Graph Indexing

Davide Mottin, Konstantina Lazaridou

Hasso Plattner Institute

Graph Mining course Winter Semester 2016

HPI-Kolloquium – Graph summarization

Prof. Danai Koutra

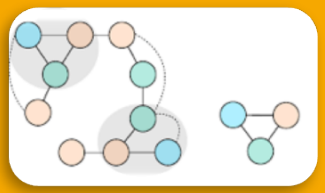
Networks naturally capture a host of real-world interactions, spanning from social interactions to brain activity. But, given a massive graph (e.g., a large email exchange network), what can be learned about its structure? What are its "important" patterns?

In this talk I will present our work on **scalable algorithms** that help us to make sense of **large graph** data during early **exploratory analysis**. I will focus on two approaches: “VoG” and “TimeCrunch”. VoG disentangles the complex graph connectivity patterns, and efficiently summarizes large graphs with important and semantically meaningful structures by leveraging information-theoretic methods. TimeCrunch discovers coherent **temporal patterns**, and **summarizes time-evolving networks** in a scalable and effective way. Both methods provide key insights into large real-world graphs.

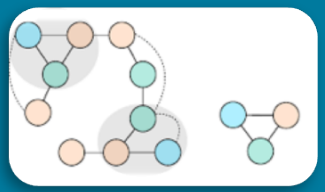
I will conclude the talk by presenting "**Perseus**", an interactive large-scale graph mining and visualization tool that can be used to obtain a different type of summaries that consist of graph properties (or statistics).

Thursday Nov 17
16.00 – 17.00 – HS1

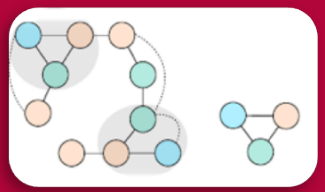
Lecture road



Graph Indexing



Feature-based Graph Index

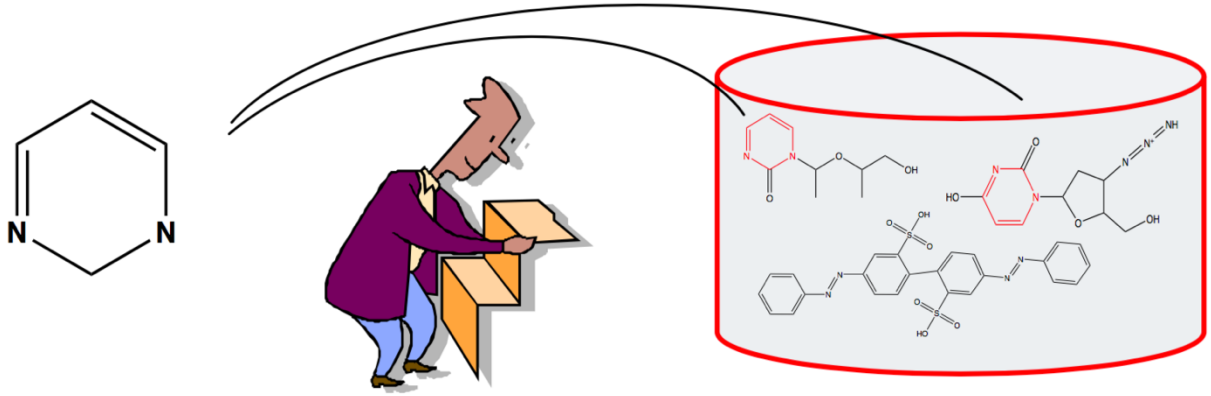


Structure Similarity Search

Graph Search

Querying a set of graphs
Given a set of graphs $D = \{G_1, \dots, G_n\}$ and a query graph Q, find all graphs $D_Q = \{G | Match(Q, G) = 1, G \in D\}$, where $Match$ is a boolean function.

- *Match* is a function:
  - Subgraph isomorphism for substructure search
  - Subgraph approximate match for substructure similarity search



Query

Set of graphs (or graph database)

Scalability issues

- Require to process large graphs/set of graphs
- Structure search is hard (**NP**-complete for subgraph isomorphism)
- Graph search need to be fast
- Requires a lot of Disk I/Os

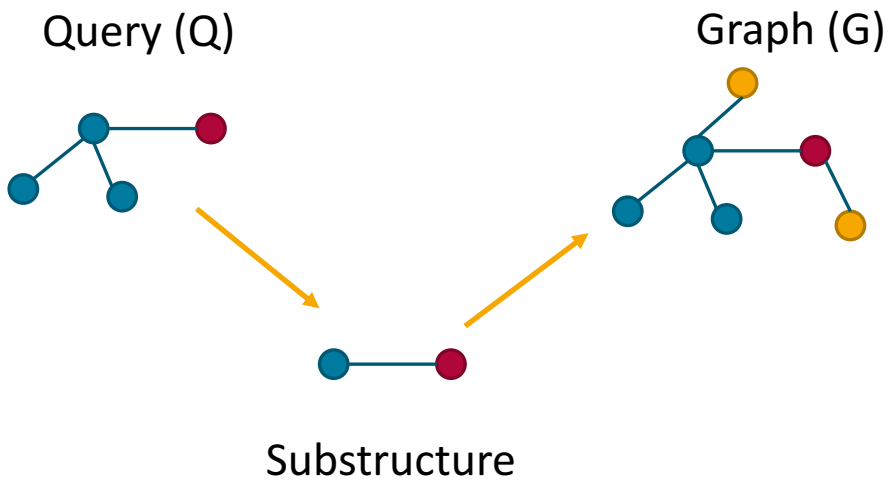
Need for efficient structures to summarize the graph and process the queries easily

