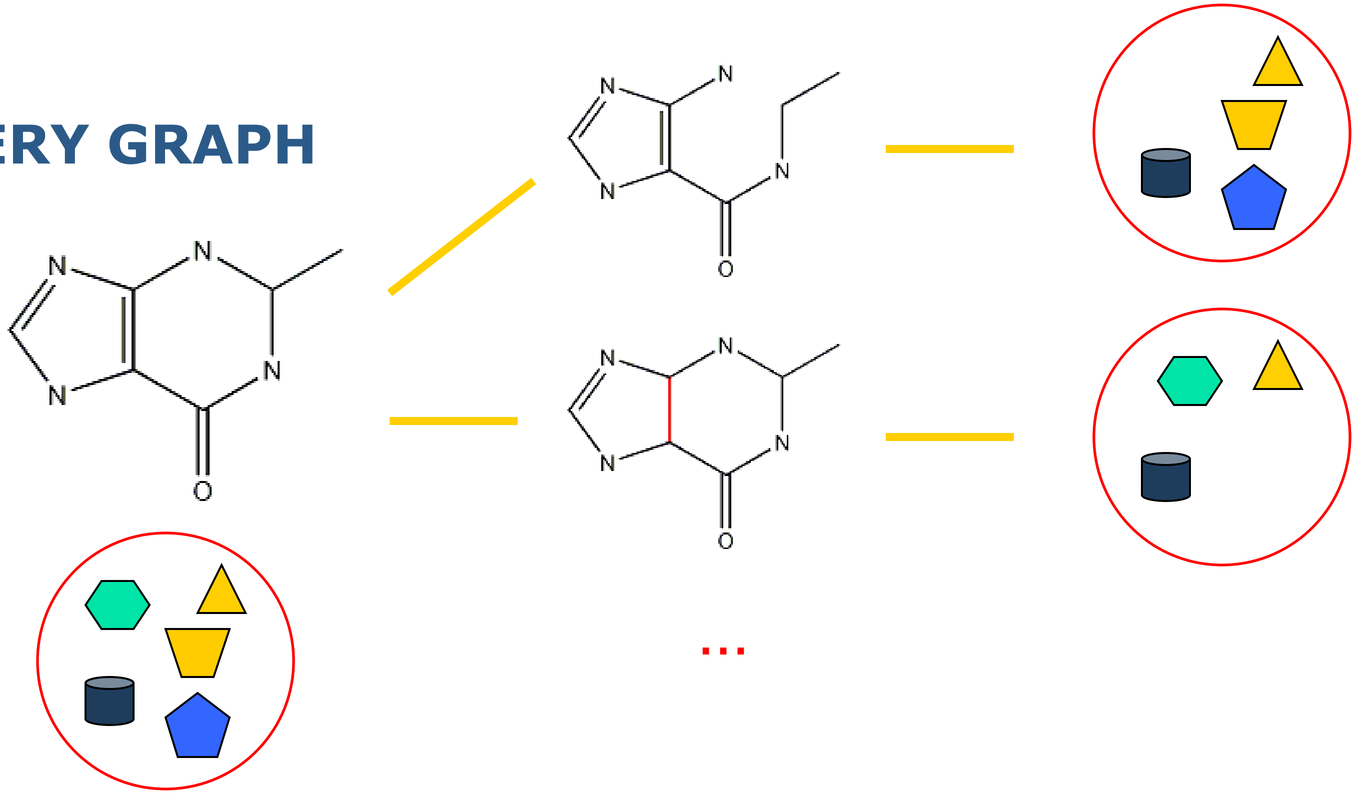
Index: Precise vs. Approximate Search

- Precise Search
  - Use frequent patterns as indexing features
  - Select features in the **database space** based on their selectivity
  - Build the index
- Approximate Search
  - Hard to build indices covering similar subgraphs—explosive number of subgraphs in databases
  - Idea: (1) keep the index structure

    (2) select features in the **query space**

Substructure Similarity Measure

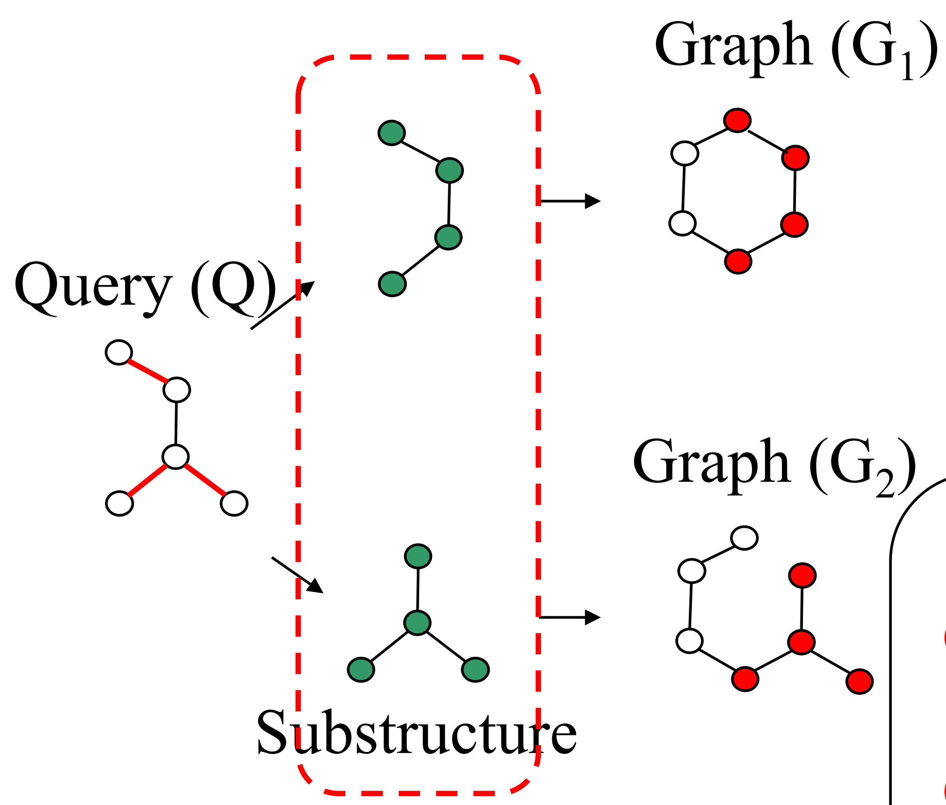
- **Query relaxation measure**
  - Number of edges can be relabeled or missed; but the positions of these edges are not fixed



Substructure 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

Intuition: Feature-Based Similarity Search



- If graph G contains the major part of a query graph Q, G should share a number of common features with Q

- Given a relaxation ratio, calculate the maximal number of features that can be missed !

