Graph Indexing: Tree + Delta >= Graph

Peixian Zhao, Jeffrey Xu Yu, Philip S. Yu

The Chinese University of Hong Kong, {pxzhao,yu}@se.cuhk.edu.hk

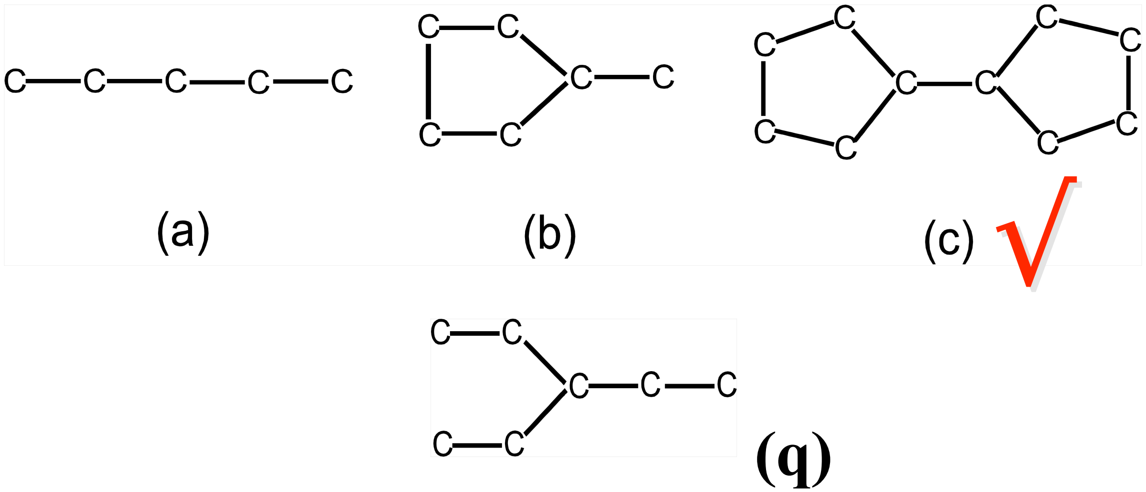
IBM Watson Research Center, psyu@us.ibm.com

An Overview

- Graph containment query
- The framework and query cost model
- Some existing path/graph based solutions
- A new tree-based approach
- Experimental studies
- Conclusion

Graph Containment Query

- Given a graph database G = $\{g_1, g_2, \ldots, g_N\}$ and a query graph q, find the set $sup(q) = \{g_i \mid q \subseteq g_i, g_i \in G\}$



- Infeasible to check subgraph isomorphism for every g_i in G, because subgraph-isomorphism is NP-Complete.

The Framework

- 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 uses the features in query graph, *q,* to compute the *candidate set.*

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

  - **Verification** phase checks subgraph isomorphism for every graph in C_q. False positives are pruned.

Query Cost Model

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

Existing Solutions: Paths vs Graphs

- Path-based Indexing Approach: GraphGrep (*PODS'02*)
  - All paths up to a certain length l_p are enumerated as indexing features
    - – An efficient index construction process
    - – Index size is determined by l_p
    - – Limited pruning power, because the structural information is lost.
- Graph-based Indexing Approach: gIndex (*SIGMOD'04)*
  - Discriminative frequent subgraphs are mined from *G* as indexing features
    - – A costly index construction process
    - – Compact index structure
    - – Great pruning power, because structural information is well-preserved

Tree Features?

- Regarding paths and graphs as index features:
  - The cost of generating **path** features is small but the candidate set can be large.
  - The cost of generating frequent **graph** features 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?**

A New Approach: Tree+Δ

- To explore indexability of path, tree and graph.
- A new approach Tree+Δ :
  - To select frequent tree features.
  - To select a small number of discriminative graph-features that can prune graphs effectively, on demand, ***without costly graph mining***.