These structures are called **Graph Indexes**

Indexing strategies

- Graph embedding
  - Represent each graph as a vector of feature and use high-dimensional indexes such as R-trees [Petrakis et al., 1997]
  - Use eigenvalues of adjacency matrices as signatures [Shokoufandeh et al., 1999]
- Metric indexing
  - Organizes graphs hierarchically according to their mutual distances [Berretti et al., 2001]
  - Use maximum description length to compress [Holder et al., 1994]
- Feature-based graph Index
  - Represent graphs as set of features (substructures) and index them.

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 these substructures (conservative strategy)

Graph search with indexes

- Two step in processing graph queries

Step 1. Index Construction

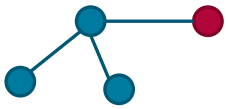
- Enumerate structures in the graph database, build an inverted index between structures and graphs

Step 2. Query Processing

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

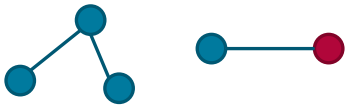
Graph search with indexes (2)

Query (Q)



(1) Search

Features



f1 f2

$$C_Q = \bigcap_{f \sqsubseteq Q, f \in F} D_f$$

Candidate set: set of graphs containing all the features
- F is the set of features

(2) Fetching

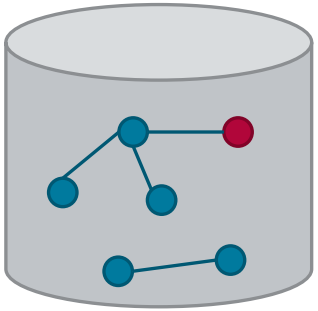
Retrieve the candidate graphs from the disk

(3) Verification

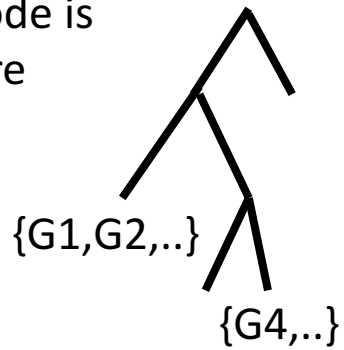
Check if the candidates satisfy the query

Query processing

Database
D={G_1, … G_n}



Each node is a feature



Index construction

Cost Analysis

- Query response time

$$T_{index} + |C_Q| * (T_{io} + T_{isomorphism_testing})$$

Disk I/O time

Retrieve index into main memory

Number of candidate graphs

Isomorphism testing time

Remarks

- Time for isomorphism test does not change much with queries
- Indexing time is important if the index does not fit into memory
- Make $|C_Q|$ as small as possible

Indexes try to reduce as much as possible this set

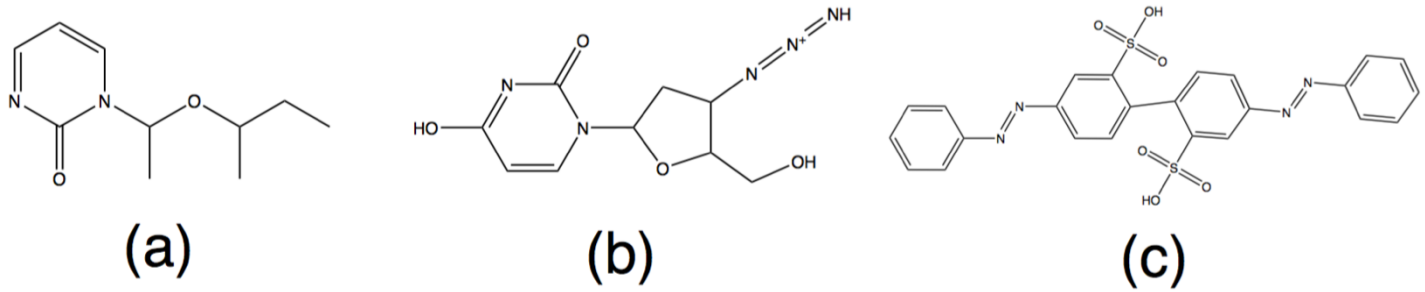
Types of features

Different approaches use different types of features:

1. Paths
   - 👍 easy to compute and to manipulate
   - 👎 generate many false positive candidates
2. Trees
   - 👍 Easier to manipulate than subgraphs, and more efficient
   - 👎 Generate more false positives than subgraphs
3. Subgraphs
   - 👍 Generate fewer candidates
   - 👎 Complex structures, generate bigger indexes

Path-based Approaches

[James et al. 2003, Shasha et al. 2002]



1. Enumerate paths of a specific length

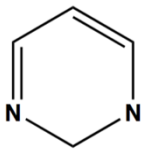
0-length: C, O, N, S
1-length: C-C, C-O, C-N, N-N, S-O
2-length: C-C-C, C-O-C, C-N-C, …
3-length: ...