At least one of them should be contained

Feature-Graph Matrix

graphs in database

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 and at most 2 features to miss due to the relaxation threshold

Edge Relaxation – Feature Misses

- If we allow k edges to be relaxed, J is the maximum number of features to be hit by k edges—it becomes the maximum coverage problem
- NP-complete
- A greedy algorithm exists

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

  - We design a heuristic to refine the bound of feature misses

Query Processing Framework

Step 1. Index Construction

- Select small structures as features in a graph database, and build the feature-graph matrix between the features and the graphs in the database

Step 2. Feature Miss Estimation

- Determine the indexed features belonging to the query graph
- Calculate the upper bound of the number of features that can be missed for an approximate matching, denoted by J
  - On the query graph, not the graph database

Step 3. Query Processing

- Use the feature-graph matrix to calculate the difference in the number of features between graph G and query Q, $F_G - F_Q$
- If $F_G - F_Q > J$, discard G. The remaining graphs constitute a candidate answer set

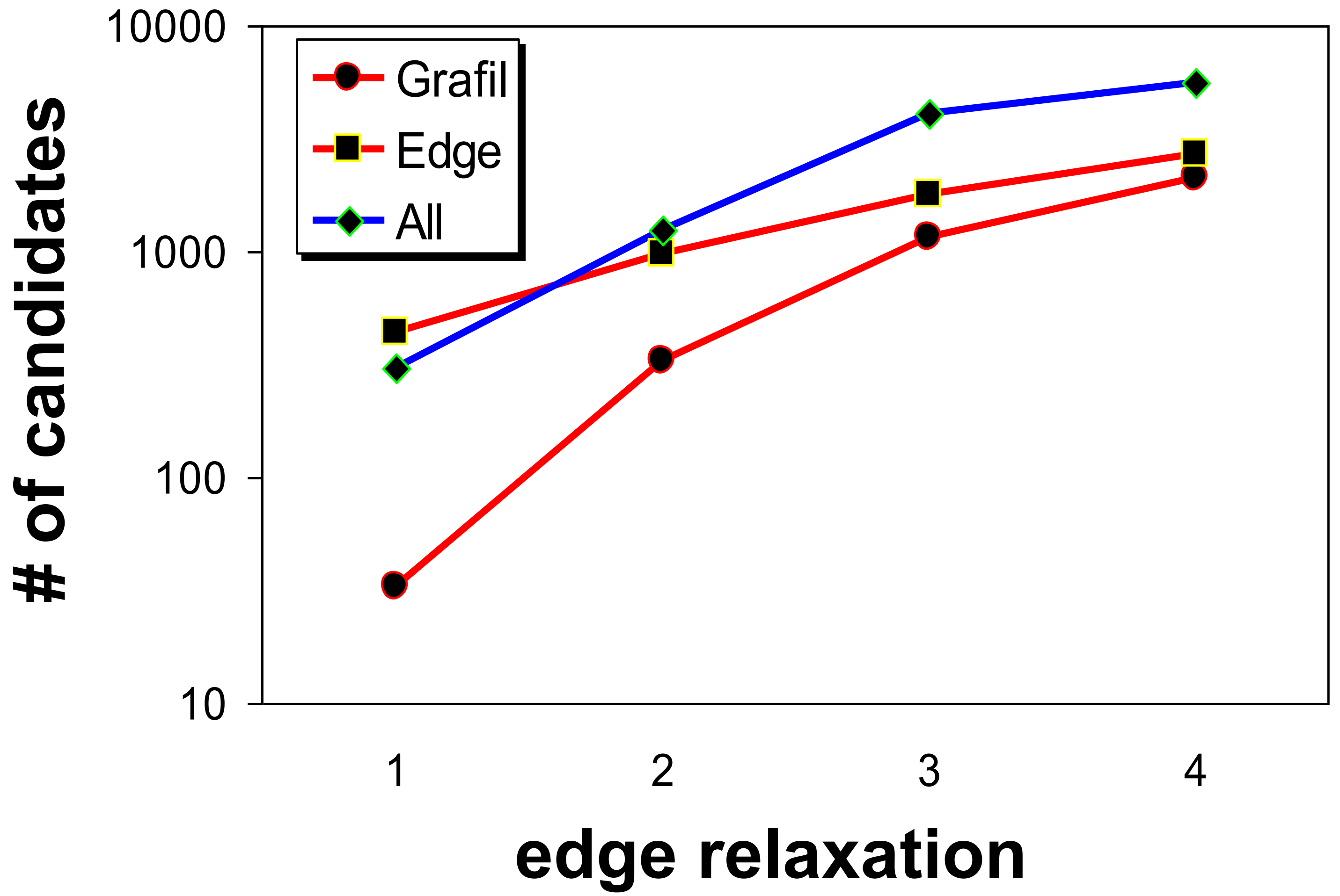
Approximate structures search

- Lets assume we found the “best” candidate set. What do we do now?
- Subgraph isomorphism is not good because we are looking for approximate structures
- The paper mentions several algorithms for approximate structure search
- The problem is in general NP-hard

