Graph Indexing: Tree + Delta>=Graph

Peixiang Zhao The Chinese University of Hong Kong
Jeffrey Xu Yu The Chinese University of Hong Kong
Philip S.Yu IBM T.J Watson Research Center

VLDB 2007

Part I Preliminaries

- **I. Preliminaries**
- II. Graph vs Tree vs Path
- III. Implementations
- 3.1 Tree Feature
- 3.2 Graph Feature (Including Query Processing)
- IV. Experimental Study
- V. Discussion

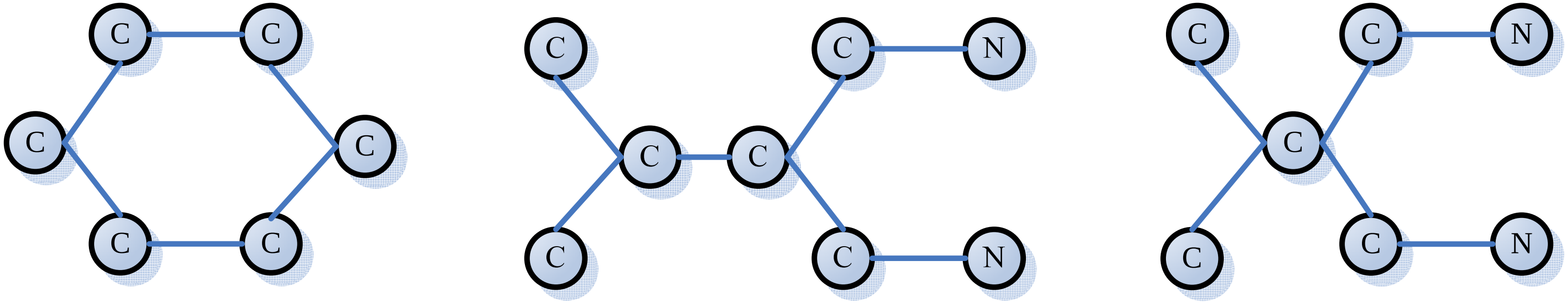
Problem Definition

- Graph Containment Query Problem

  Given a graph database, $G = \{g_1, g_2, ..., g_n\}$,and a query graph *q*, a graph containment query problem is to find the graphs from *G* in which *q* is a subgraph.

  (find all *q*'s supergraph in *G*)

Problem Definition

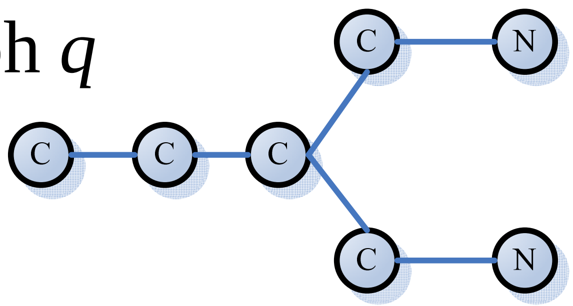


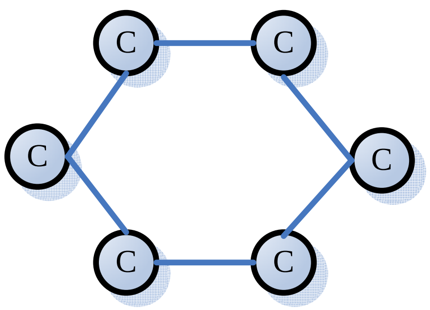
a *b* *c*

- A Graph Database with Three Graphs

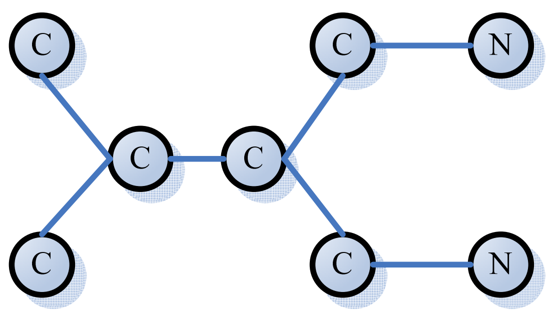
Problem Definition

- A query graph *q*

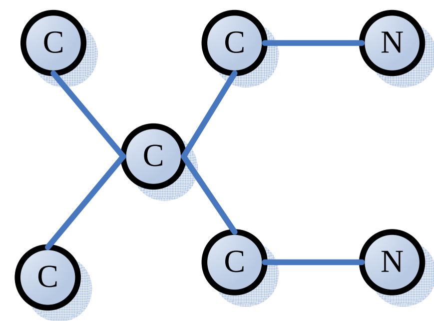




a



b

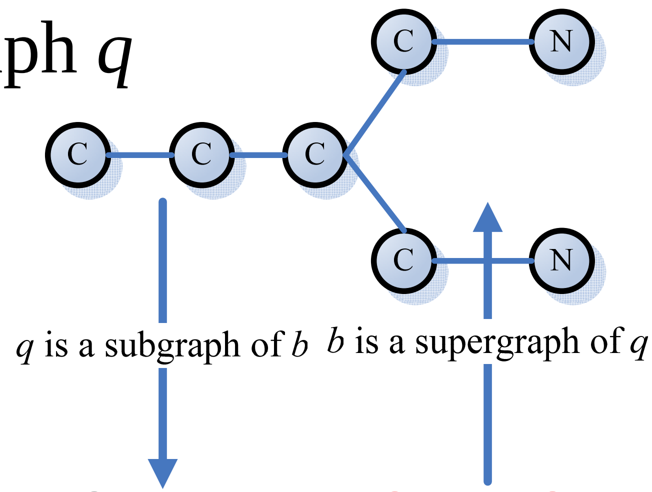


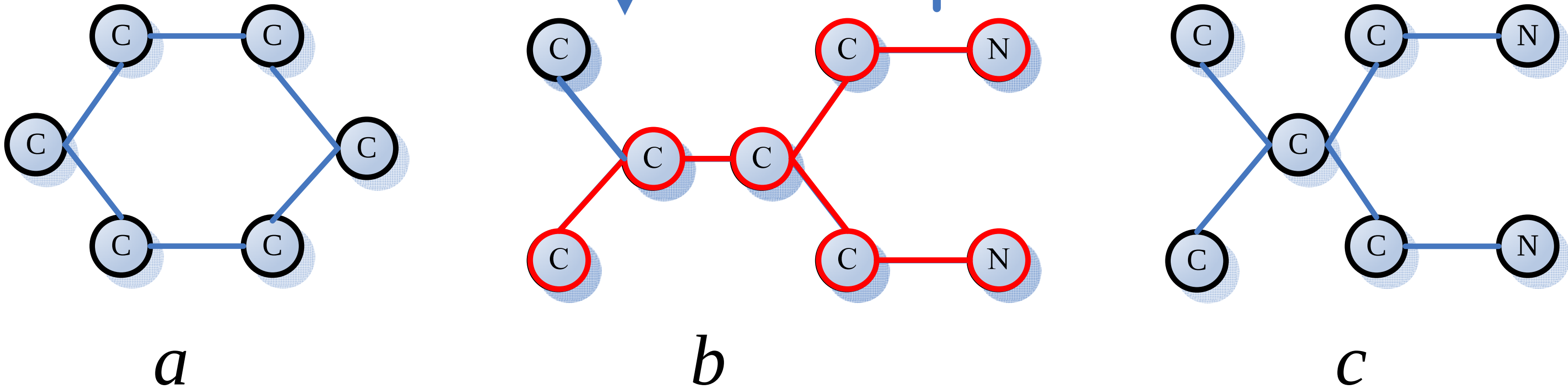
c

- A Graph Database with Three Graphs

Problem Definition

- A query graph *q*





- A Graph Database with Three Graphs

Feature-based Index

- **Index construction** generates a set of features, F, from the graph database G. Each feature, f, maintains a set of graph ids in G containing, f, sup(f).
- **Query processing** is a filtering-verification process.