2. Build an inverted index between paths an graphs

$S_C = \{a, b, c\}, S_O = \{a, b, c\}$
$S_{C-C} = \{a, b, c\}, S_{C-N} = \{a, b, c\}, \ldots$
$S_{C-C-C} = \{a, b\}, \ldots$

Path-based Approaches (2)

- Given a query



- Decompose it into paths and compute the intersection among candidate sets
  - $S_C = \{a, b, c\}, S_N = \{a, b, c\}$
  - $S_{C-C} = \{a, b, c\}, S_{C-N} = \{a, b, c\}, \ldots$
  - $S_{C-N-C} = \{a, b\}, \ldots$
- The intersection is {a,b}
- Retrieve graphs a,b from the disk and verify if they contain the query graph

Types of features

Different approaches use different types of features:

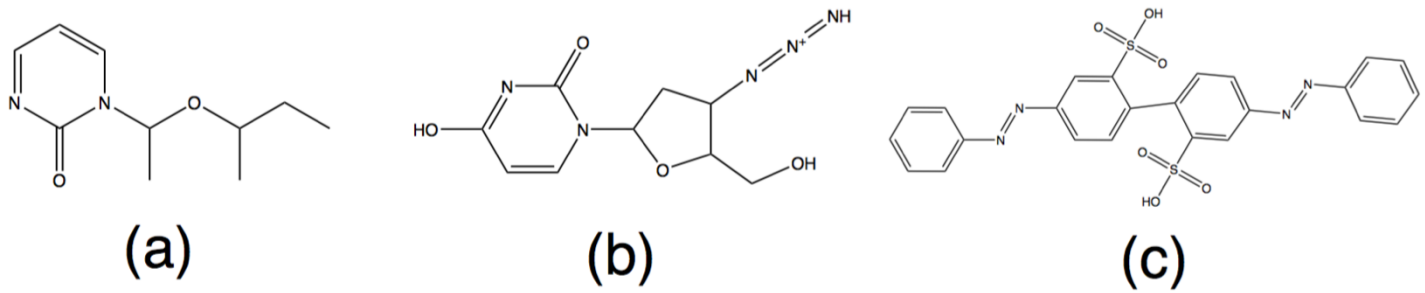
1. Paths (GraphGrep)
   - 👍 easy to compute and to manipulate
   - 👎 generate many false positive candidates
2. Trees (Tree+Δ, GCoding, GString)
   - 👍 Easier to manipulate than subgraphs, and more efficient
   - 👎 Generate more false positives than subgraphs
3. Subgraphs (gIndex, C-Tree, GDIndex, FG-Index, $\text{Turbo}_{\text{ISO}}$)
   - 👍 Generate fewer candidates
   - 👎 Complex structures, generate bigger indexes

gIndex: a subgraph-based approach

- Identify frequent structures in the database, the frequent structures are subgraphs that appear 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

Yan, X., Yu, P.S. and Han, J. Graph indexing: a frequent structure-based approach. SIGMOD, 2004.

Discriminative features



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

Good news
Discriminative features are two orders of magnitude less than normal subgraph features, and one order of magnitude less than frequent subgraphs!

What is discriminative?

- Mine not only frequent structures but also useful
  - Given a set of features $f_1, f_2, \dots, f_n$ and a new structure x measure the probability of reconstructing x having already indexed $f_1, f_2, \dots, f_n$.

$$P(x|f_1, f_2, \dots, f_n), f_i \sqsubseteq x$$

What is the advantage of indexing x? When P is small enough, x is a discriminative feature and should be included in the index

$$\gamma_x = \frac{1}{P(x|f_1, f_2, \dots, f_n)} = \frac{\left|\bigcap_i D_{f_i}\right|}{|D_x|}$$

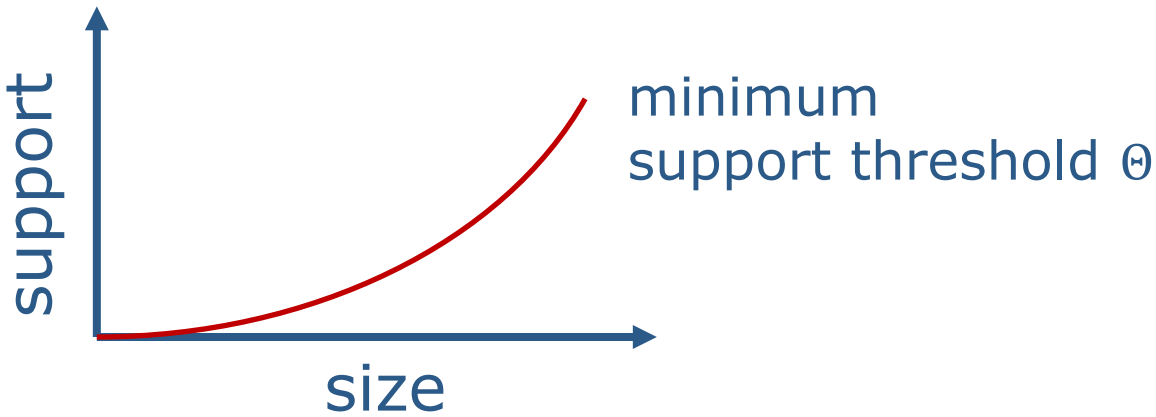
γ_x is called *discriminative ratio* of x.