Performance Study

- Database
  - Chemical compounds of Anti-Aids Drug from NCI/NIH, randomly select 10,000 compounds
- Query
  - Randomly select 30 graphs with 16 and 20 edges as query graphs
  - Competitive algorithms
    - Grafil: Graph Filter—our algorithm
    - Edge: use edges only
    - All: use all the features

Comparison of the Three Algorithms



Outline

- Mining frequent graph patterns
- Constraint-based graph pattern mining
- Graph indexing methods
- Similairty search in graph databases
- Graph containment search and indexing

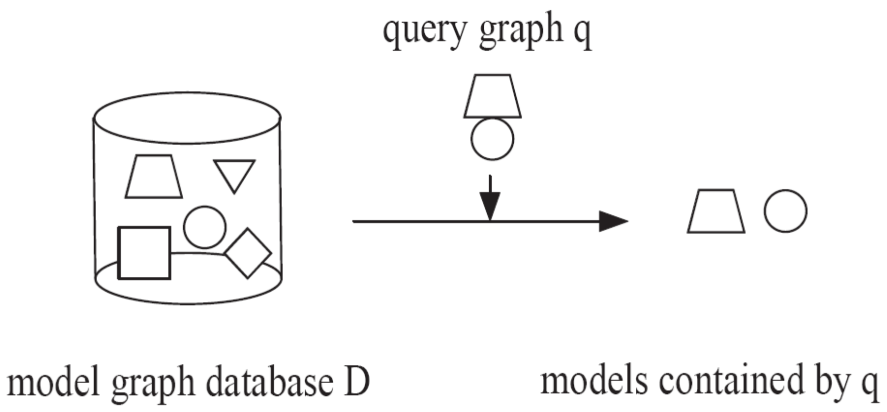
Towards Graph Containment Search and Indexing

Chen Chen, Xifeng Yan, Philip S. Yu, Jiawei Han,
Dong-Qing Zhang, Xiaohui Gu
University of Illinois at Urbana-Champaign
IBM T.J. Watson Research Center
Thomson - Images & Beyond

Graph Search in Two Directions

Given a graph database *D* and a query graph *q,*

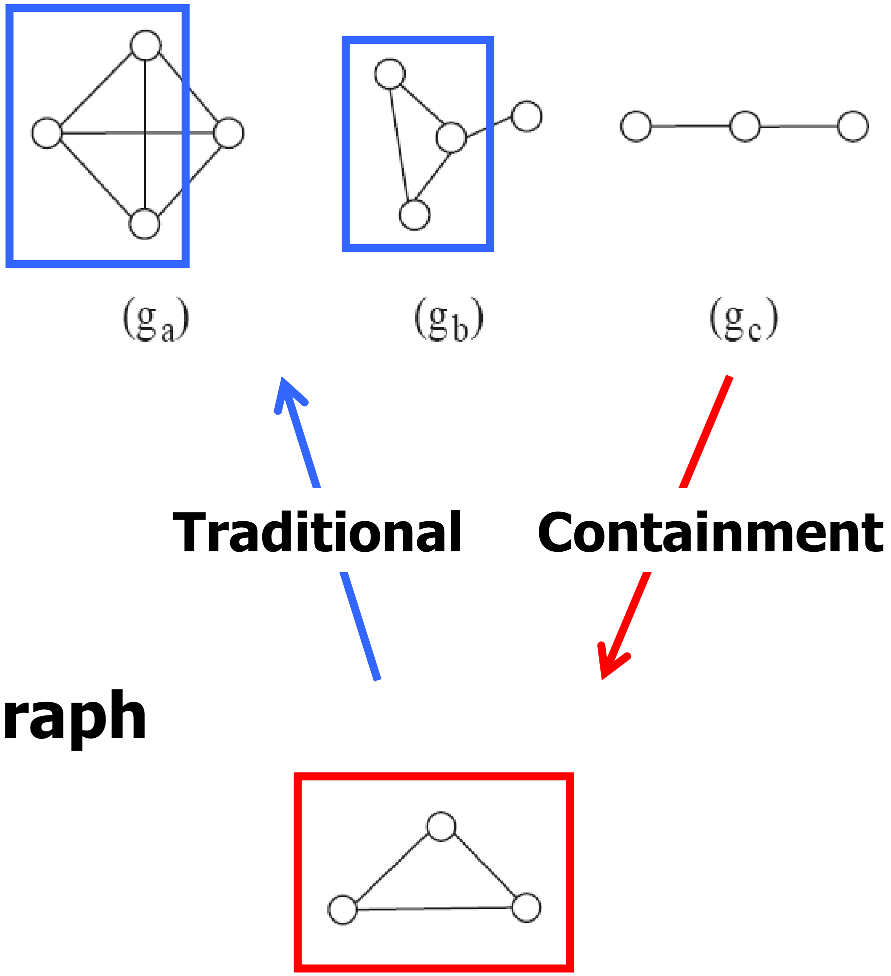
- *(Traditional) graph search:* Finds all graphs **containing** *q*
- *Graph containment search:* Finds all graphs **contained** by *q*



Example

- Graph Database

- Query Graph



Applications

- Chem-informatics: Searching for “descriptor” structures by full molecules
- Pattern Recognition: Searching for model objects by the captured scene
  - Attributed Relational Graphs (ARGs)
- Cyber Security: Virus signature detection
- …

Solution 0

- The Naïve SCAN approach
  - Load each database graph from the disk, and compare it with the query
- Disadvantages
  - For each entry in the database, one (NP-hard) subgraph isomorphism test is needed
  - I/O overheads
- We need Index!