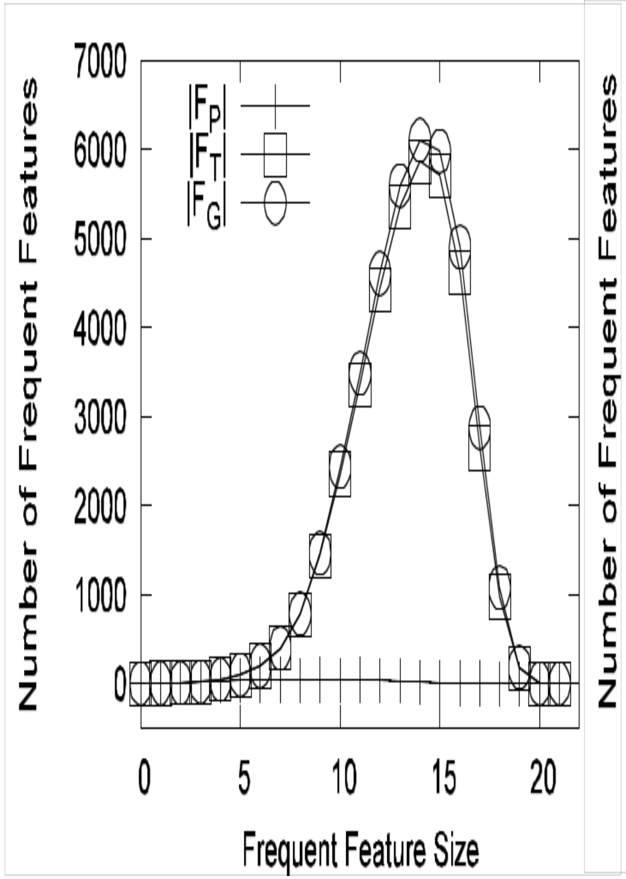
Indexability of Path, Tree and Graph

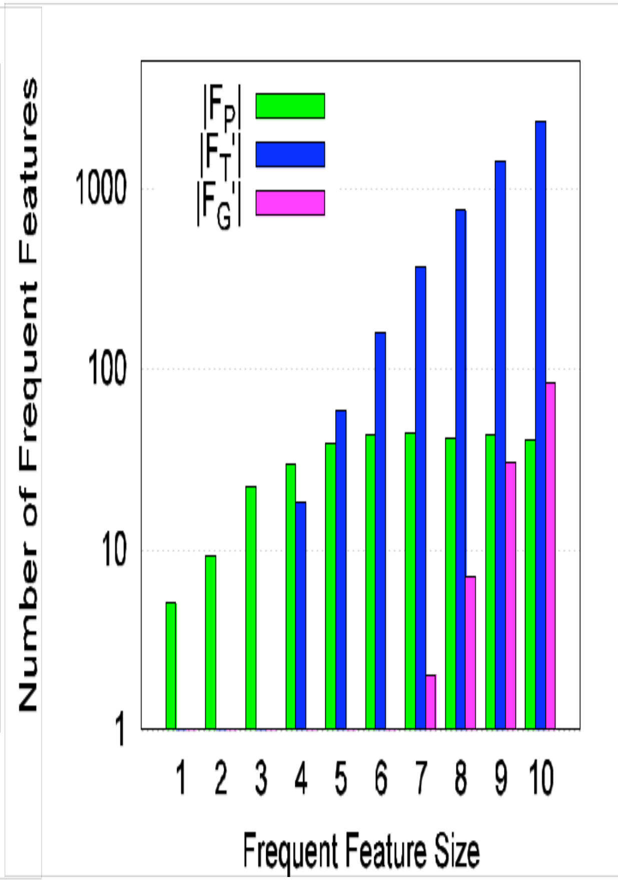
- We consider three main factors to answer indexability.
  - The frequent feature set size: $|\boldsymbol{F}|$
  - The feature selection cost (mining): $\boldsymbol{C_{FS}}$
  - The candidate set size: $|\boldsymbol{C_q}|$