A feature x is discriminative is $\gamma_x \geq \gamma_{\min}$

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
- Use a monotonically nondeacreasing function ψ, that given a feature size returns the support threshols.
  - A feature f is frequent if and only if $|D_f| \geq \psi(size(f))$

Exponential



Reducing the number of intersections

- How to reduce the number of times a feature x is checked in $C_Q \cap D_x$?
- **Remark**: if a feature $f_x \sqsubset f_y$ than $D_{f_y} \subseteq D_{f_x}$.
- Therefore $C_Q \cap D_{f_y} \cap D_{f_x} = C_Q \cap D_{f_y}$ and we can skip the comparison with f_x
- **Solution:** check features in a top-down fashion enumerating subgraphs of the query from the bigger to the smaller!

Types of features

Different approaches use different types of features:

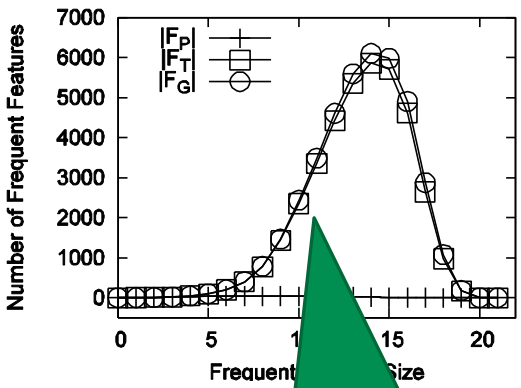
1. Paths
   - 👍 easy to compute and to manipulate
   - 👎 generate many false positive candidates
2. Trees
   - 👍 Easier to manipulate than subgraphs, and more efficient
   - 👎 Generate more false positives than subgraphs
3. Subgraphs
   - 👍 Generate fewer candidates
   - 👎 Complex structures, generate bigger indexes

Why trees?

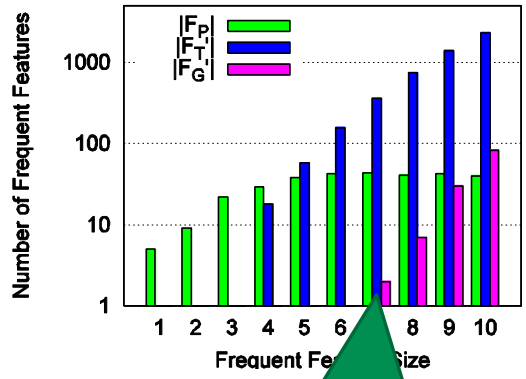
- Tree features are easier to compare
  - subgraph isomorphism can be polynomial on ordered trees
- Tree features are more expressive than paths
  - Paths generate more candidates than trees since they are less restrictive
- Most of the discovered frequent patterns are trees!!!
  - Frequent tree-features and graph-features share similar distributions and frequent tree-features have similar pruning power like graph-features
- Tree mining can be done much more efficiently than graph mining on G

Size of the feature vs number of features for each feature type

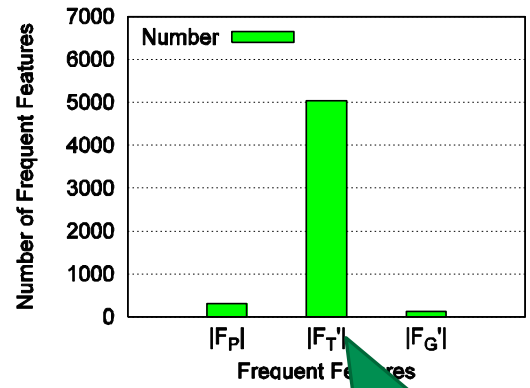
$$|F_{T'}| = |F_T| - |F_P|$$
$$|F_{G'}| = |F_G| - |F_T|$$



Tree and graph features have almost the same distribution



After some point you discover no new path



Most of the features are trees

Intuitions

- All subtrees of a frequent graph are frequent
- There is little chance that subtrees of frequent graph g coincide with those of frequent graph g', due to the structural diversity and label variety

The feature selection cost: Cfs

- Given a graph database, G, and a minimum support threshold, $\min_supp$, to discover the frequent feature set F (FP / FT / FG) from G

	Path	Tree	Graph
Isomorphism	$O(n)$	$O(n)$	P or NPC (?)
Sub-Isomorphism	$O(n + m)$	$O(m^{3/2}n/\log m)$	NP-Complete

- Tree
  - A good compromise between
    - the more expressive, but computationally harder general graph
    - the faster but less expressive path
  - Specialization of general graph avoiding undesirable theoretical properties and algorithmic complexity incurred by graph

Tree + Δ

- Tree-based Graph Indexing
- Δ : On-demand select a small number of discriminative graph-features without conducting costly graph mining beforehand
- Orders of magnitude smaller in index size, but performs much better than existing approaches in indexing construction and query processing

Zhao, P., Yu, J.X. and Yu, P.S. Graph indexing: tree+ delta<= graph. PVLDB, 2007.