Different Philosophies in Two Searches

- Graph search: **Feature-based pruning** strategy
  - Each query graph is represented as a vector of features, where features are subgraphs in the database
  - If a graph in the database contains the query, it must also contain all the features of the query
- Different logics: Given a data graph *g* and a query graph *q*,
  - (Traditional) graph search: **inclusion logic**

$$f \subset q, f \not\subset g \Rightarrow q \not\subset g$$

    - If feature *f* is <u>in</u> *q* then the graphs <u>not having</u> *f* are pruned
  - Graph containment search: **exclusion logic**

$$f \not\subset q, f \subset g \Rightarrow g \not\subset q$$

    - If feature *f* is <u>not in</u> *q* then the graphs <u>having</u> *f* are pruned

Contrast Features for C-Search Pruning

- Contrast Features: Those contained by many database graphs, but unlikely to be contained by query graphs
- Why contrast feature? —because they can prune a lot in containment search!
- Challenges: There are nearly infinite number of subgraphs in the database that can be taken as features
- Contrast features should be those contained in many database graphs; thus, we only focus on those **frequent subgraphs** of the database

The Basic Framework

- **Off-line index construction**
  - Generate and select a feature set *F* from the graph database *D*
  - For feature *f* in *F*, D_f records the set of graphs containing *f*, i.e., $D_f = \{g \mid f \subseteq g, g \in D\}$, which is stored as an inverted list on the disk
- **Online search**
  - For each indexed feature $f \in F$, test it against the query *q*, pruning takes place iff. *f* is not contained in *q*
  - Candidate answer set $C_q = D - \bigcup_{f \not\subset q,\, f \in F} {}_f$
- **Verification**
  - Check each candidate in C_q by a graph isomorphism test

Cost Analysis

- Given a query graph *q* and a set of features *F*, the search time can be formulated as

$$\boxed{\sum_{f \in \mathcal{F}} T(f,q)} + \boxed{\sum_{g \in \mathcal{C}_q} T(g,q)} + \boxed{T_{index}}$$

Neglected because ID-list operations are cheap compared to isomorphism tests between graphs

- A simplistic model: Of course, it can be extended

Feature Selection

- Core problem for index construction
- Carefully choose the set of indexed features *F* to maximize pruning capability,

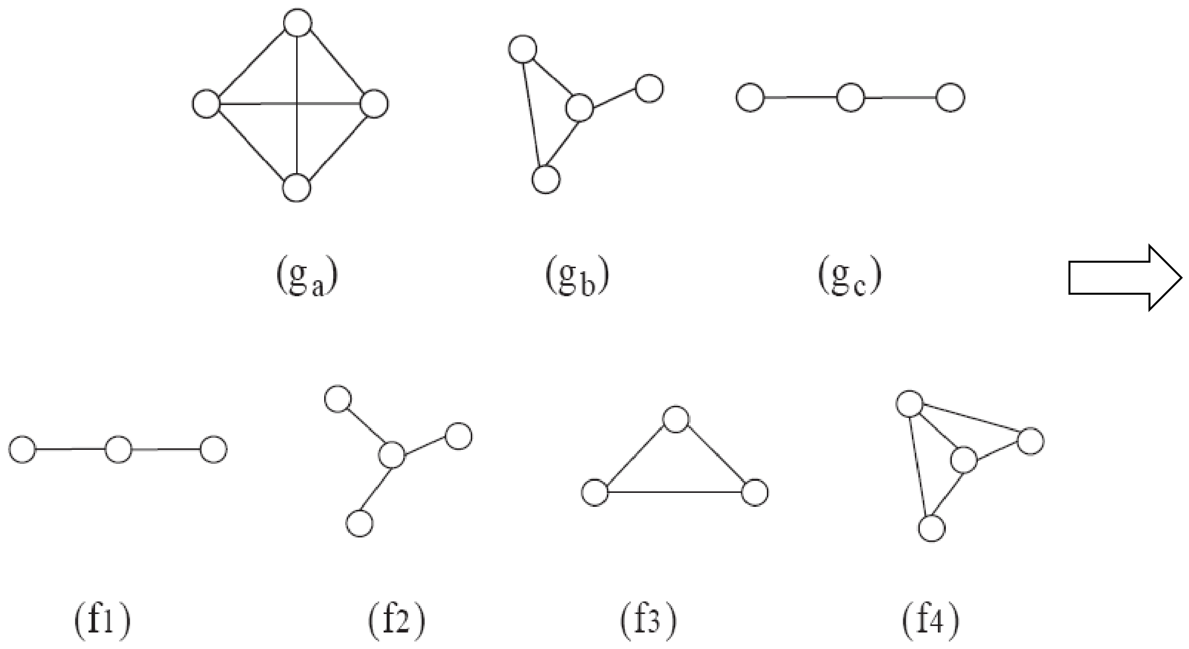
  i.e., minimize

$$\sum_{q \in Q} (|F| + |C_q|)$$

  for the query workload *Q*

Feature-Graph Matrix

- The *(i, j)*-entry tells whether the *j*$_{th}$ model graph has the *i*$_{th}$ feature, i.e., if the *i*$_{th}$ feature is not contained in the query graph, then the *j*$_{th}$ model graph can be pruned iff. the *(i, j)*-entry is 1



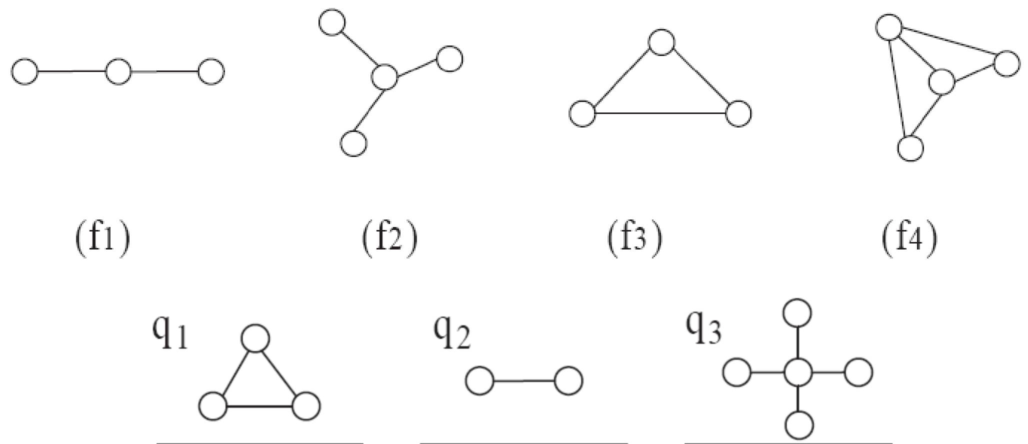
	g_a	g_b	g_c
f_1	1	1	1
f_2	1	1	0
f_3	1	1	0
f_4	1	0	0