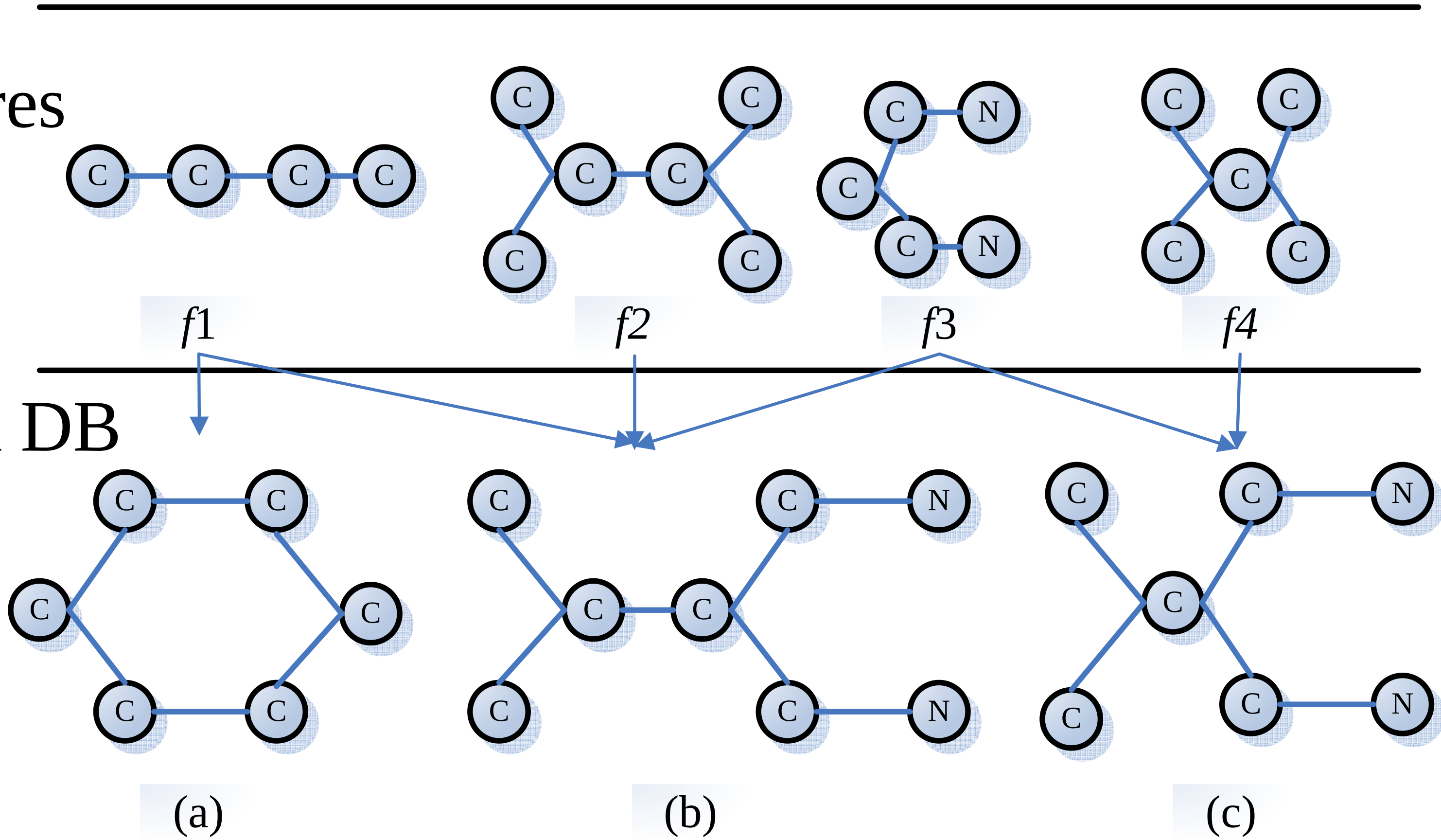
  Filtering phase use the features in query graph q, to compute the candidate set.

  $$C_q = \bigcap_{f \subseteq q \wedge f \in F} \sup(f)$$

  Verification phase checks subgraph isomorphism for every graph in Cq. False positives are pruned.