Candidate answer set Size: $|C_Q|$

- Pruning power $power(f)$ of a frequent feature f is

$$power(f) = \frac{|D| - |D_f|}{|D|}$$

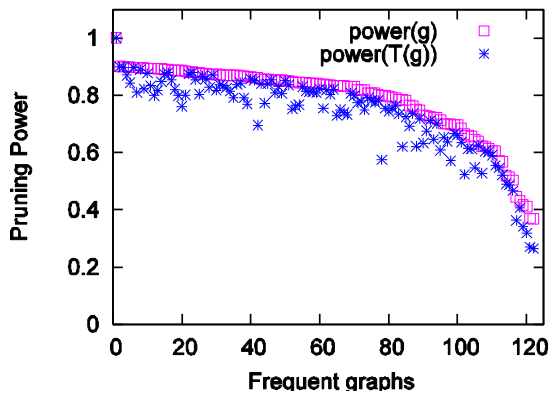
- The pruning power of a frequent feature set $S = \{f_1, f_2, \dots, f_n\}$

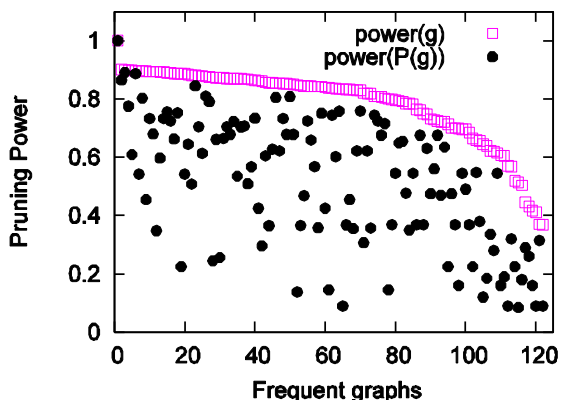
$$power(S) = \frac{|D| - \left|\bigcap_{i=1}^{n} D_{f_i}\right|}{|D|}$$

- Theorem 1: Given a frequent graph-feature g, and let its frequent sub-tree set be $T(g) = \{t_1, \dots, t_n\}$. Then, $power(g) \geq power(T(g))$
- Theorem 2: Given a frequent tree-feature t, and let its frequent sub-path set be $P(t) = \{p_1, \dots, p_m\}$. Then, $power(t) \geq power(P(t))$

Pruning Power

- The pruning power of all frequent subtree features, $T(g)$, of a frequent graph-feature *g* can be similar to the pruning power of *g*
- There is a big gap between the pruning power of a graph-feature *g* and that of all its frequent sub-path features, *P*(*g*)





Discriminative Graph Features Δ

- Consider a query graph *q* which contains a subgraph *g*
  - If $power(T(g)) \approx power(g)$, there is no need to index the graph-feature *g*, because its subtrees jointly have the similar pruning power
  - if $power(g) \gg power(T\ (g))$, it will be necessary to select *g* as an index feature because *g* is more *discriminative* than $T(g)$, in terms of pruning
- Discriminative graph-features (*w.r.t.* its subtree-features, controlled by ε_0) are selected from queries on-demand, without mining the whole set of frequent graph-features from D beforehand
  - Discriminative graph-features are used as additional indexing features, denoted Δ, which can also be reused further to answer subsequent queries

Discriminative Graph Selection

- The occurrence probability of g in the graph database D is

$$P(g) = \frac{|D_g|}{|D|} = \sigma_g$$

- the conditional occurrence probability of g' $\sqsupseteq g$, models the probability to select g' from D in the presence of g

$$P(g'|g) = \frac{P(g \wedge g')}{P(g)} = \frac{P(g')}{P(g)} = \frac{|D_{g'}|}{|D_g|}$$

- The upper (similarly, the lower) bound of $P(f'|f)$

Expressed only in terms of $T(g')$!!!

$$P(g'|g) = \frac{|D_{g'}|}{|D_g|} \leq \frac{|D| - \frac{|D| - |D_{T(g')}|}{1 - \epsilon_0}}{\sigma|D|} = \frac{\sigma_{T(g')} - \epsilon_0}{(1 - \epsilon_0)\sigma}$$

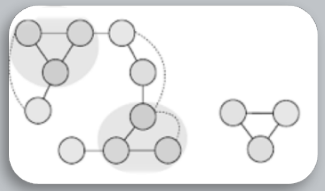
- The conditional occurrence probability of $P(g'|g)$, is solely upper-bounded by $T(g')$

Generic indexes

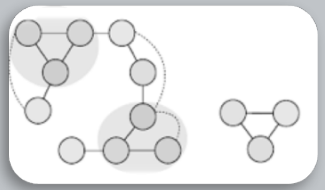
- Lindex is an index that works on any kind of feature (paths, graphs, stars, trees)
- Organize index features in a lattice (directed acyclic graph)
- Use set theory and graph theory to fast prune the graph

Yuan, D. and Mitra, P. Lindex: a lattice-based index for graph databases. *VLDBJ, 2013*.