Contrast Graph Matrix

- If the i_{th} feature is contained in the query, then the corresponding row of the feature-graph matrix is set to 0, because the i_{th} feature does not have any pruning power now

	g_a	g_b	g_c
f_1	1	1	1
f_2	1	1	0
f_3	1	1	0
f_4	1	0	0

$\Rightarrow$



q_1:

	g_a	g_b	g_c
f_1	0	0	0
f_2	1	1	0
f_3	0	0	0
f_4	1	0	0

q_2:

	g_a	g_b	g_c
f_1	1	1	1
f_2	1	1	0
f_3	1	1	0
f_4	1	0	0

q_3:

	g_a	g_b	g_c
f_1	0	0	0
f_2	0	0	0
f_3	1	1	0
f_4	1	0	0

Training by the Query Log

- Given a query log $L = \{q_1, q_2, . . . , q_r\}$, we can **concatenate** the contrast graph matrix of all queries to form a contrast graph matrix for the whole query set
- What if there are no query logs?
  - As the query graphs are usually not too different from database graphs, we can start the system by setting $L = D$, and then real queries will flow in
  - Our experiments confirm the effectiveness of this alternative

Maximum Coverage with Cost

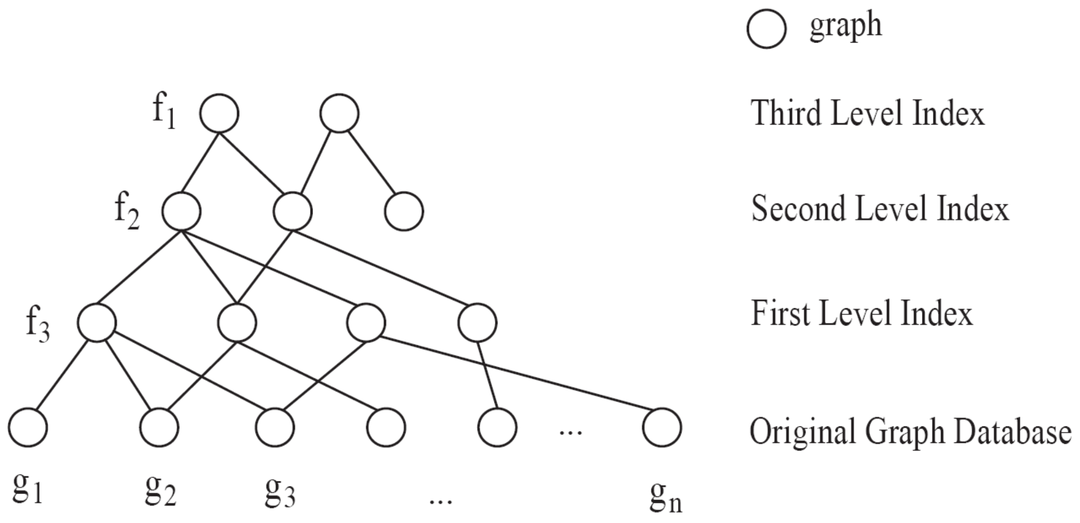
- Including the i_{th} feature
  - **Gain**: the sum of the i_{th} row, which is the number of (d-graph, q-graph) pairs it can prune
  - **Cost**: $|L| = r$, because for each query q, we need to decide whether it contains the i_{th} feature at first
- Select the optimal set of features that can maximize this gain-cost difference
  - Maximum Coverage with Cost
  - It is NP-complete (already the set-cover problem without cost…)

The Basic Containment Search Index

- Greedy algorithm
  - As the cost ($|L| = r$) is equal among features, the 1st feature is chosen as the one with greatest gain
  - Update the contrast graph matrix, remove selected rows and pruned columns
  - Stop if there are no features with gain over r
- cIndex-Basic
  - A redundancy-aware fashion
  - It can approximate the optimal index within a ratio of $1 - 1/e$

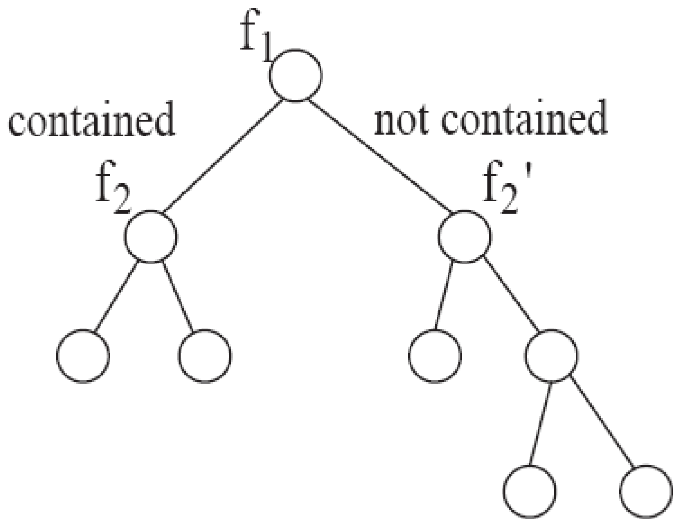
The Bottom-Up Hierarchical Index

- View indexed features as another database on which a second-level index can be built
- Iterate from the bottom of the tree
- The **cascading** effect: If f_1 is not contained in q, then the whole tree rooted at f_1 need not be examined