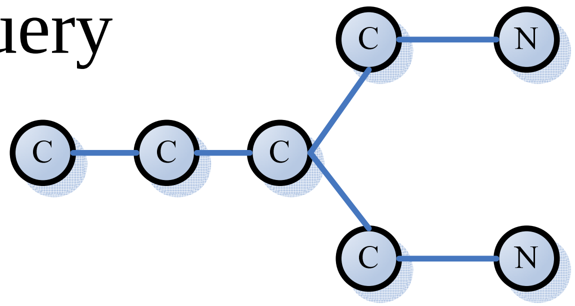
Feature-based Index

- Features
- Graph DB

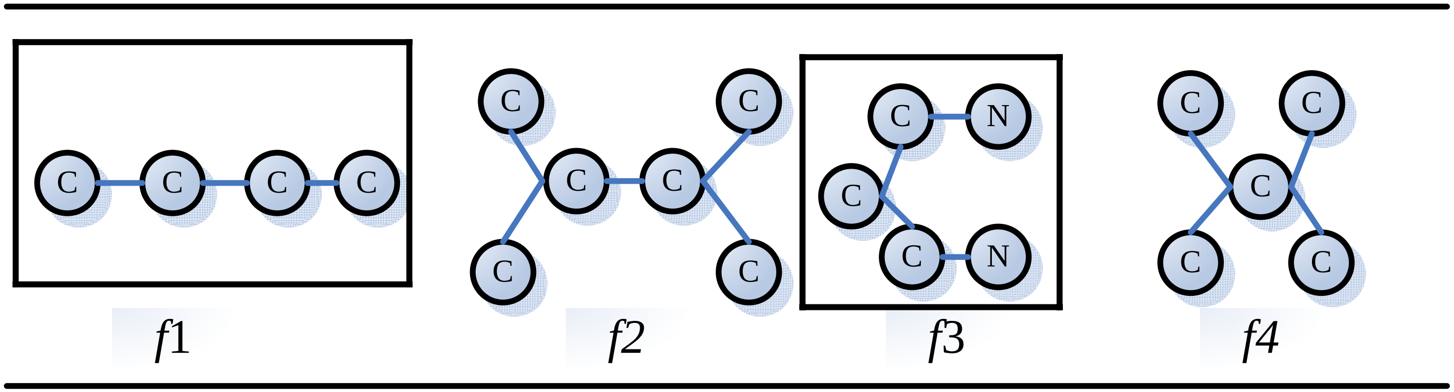


Feature-based Index

- Query

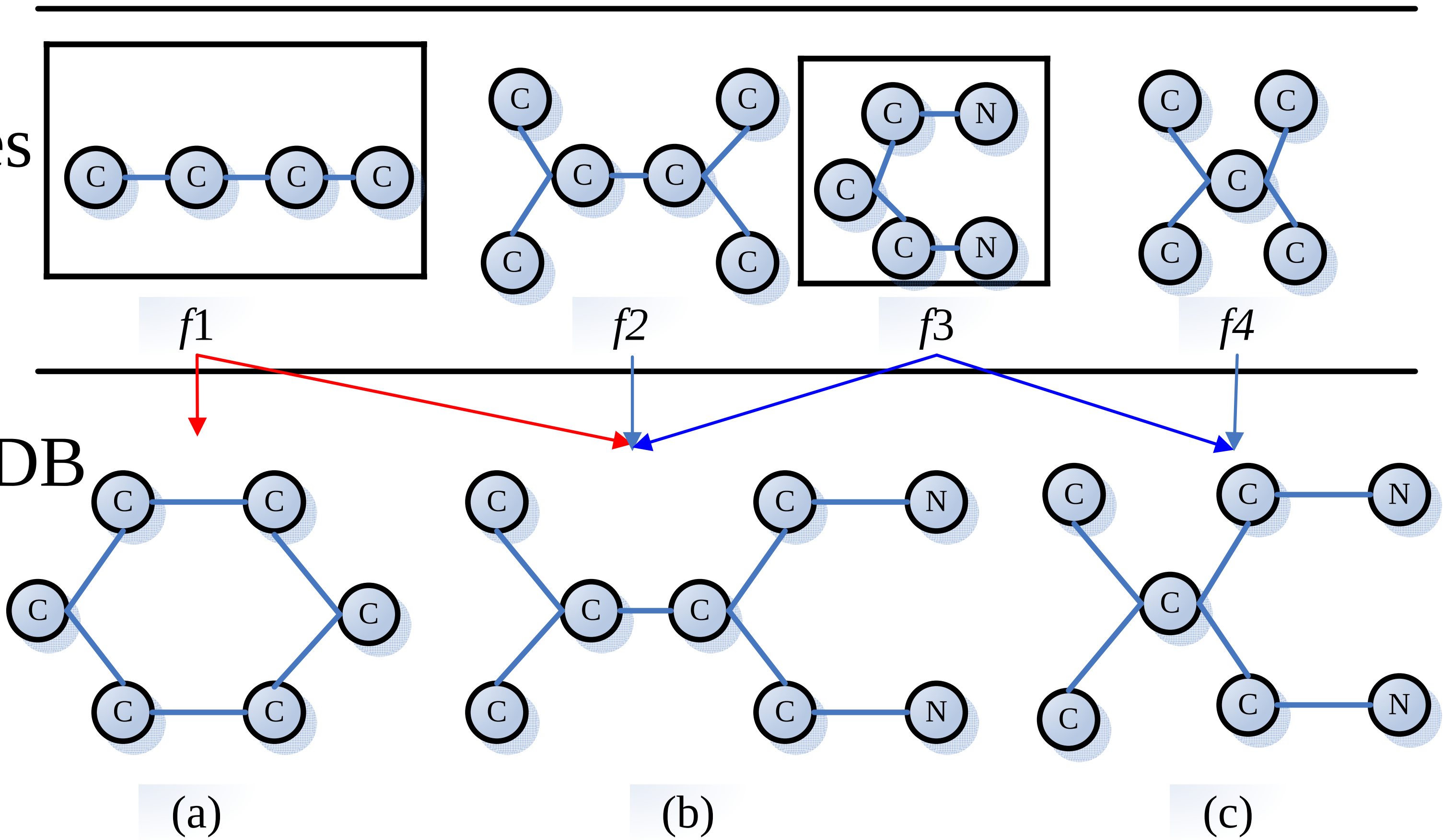


- Features



Feature-based Indexing

- Features
- Graph DB



Query Cost Model

- The cost of processing a graph containment query q upon G is modeled as: $C = C_f + |C_q| \times C_v$
  - *Cf*: the filtering cost
  - *Cv:* the verification cost (NP-Complete)
- Several Fact:
  - To improve query performance is to minimize |*Cq*|
  - The feature set F selected has great impacts on *Cf* and |*Cq*|
  - There is also an **index construction cost**, which is the cost of mining the feature set F

Brief Review

- 1.Feature-based Indexing
- GraphGrep (PODS'02 D.shasha, J.T.L wang, and R. Giugno)
  - An efficient index construction process
  - Limited pruning power.
- GIndex (SIGMOD'04 X. Yan, P.S. Yu and J. Han)
  - A costly index construction process
  - Great pruning power
- 2.Cluster-based Indexing
- C-Tree (ICDE'06 H.He and A.K. Singh)

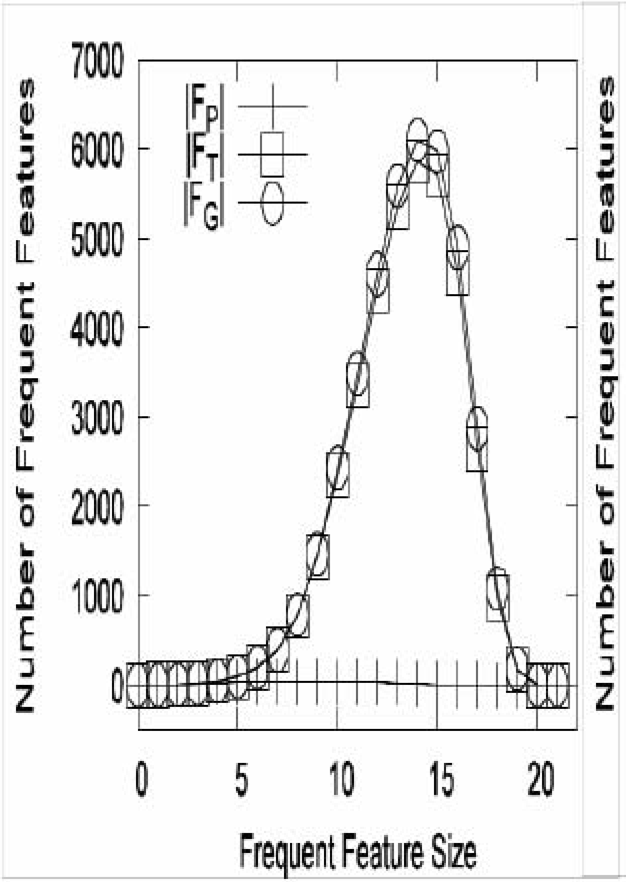
Part II Graph vs Tree vs Path

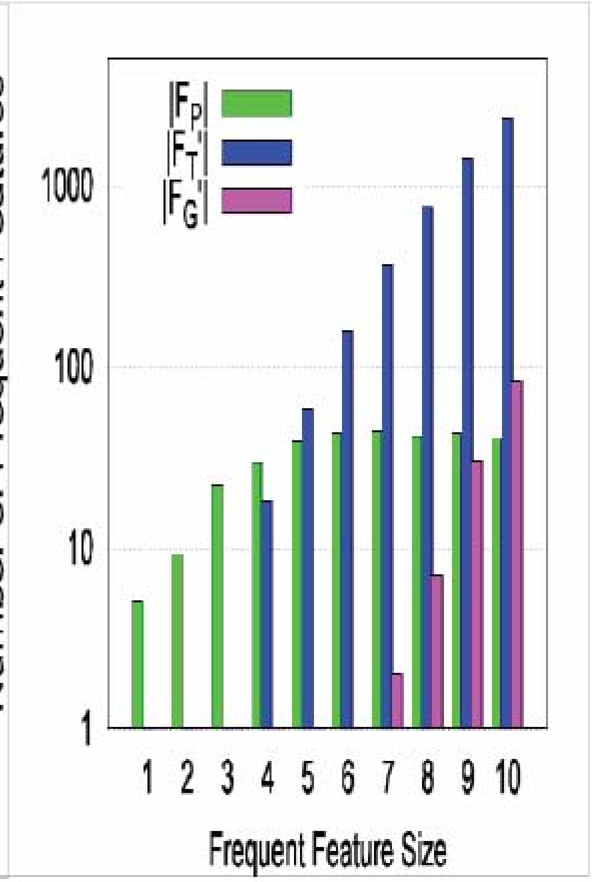
- I. Preliminaries
- **II. Graph vs Tree vs Path**
- III. Implementations
- 3.1 Tree Feature
- 3.2 Graph Feature (Including Query Processing)
- IV. Experimental Study
- V. Discussion