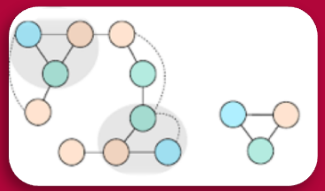
Lecture road



Graph Indexing



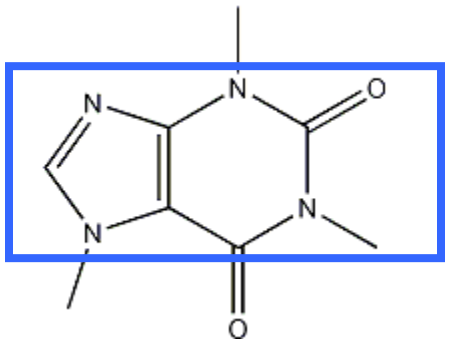
Feature-based Graph Index



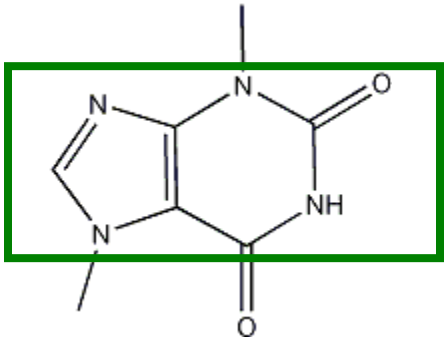
Structure Similarity Search

Structure Similarity Search

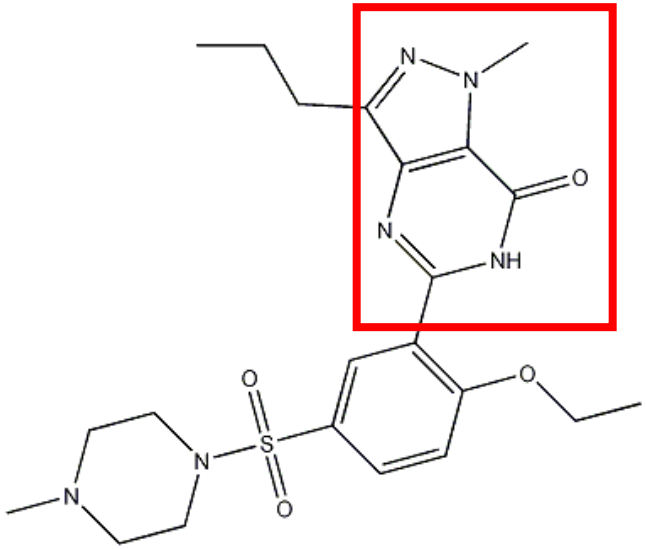
- CHEMICAL COMPOUNDS



(a) caffeine

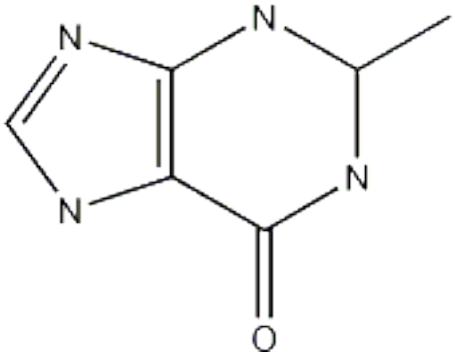


(b) diurobromine



(c) viagra

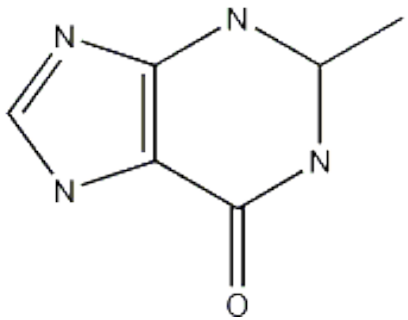
- QUERY GRAPH



Problem
No match or very few matches for a query graph

Some "Straightforward" Methods

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



- **Method 2:** Form a set of subgraph queries from the original query graph and use the exact subgraph search
  - **Exponential**: 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