The Top-Down Hierarchical Index

- Strongest features are put on the top
- The 2nd test takes messages from the 1st test
- The **differentiating** effect: index different features for different queries

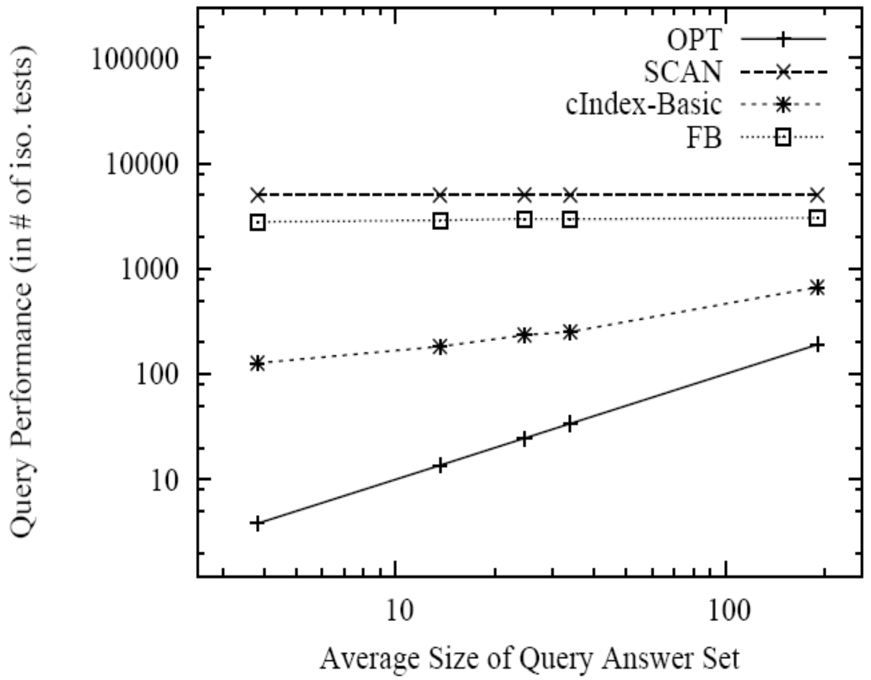


Experiment Setting

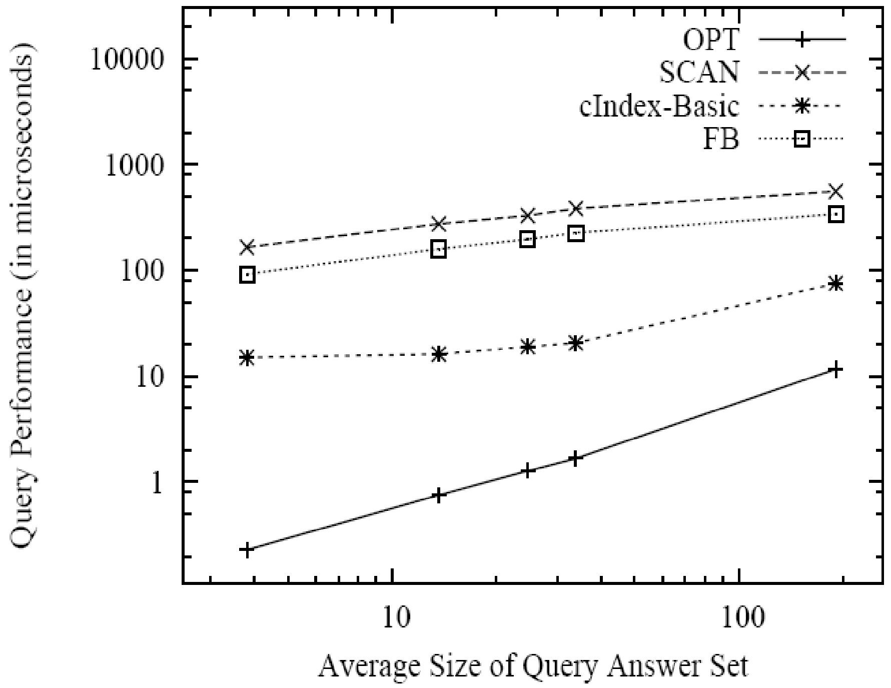
- Chemical Descriptor Search
  - NCI/NIH AIDS anti-viral drugs
  - 10,000 chemical compounds – queries
  - Characteristic substructures - database
- Object Recognition Search
  - TREC Video Retrieval Evaluation
  - 3,000 key frame images – queries
  - About 2,500 model objects – database
- Compare with:
  - Naïve SCAN
  - FB (Feature-Based): gIndex, state-of-art index built for (traditional) graph search
  - OPT: corresp. to search database graphs really contained in the query