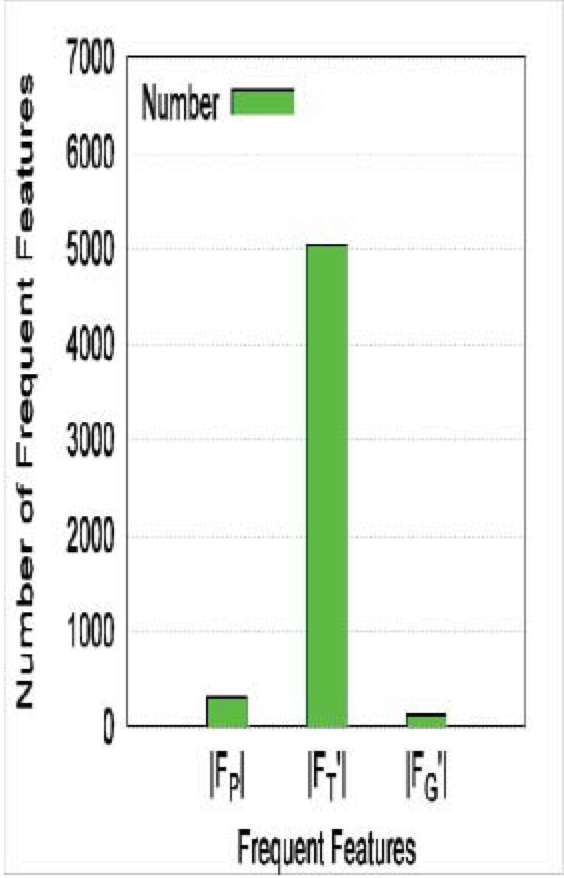
Tree Features?

- Regarding paths and graphs as index features:
  - The cost of generating **path** features is small but candidate set can be large
  - The cost of generating **graph** feature is high but the candidate set can be small
- The key observation: the majority of frequent graph-features (more than 95%) are trees
- How good can tree features do?

Frequent Feature Distributions







- The Real Dataset (AIDS antivirus screen dataset) N=1000, $\sigma = 0.1$

The Feature Selection Cost: CFS

- Given a graph database, *G*, and a minimum support threshold, σ, to discover the frequent feature set *F* from *G*.
- ***Graph***: two prohibitive operations are unavoidable
  - – Subgraph isomorphism
  - – Graph isomorphism
- ***Tree***: one prohibitive operation is unavoidable
  - – Tree-in-Graph testing
- ***Path***: polynomial time

The Candidate Set Size: $|Cq|$

- Let pruning power of a frequent feature, *f*, be

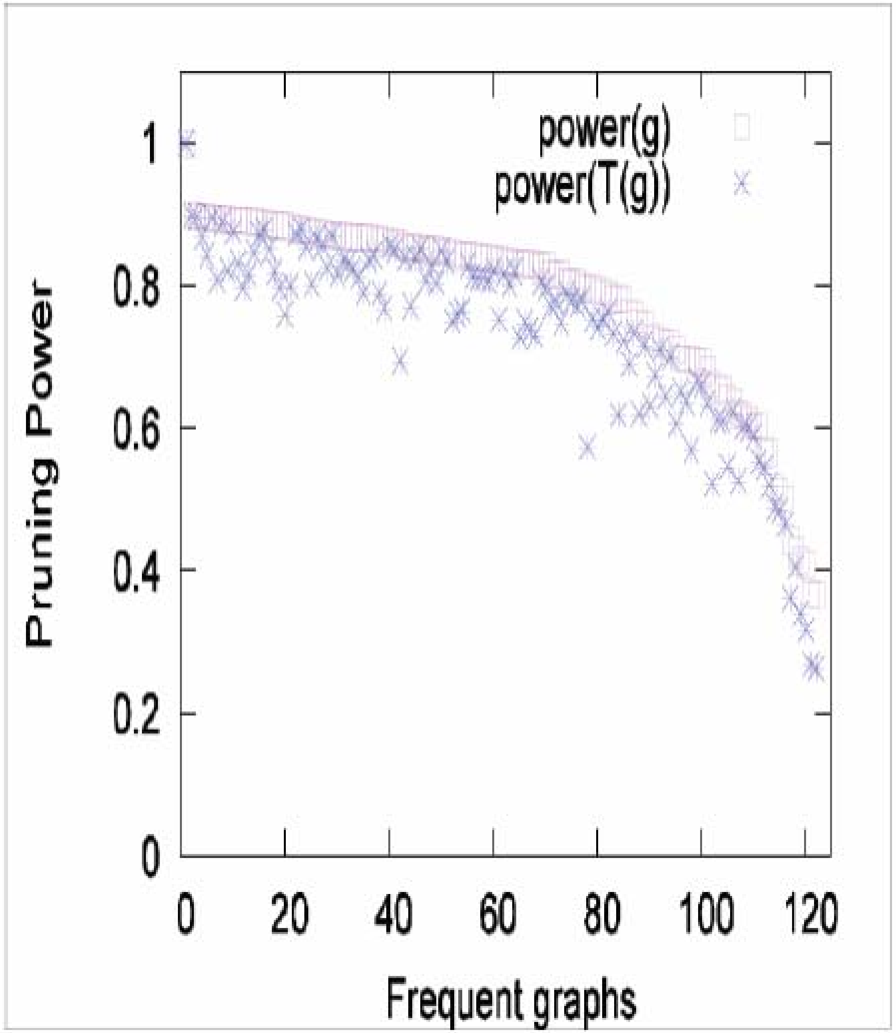
$$\text{power}(f) = \frac{|G| - |\sup(f)|}{|G|}$$