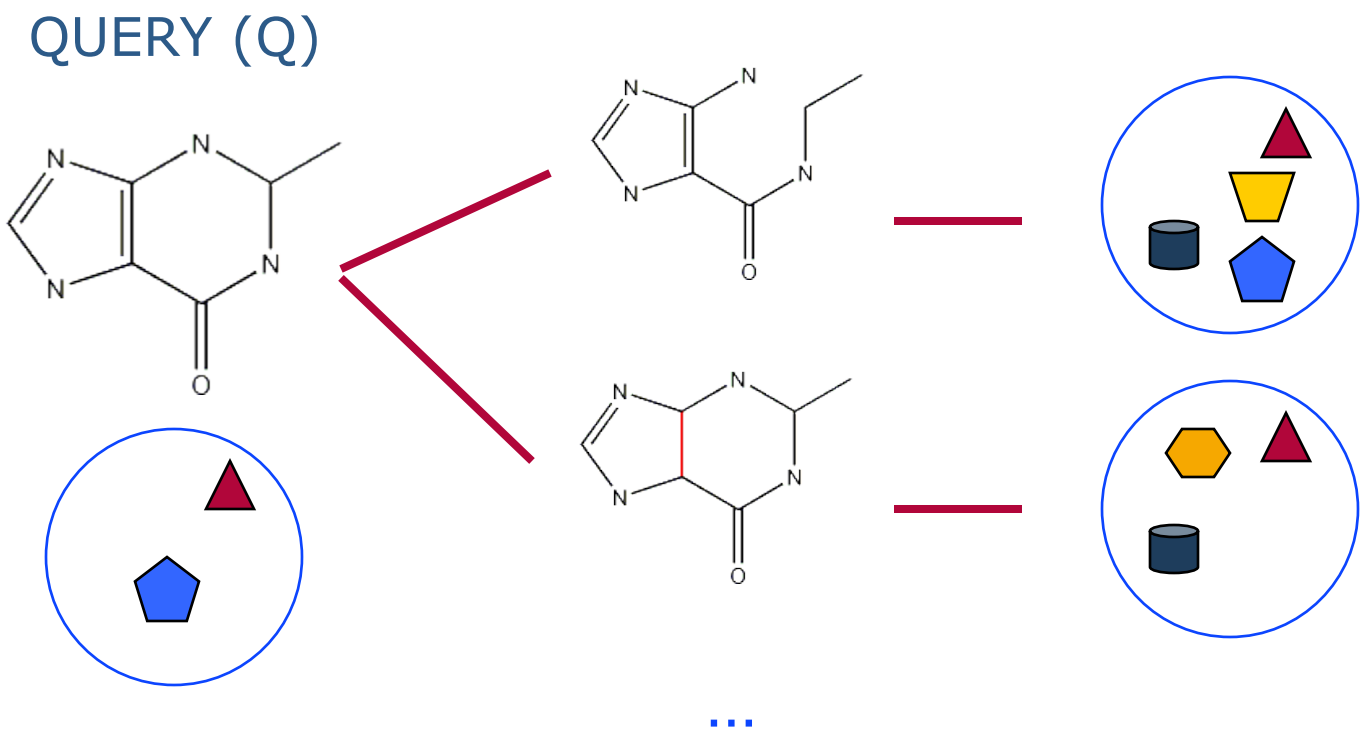
- Precise Search
  - Use frequent patterns as indexing features
  - Select features in the database space based on their selectivity
  - Build the index
- Approximate Search
  - Hard to build indices covering similar subgraphs → explosive number of subgraphs in databases

Idea:

1. Use a feature-based index structure
2. Select features in the query space (so select features such that they miss 1-2.. edges)

Substructure Similarity Measure

- Query relaxation measure:
  - Number of edges that can be relabeled or missed
  - **Problem:** the position of these edges is not fixed

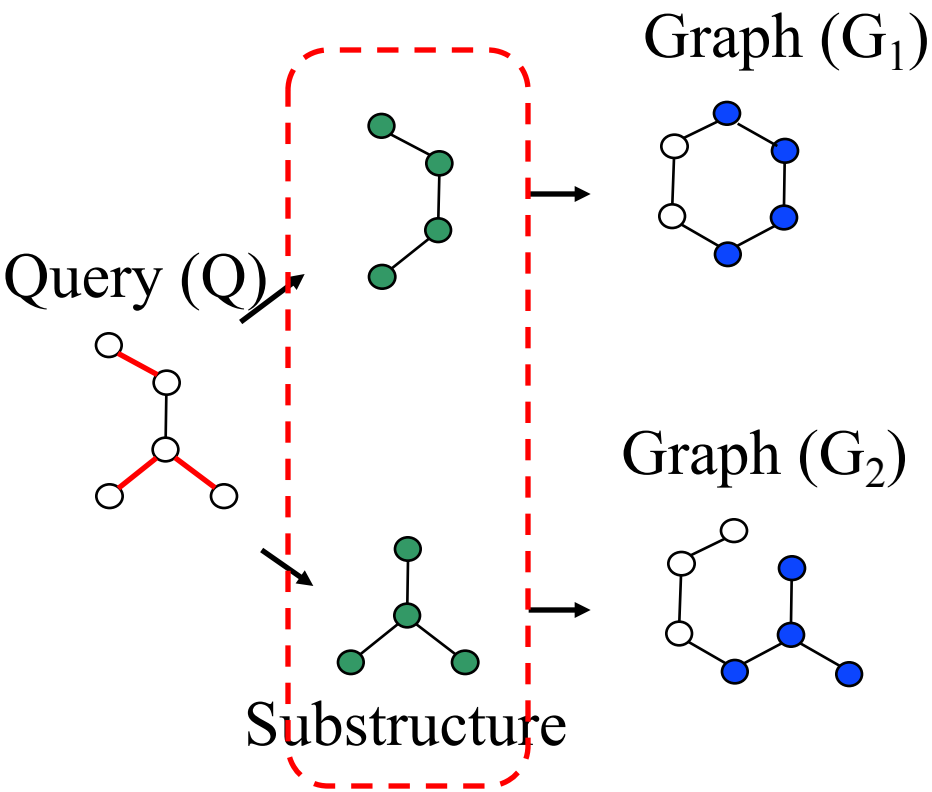


Substructure Similarity Measure

- Feature-based similarity measure
  - Each graph G_i is represented as a feature vector $\mathrm{F_i} = \{f_1, \dots, f_n\}$
  - The similarity is defined by the component-wise distance of the vector with the query vector (using the frequencies of each feature)
  - Advantages
    - Easy to index
    - Fast
    - Rough measure

Shasha, D., Wang, J.T. and Giugno, R. Algorithmics and applications of tree and graph searching. PODS, 2002.