Chemical Descriptor Search

In terms of iso. test #

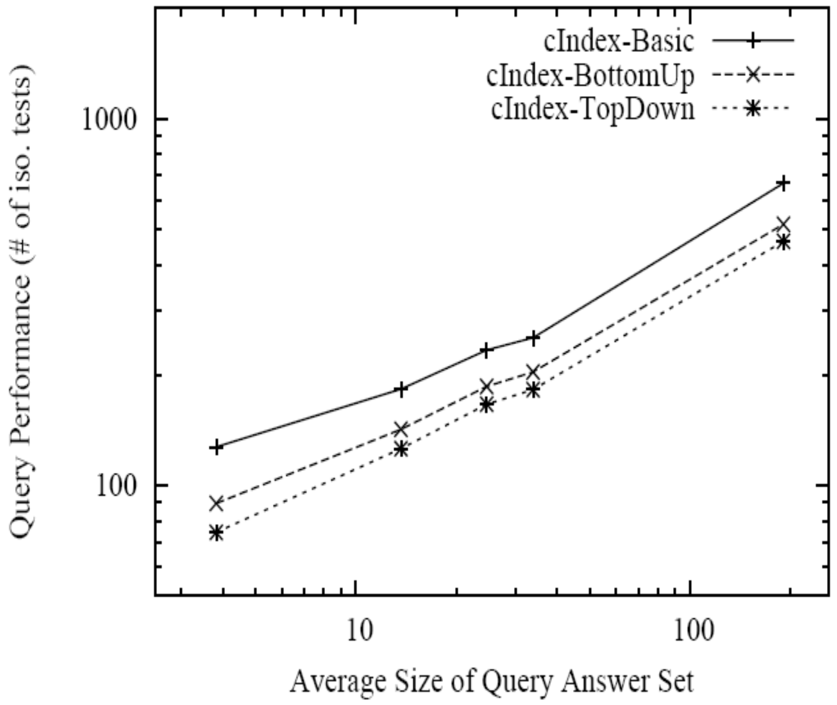


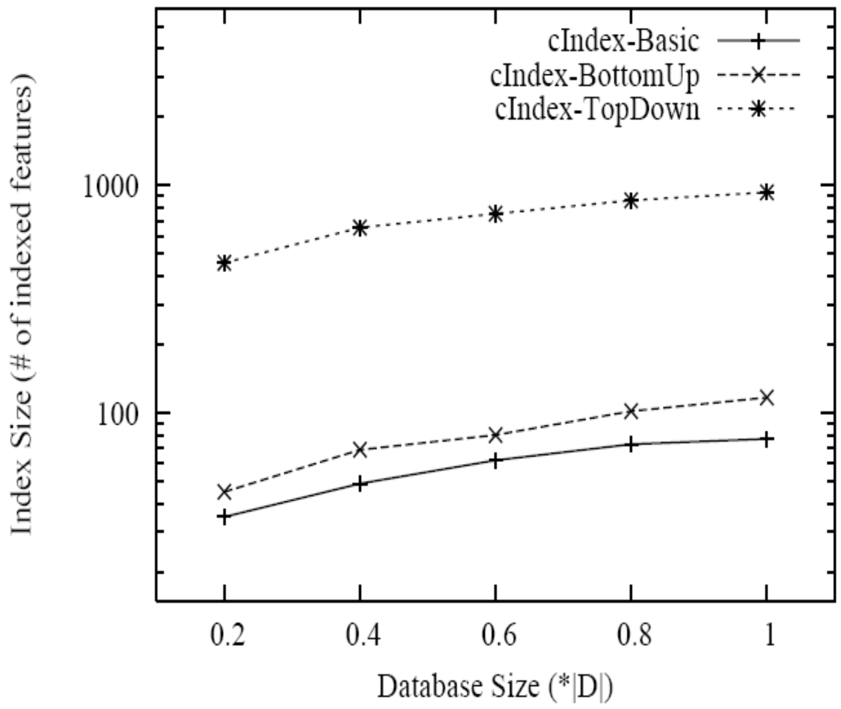
In terms of processing time



Trends are similar, meaning that our simplistic model is accurate enough

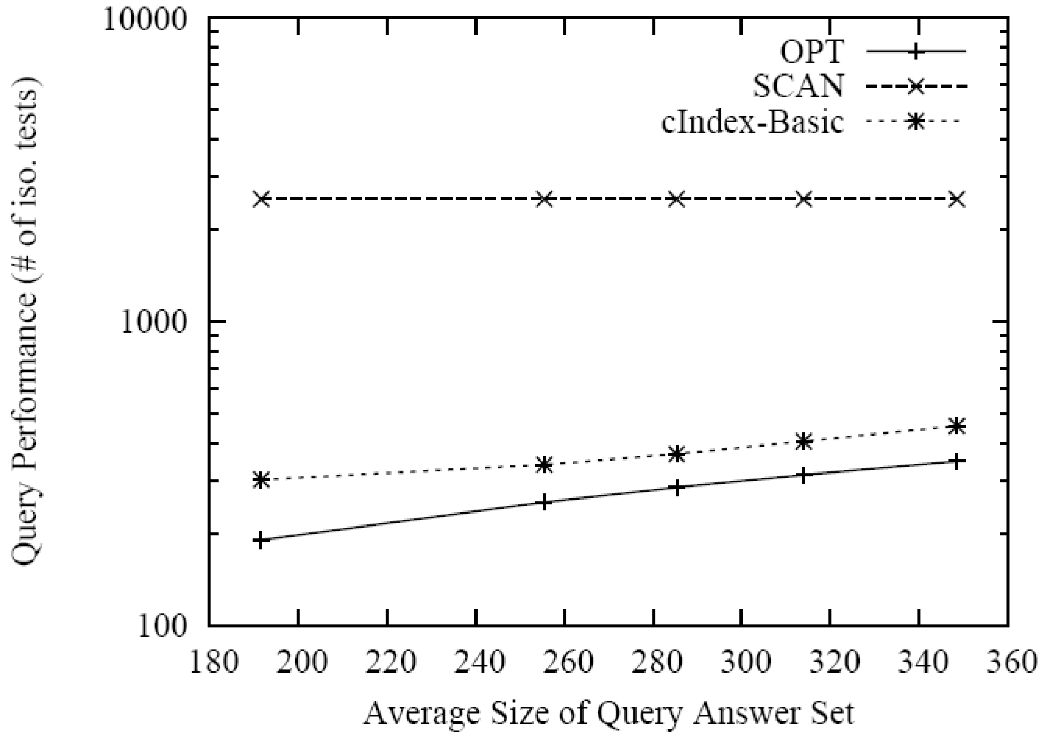
Hierarchical Indices





Space-time tradeoff

Object Recognition Search



Graph Containment Summary

- We study containment graph search, where (traditional) graph index is not applicable
- We propose the contrast feature-based indexing model, prove its usefulness in this new scenario, both theoretically and empirically
- Our method is not only valuable for graph search, but also useful for any data with transitive relation