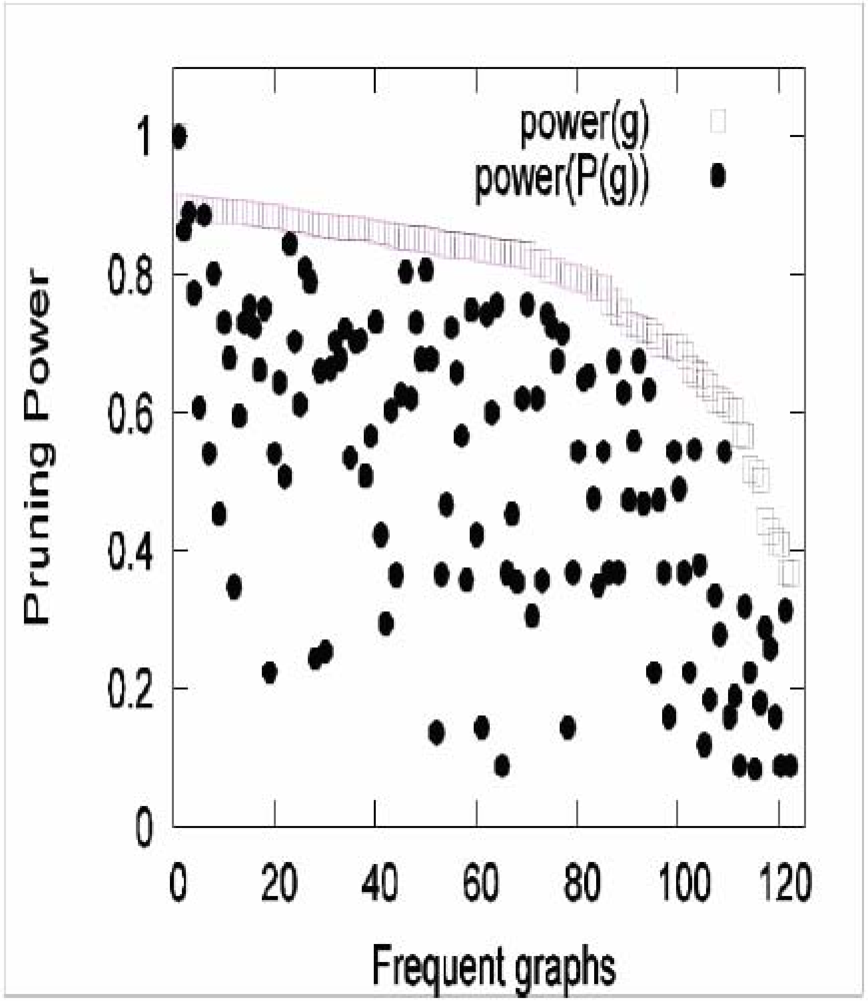
- Let pruning power of a frequent feature set $S = \{f_1, f_2 \ldots . f_n\}$

$$\text{power}(S) = \frac{|G| - |\bigcap_{i=1}^{n} \sup(f)|}{|G|}$$

- • Let a frequent subtree feature set of graph, g, be
- $T(g) = \{t_1, t_2, \ldots, t_n\}$. **power(*g*) ⩾ power(*T* (*g*))**
- • Let a frequent subpath feature set of tree, t, be
- $P(t) = \{p_1, p_2, \ldots, p_n\}$. **power(*t*) ⩾ power(*P* (*t*))**

The Pruning Power





Indexability of Tree

- The frequent tree-feature set dominates (95%).
- Discovering frequent tree-features can be done much more efficiently than mining frequent general graph-features.
- Frequent tree features can contribute similar pruning power as frequent graph features do.

Part III Implementations

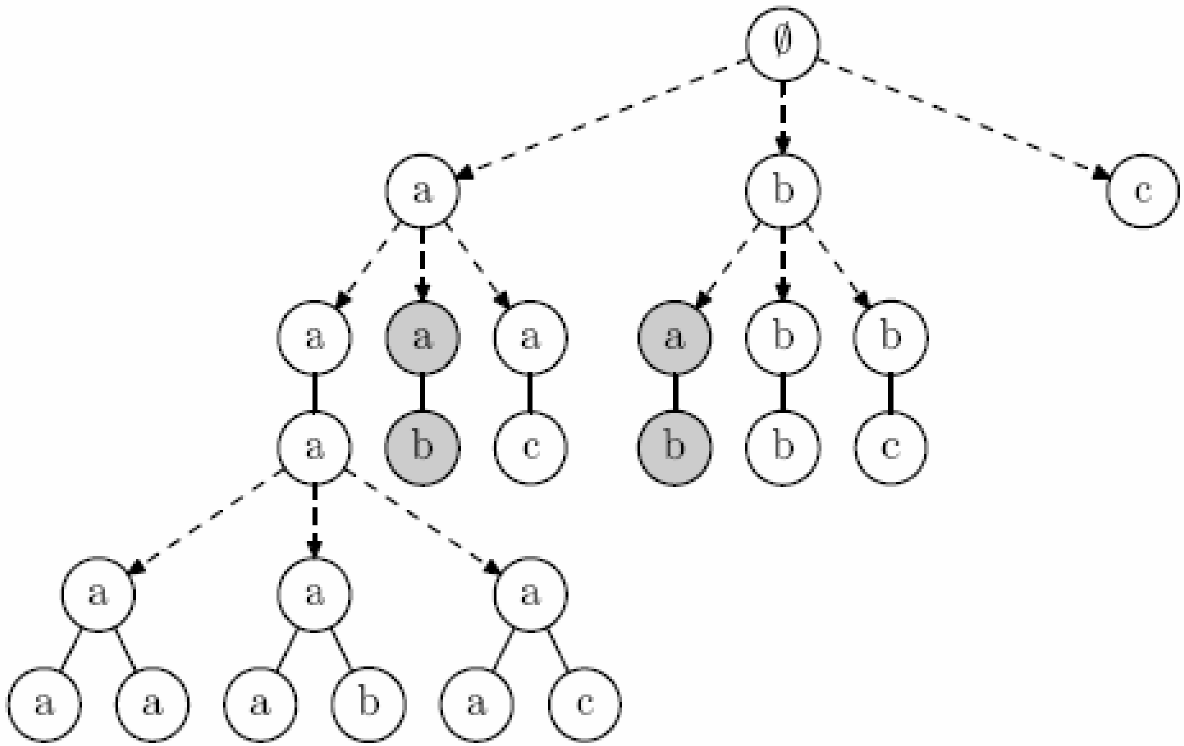
- I. Preliminaries
- II. Graph vs Tree vs Path
- III. Implementations
- **3.1 Tree Feature**
- 3.2 Graph Feature (Including Query Processing)
- IV. Experimental Study
- V. Discussion

Mining Graph Database

- Generate trees by the enumeration tree.
- Use equivalence classes to prune enumeration space
  - Fast Frequent Free Tree Mining in Graph Databases. Peixiang Zhao, Jeffrey Xu Yu ICDM'06.
- Compute canonical forms for trees effectively
  - Canonical forms for labbelled trees and their applications in frequent subtree mining.

    Yun chi, Yirong yang, Richard R. Muntz. Knoledge and Information System 2005.