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

Problem
Given a relaxation ratio (i.e., number of missing edges), calculate the maximum number of features that can be missed.

At least one feature 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

Invalid graphs (more than 2 features missing)

Assume a query graph has 5 features and at most 2 features can miss due to some relaxation threshold (computed analytically from the number of missing edges)

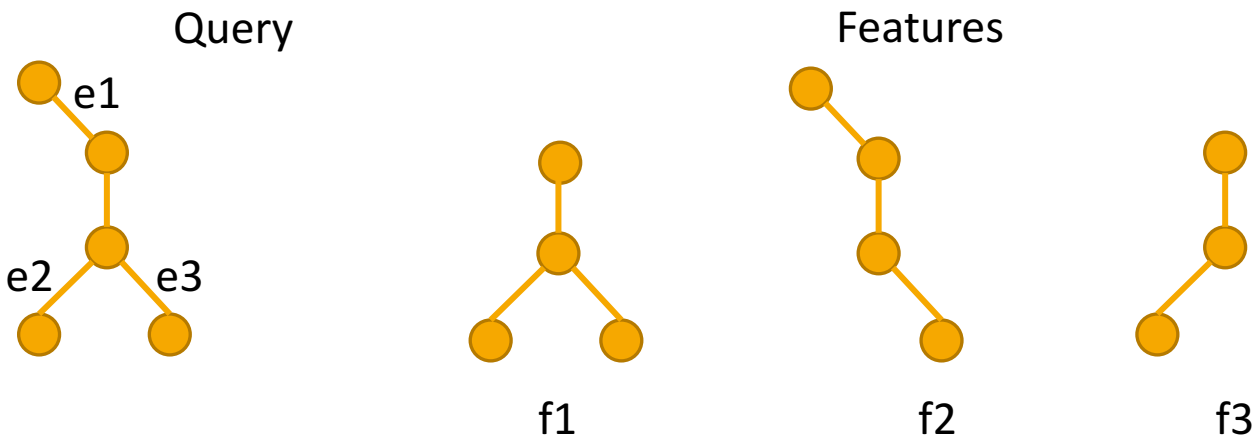
Edge Relaxation – Feature Misses

- If we allow k edges to be relaxed, find J such that J is the maximum number of features to be hit by k edges—it becomes the maximum coverage problem (or set *k*-cover problem)
- **NP**-complete
- A greedy algorithm exists with approximation guarantee

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

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

Edge-feature matrix



The user selected e1,e2,e3 as relaxable edges

Embeddings (matches of the pattern in the query

Edges

	f1	f2(1)	f2(2)	f3(1)	f3(2)	f3(3)	f3(4)
e1	0	1	1	1	0	0	0
e2	1	1	0	0	1	0	1
e3	1	0	1	0	0	1	1

Finding the maximum number of features that can be missed

- Using the edge-feature matrix, consider all the features that are hit at least k times, where k is the maximum number of relaxable edges
- Use a greedy strategy to approximate the value:
  1. Select the row that hits the largest number of columns
  2. Removes the row and the columns that have a 1 in this row
  3. Repeat 1-2 until k rows have been removed.
  4. Return the number of columns removed.
- Improve the solution with searching the most selective features (those that include more edges)

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

In the next episode …

Node classification

Graph summarization

And much more …

ANY
QUESTIONS
?

References

- Petrakis, E.G.M. and Faloutsos, A., 1997. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering*, *9*(3), pp.435-447.
- Shokoufandeh, A., Dickinson, S.J., Siddiqi, K. and Zucker, S.W., 1999. Indexing using a spectral encoding of topological structure. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* (Vol. 2). IEEE.
- Berretti, S., Del Bimbo, A. and Vicario, E., 2001. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(10), pp.1089-1105.
- Holder, L.B., Cook, D.J. and Djoko, S., 1994, July. Substructure Discovery in the SUBDUE System. In *KDD workshop* (pp. 169-180).
- Yan, X., Yu, P.S. and Han, J., 2004, June. Graph indexing: a frequent structure-based approach. SIGMO.
- Shasha, D., Wang, J.T. and Giugno, R. Algorithmics and applications of tree and graph searching. PODS, 2002.
- James, C.A., Weininger, D. and Delany, J., 1995. Daylight theory manual. *Daylight chemical information systems*, *3951*.
- Yuan, D. and Mitra, P. Lindex: a lattice-based index for graph databases. *VLDBJ, 2013*.
- Zhao, P., Yu, J.X. and Yu, P.S. Graph indexing: tree+ delta<= graph. PVLDB, 2007.