Connection subgraphs

- We define a *connection subgraph as a small subgraph of a* large graph that best captures the relationship between two or more query nodes.
- The primary motivation for this work is to provide a paradigm for exploration and knowledge discovery in large social networks graphs, but also in biological and other domains
- The main problem is characterizing the importance of the nodes in the connection subgraph

Connection subgraphs

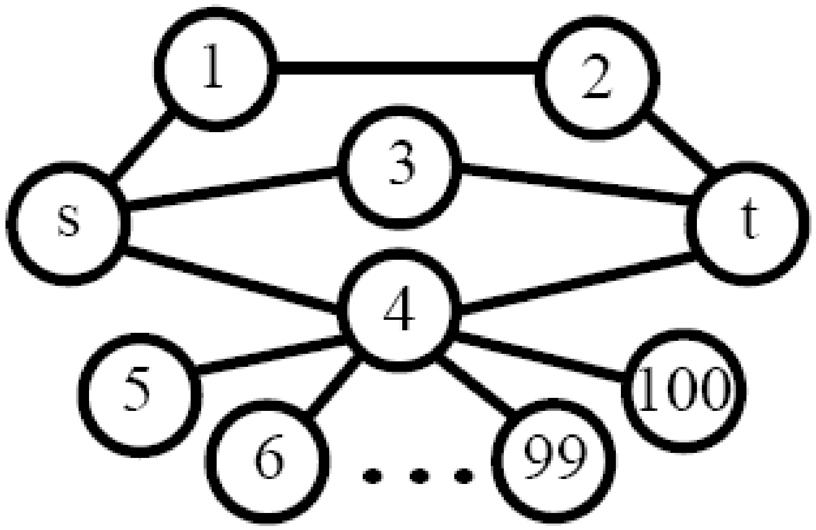
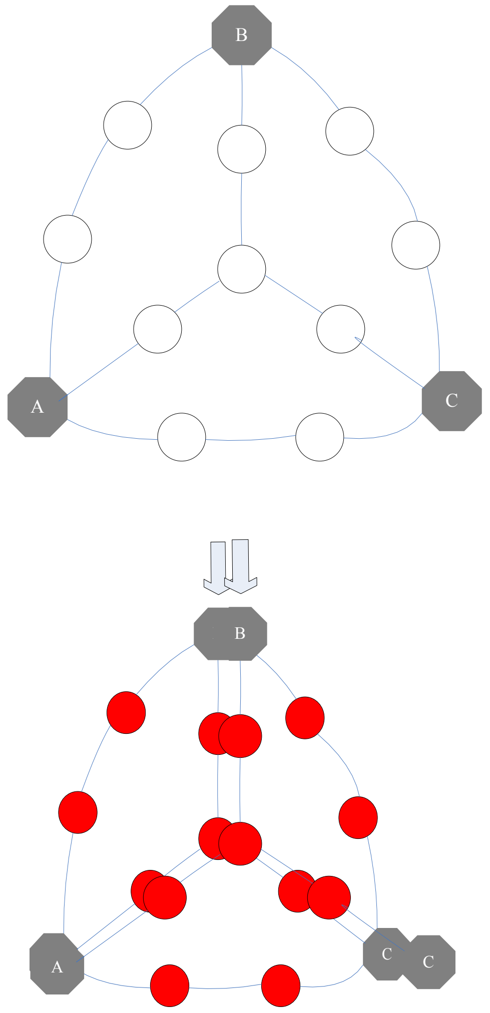


Figure 2: A simple network where both shortest path and network flow fail to adequately model social relationships. With all edges having weight 1, flow fails to distinguish between the paths s,1,2,t and s,3,t, even though the latter is shorter. Total path length fails to distinguish between the paths s,3,t and s,4,t, even though path through 4 is diluted by many extra connections.

Center-Piece Subgraph(Ceps)

- Given Q query nodes
- Find Center-piece$\leq$(b)

- Input of Ceps
  - Q Query nodes
  - Budget b
  - k softAnd number
- App.
  - Social Network
  - Law Inforcement
  - Gene Network
  - …



Challenges in Ceps

- **Q1: How to measure importance?**

- (Q2: How to extract connection subgraph?
- Q3: How to do it efficiently?)

Challenges in Ceps

- **Q1: How to measure importance?**
- A: “proximity” – but how to combine scores?
- (Q2: How to extract connection subgraph?
- Q3: How to do it efficiently?)

Paper by Faloutsos et. al

Conclusions

- Graph mining has wide applications
- Frequent and closed subgraph mining methods
  - gSpan and CloseGraph: pattern-growth depth-first search approach
- gPrune: Pruning graph mining search space with constraints
- gIndex: Graph indexing
  - Frequent and discirminative subgraphs are high-quality indexing fatures
- Grafill: Similairty (subgraph) search in graph databases
  - Graph indexing and feature-based approximate matching
- cIndex: Containment graph indexing
  - A contrast feature-based indexing model
- Connection subgraphs

Research Papers Covered in this Talk

- X. Yan, P. S. Yu, and J. Han, ***Graph Indexing: A Frequent Structure-based Approach***, *SIGMOD'04 (also in TODS'05, Google Scholar: ranked #1 out of 63,300 entries on "Graph Indexing")*
- X. Yan, P. S. Yu, and J. Han, "***Substructure Similarity Search in Graph Databases***", SIGMOD'05 *(also in TODS'06)*
- C. Chen, X. Yan, P. S. Yu, J. Han, D. Zhang, and X. Gu, "***Towards Graph Containment Search and Indexing***", VLDB'07, Vienna, Austria, Sept. 2007
- Christos Faloutsos, Kevin S. McCurley, Andrew Tomkins: Fast discovery of connection subgraphs. KDD 2004: 118-127