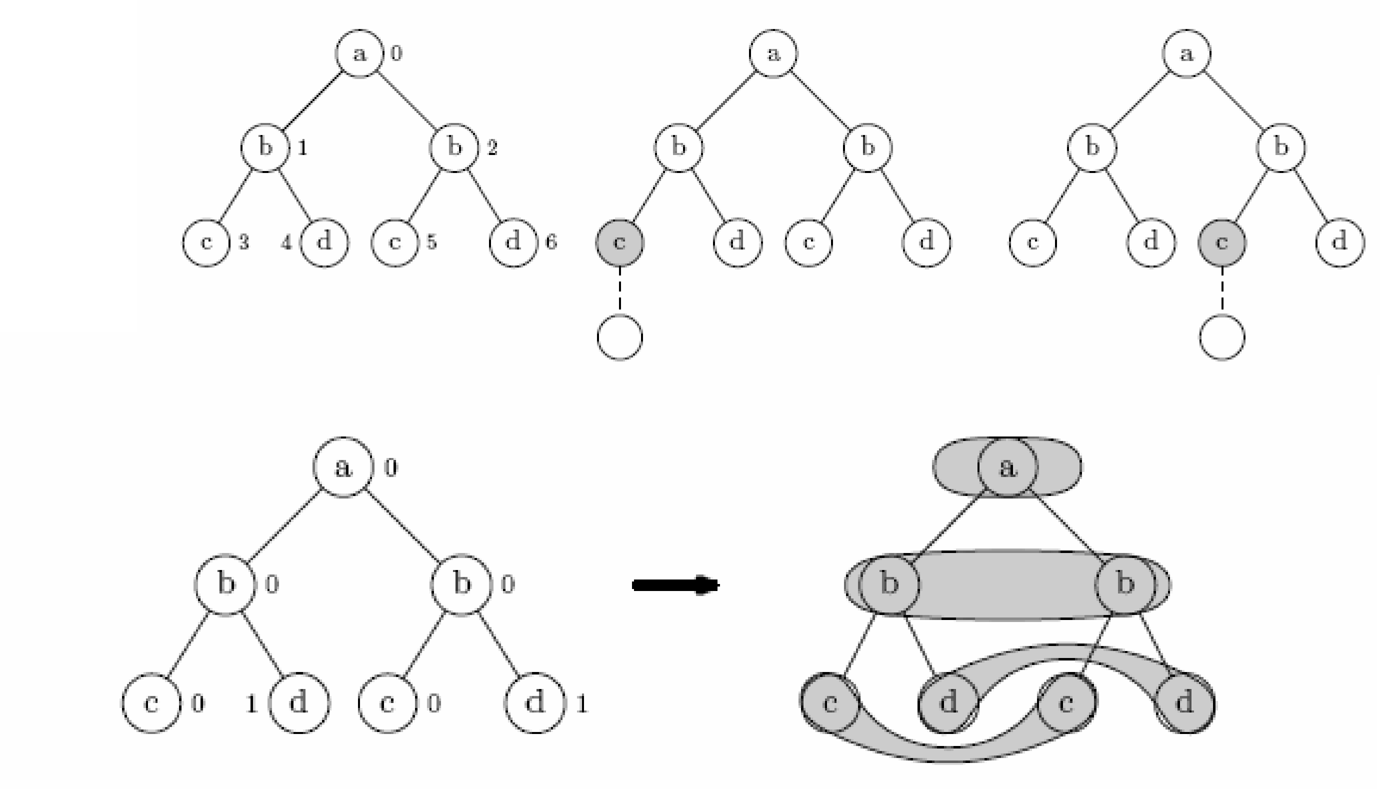
The Enumeration Tree



Compute canonical form for tree

- The cost of computing canonical forms for trees is much lower than the cost for graphs.
- Complexity=$O(c^2 \cdot k \cdot \log k)$
  - c is the maximal degree of the vertices in the tree
  - k is the number of vertices

Equivalence Classes



Mining Cost of Index Construction

- Isomorphism
  - Graph isomorphism: maybe NP-Complete
  - Subgraph isomorphism: NP-Complete
  - Tree-in-Graph testing: maybe NP-Complete
- Canonical Form
  - Graph: maybe NP-Complete
  - Tree: $O(c^2 k \log k)$

Part III Implementations

- I. Preliminaries
- II. Graph vs Tree vs Path
- III. Implementations
- 3.1 Tree Feature
- **3.2 Graph Feature (Including Query Processing)**
- IV. Experimental Study
- V. Dicussion

Add Graph Features On Demand

- 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.
  - If **power(g) >> power($T(g)$),** it needs to select g as an index feature, because g is more ***discriminative*** than $T(g)$, in terms of pruning.
- • Select discriminative graph-features on-demand, **without mining** the whole set of frequent graph-features from G.
- • The selected graph features are additional indexing features, denoted Δ, for later **reuse**.

Discriminative Ratio

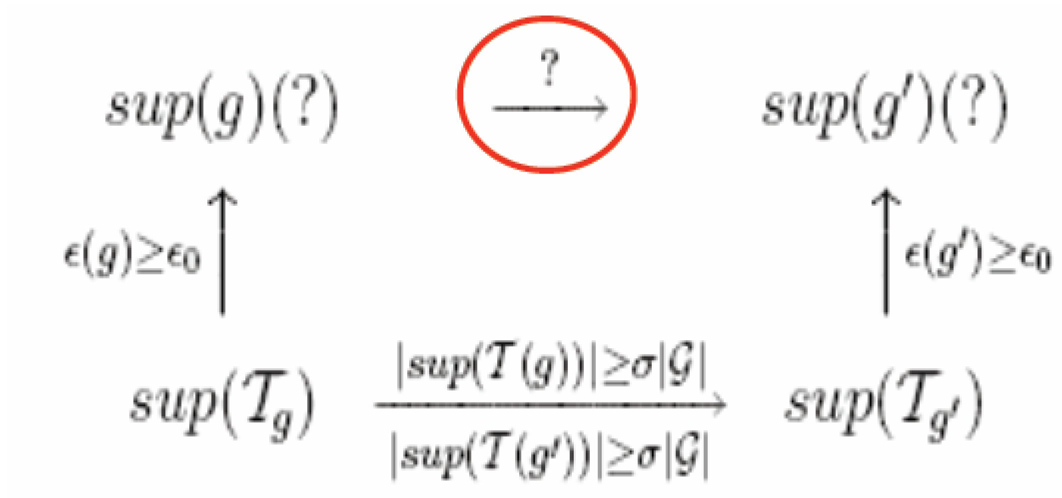
- A discriminative ratio, ε (g), is defined to measure the similarity of pruning power between a **graph-feature** g and its **subtrees** T(g).

$$\varepsilon(g) = \begin{cases} \dfrac{\text{power}(g) - \text{power}(T(g))}{\text{power}(g)} & \text{if power}(g) \neq 0 \\ 0 & \text{if power}(g) = 0 \end{cases}$$

- A non-tree graph feature, g, is discriminative if **ε (g)** $\geqslant$ **ε_0**.

Discriminative Graph Selection (1)

- Consider two graphs ***g*** and ***g'***, where $g \subset g'$
  - • If the gap between power(*g'*) and power(*g*) is large, reclaim *g'* from *G*. Otherwise, do not reclaim *g'* in the presence of *g*.
- Approximate the *discriminative* between *g'* and *g*, in the presence of frequent tree-features discovered.



Discriminative Graph Selection (2)

- Let ***occurrence probability*** of *g* in the graph DB *be*

$$Pr(g) = \frac{|sup(g)|}{|\mathcal{G}|} = \sigma_g$$

- The ***conditional occurrence probability*** of *g'*, *w.r.t. g*:

$$Pr(g'|g) = \frac{Pr(g \wedge g')}{Pr(g)} = \frac{Pr(g')}{Pr(g)} = \frac{|sup(g')|}{|sup(g)|}$$

- ***When Pr(g'|g) is small, g' has higher probability to be discriminative w.r.t. g.***

Discriminative Graph Selection (3)

- The upper and lower bound of $Pr(g'|g)$ become

$$Pr(g'|g) = \frac{|sup(g')|}{|sup(g)|} \leq \frac{|\mathcal{G}| - \frac{|\mathcal{G}|-|sup(T(g'))|}{1-\epsilon_0}}{\sigma|\mathcal{G}|} = \frac{\sigma_{T(g')} - \epsilon_0}{(1-\epsilon_0)\sigma}$$

$$Pr(g'|g) = \frac{|sup(g')|}{|sup(g)|} \geq \frac{\sigma|\mathcal{G}|}{|\mathcal{G}| - \frac{|\mathcal{G}|-|sup(T(g))|}{1-\epsilon_0}} = \frac{\sigma(1-\epsilon_0)}{\sigma_{T(g)} - \epsilon_0}$$

- because ε **(g)** $\geqslant$ ε_0 and ε **(g')** $\geqslant$ ε_0. recall:
  $\sigma x = |\ \text{sup}(x)\ |\ /\ |G\ |$

Discriminative Graph Selection (4)

- Because $0 \leqslant Pr(g'|g) \leqslant 1$, the conditional occurrence probability of $Pr(g'|g)$, is solely upper-bounded by $T(g')$.

$$\sigma_{T(g)} \geq max\{\epsilon_0, \sigma + (1-\sigma)\epsilon_0\}$$

$$max\{\epsilon_0, \sigma\} \leq \sigma_{T(g')} \leq \sigma + (1-\sigma)\epsilon_0$$

$$(\sigma_{T(g)} - \epsilon_0)(\sigma_{T(g')} - \epsilon_0) \geq [\sigma(1-\epsilon_0)]^2$$

Algorithm

Algorithm 4 *Query Processing* $(q, \mathcal{F}_T, \mathcal{G})$

Input: A query graph q, the frequent tree-feature set $\mathcal{F}_T$, and the graph database $\mathcal{G}$

Output: Candidate answer set C_q

$\mathcal{D} \leftarrow \emptyset$;
$\mathcal{T}(q) \leftarrow \{t \mid t \subseteq q, t \in \mathcal{F}_T, size(t) \leq maxL\}$;
$C_q \leftarrow \bigcap_{t \in \mathcal{T}(q)} sup(t)$;
if $(C_q \neq \emptyset)$ **and** (q is cyclic) **then**
  $\mathcal{D} \leftarrow SelectGraph(\mathcal{G}, q)$;
  for all $(g \in \mathcal{D})$ **do**
    $C_q \leftarrow C_q \bigcap sup(g)$;
return C_q;

Algorithm 3 *SelectGraph* $(\mathcal{G}, q)$

Input: A graph database $\mathcal{G}$, a non-tree query graph q

Output: The selected discriminative graph set $\mathcal{D} \subseteq \mathcal{D}(q)$

$\mathcal{D} \leftarrow \emptyset$;
$\mathcal{C} \leftarrow \{c_1, c_2, \cdots, c_n\}, c_i \subseteq q$, c_i is a simple cycle;
for all $c_i \in \mathcal{C}$ **do**
  $g \leftarrow g' \leftarrow c_i$;
  while $size(g') \leq maxL$ **do**
    if $g \notin \Delta$ **then** $\mathcal{D} \leftarrow \mathcal{D} \cup \{g\}$;
    $g' \leftarrow g' \diamond v$;
    if $\mathcal{T}(g), \mathcal{T}(g')$ satisfy Eq. (17), Eq. (18), Eq. (19) **and** $(\sigma_{\mathcal{T}(g')} < \sigma^* \times \sigma_{\mathcal{T}(g)})$ **then**
      $g \leftarrow g'$;
scan $\mathcal{G}$ to compute $sup(g)$ for every $g \in \mathcal{D}$ and add an index entry for g in Δ, if needed;
return $\mathcal{D}$;

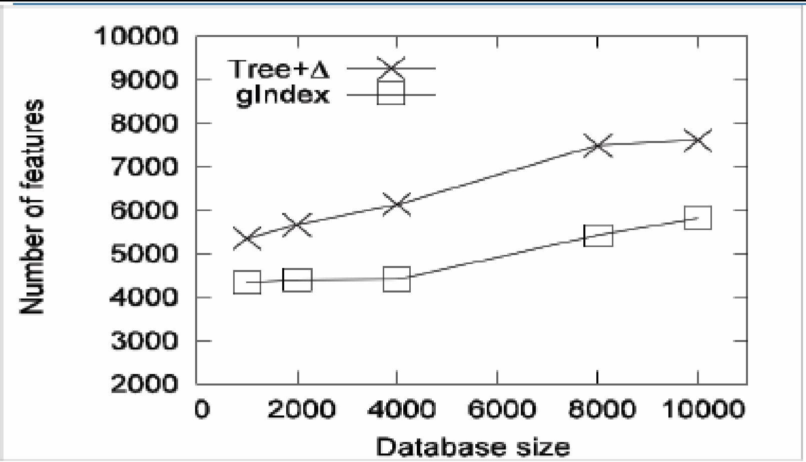
Part IV. Experimental Study

- I. Preliminaries
- II. Graph vs Tree vs Path
- III. Implementations
- 3.1 Tree Feature
- 3.2 Graph Feature (Including Query Processing)
- **IV. Experimental Study**
- V. Discussion

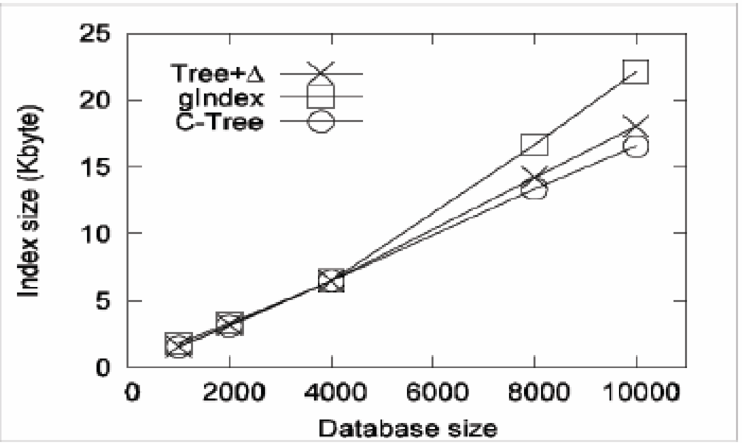
An Experimental Study

- We compared our Tree+ Δ with **gIndex** (X. Yan, P.S. Yu, and J. Han,SIGMOD'04) and **C-Tree** (H. He and A.K. Singh, ICDE'06).
- We used AIDS Antiviral Screen Dataset from the Developmental Theroapeutics Program in NCI/NH (**http://dtp.nci.nih.gov/docs/aids/aids_data.html**)
  - • 42,390 compunds from DTD's Drug Information System.
  - • 63 kinds of atoms (vertex labels).
  - • On average, a compond has 43 vertices and 45 edges.
  - • At max, 221 vertices and 234 edges.
- We also used the graph generator (M. Kuramochi and G. Karypis, ICDM'01).
- We tested on a 3.4GHz Intel PC with 2GB memory.

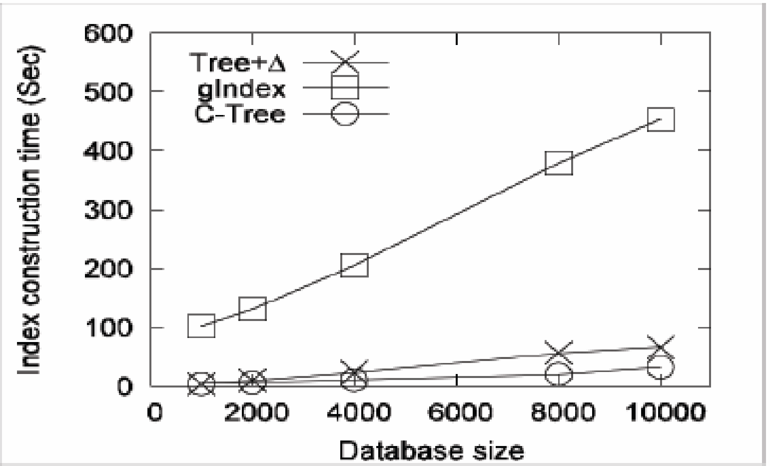
Index Construction (Real Dataset)



Feature Size

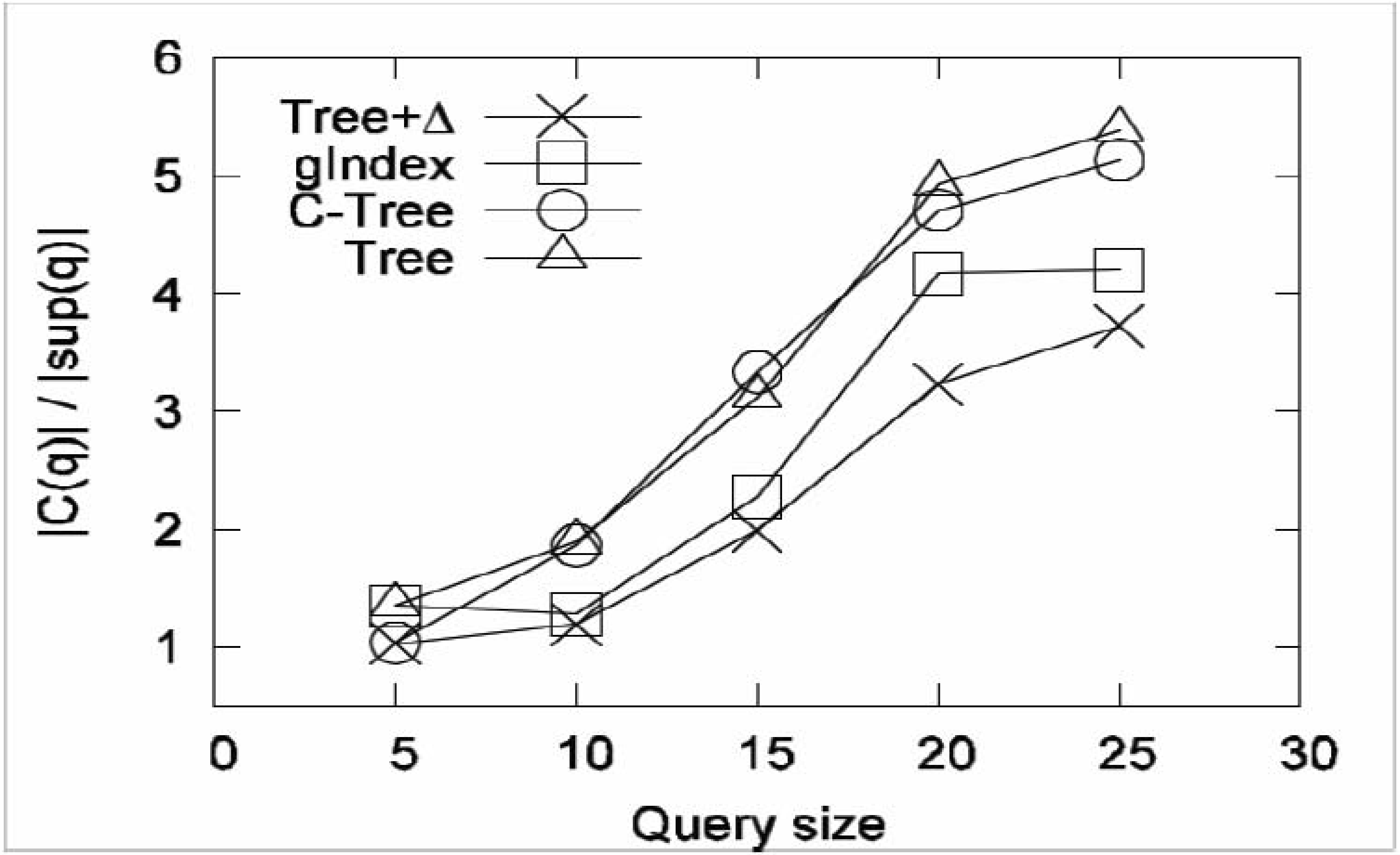


Index Size



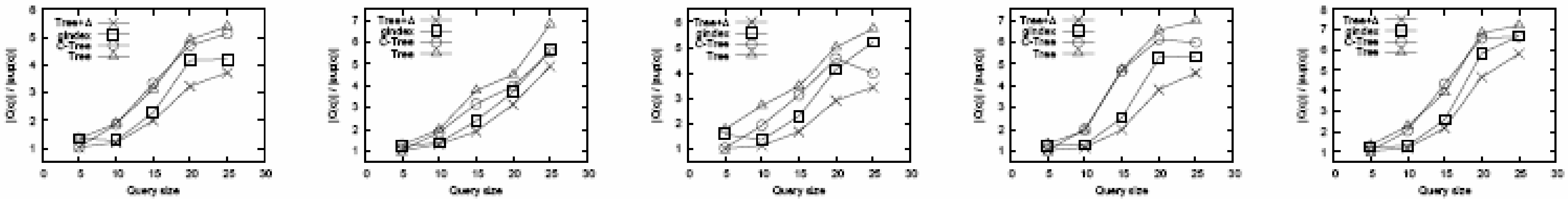
Construction Time

Real Dataset: False Positive Ratio |*Cq*|/|sup(*q*)|)



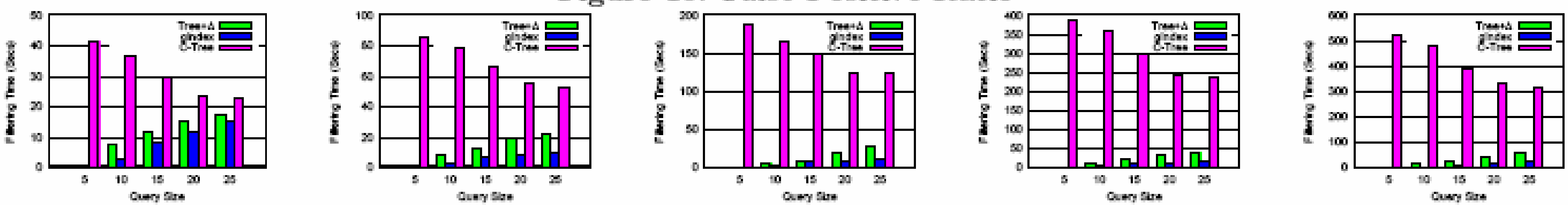
N=1,000

$|Cq|/|sup(q)|$ and Query Time



(a) N=1000 (b) N=2000 (c) N=4000 (d) N=8000 (e) N=10000

Figure 10: False Positive Ratio



(a) N=1000 (b) N=2000 (c) N=4000 (d) N=8000 (e) N=10000

Conclusion

- Tree is an effective and efficient graph indexing feature to answer graph containment queries.
- We analyze the indexibility for tree features.
- We propose a Tree+ Δ approach that holds a compact index structure, achieves better performance in index construction, and provides satisfactory query performance for answering graph containment queries.

Part V. Discussion

- I.   Preliminaries
- II. Graph vs Tree vs Path
- III. Implementations
-   3.1 Tree Feature
-   3.2 Graph Feature (Including Query Processing)
- IV. Experimental Study
- **V.   Discussion**