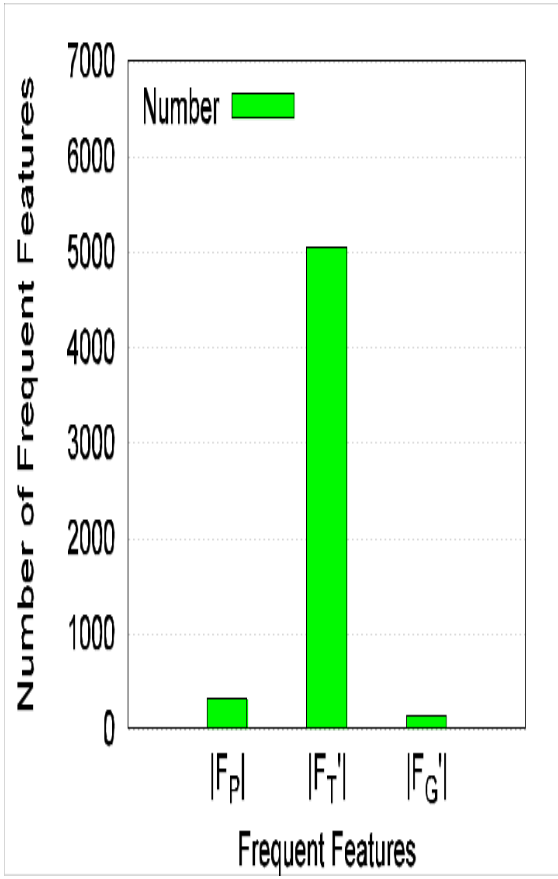
The Frequent Feature Set Size: |*F*|

- 95% of frequent graph features are trees. Why?
- Consider non-tree frequent graph features g and g'.
  - Based on Apriori principle, all g's subtrees, t_1, t_2, …, t_n are frequent.
  - Because of the structural diversity and vertex/edge label variety, there is a little chance that subrees of g coincide with those of g'.

Frequent Feature Distributions







The Real Dataset (AIDS antivirus screen dataset) N = 1,000, σ = 0.1

The Feature Selection Cost: C_{FS}

- 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: $|C_q|$

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

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

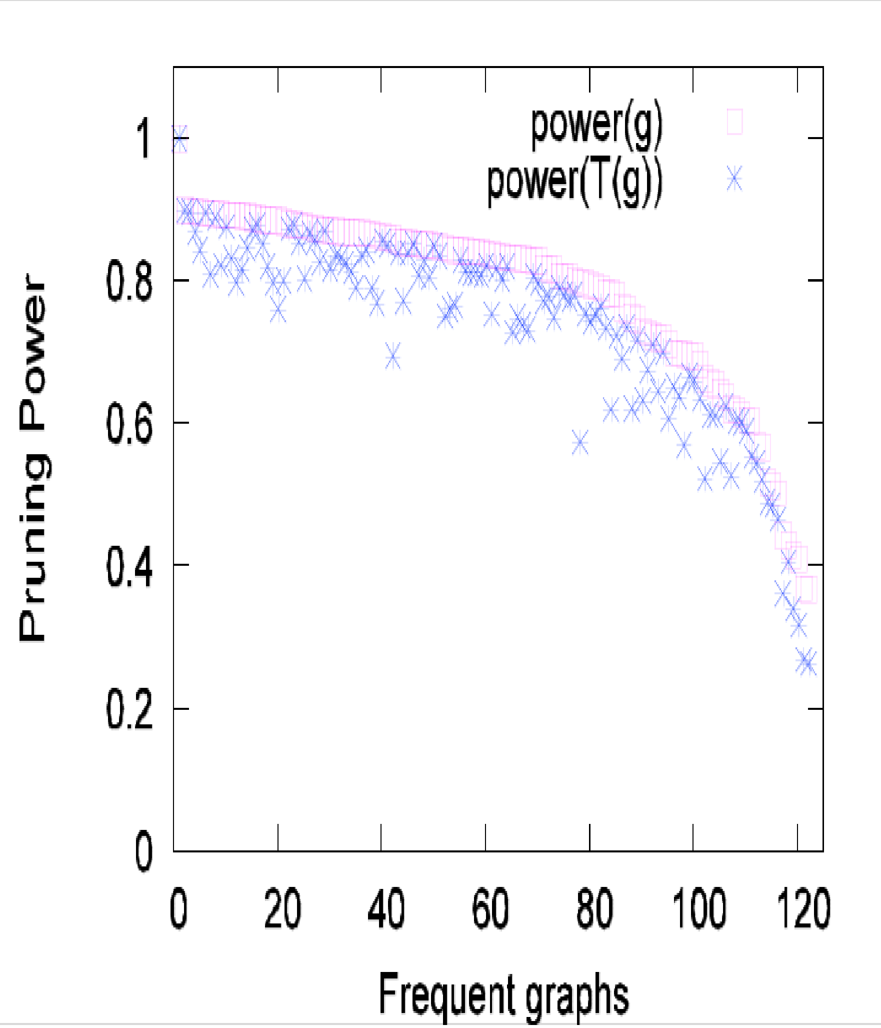
- Let pruning power of a frequent feature set $S = \{f_1, f_2, \ldots, f_n\}$

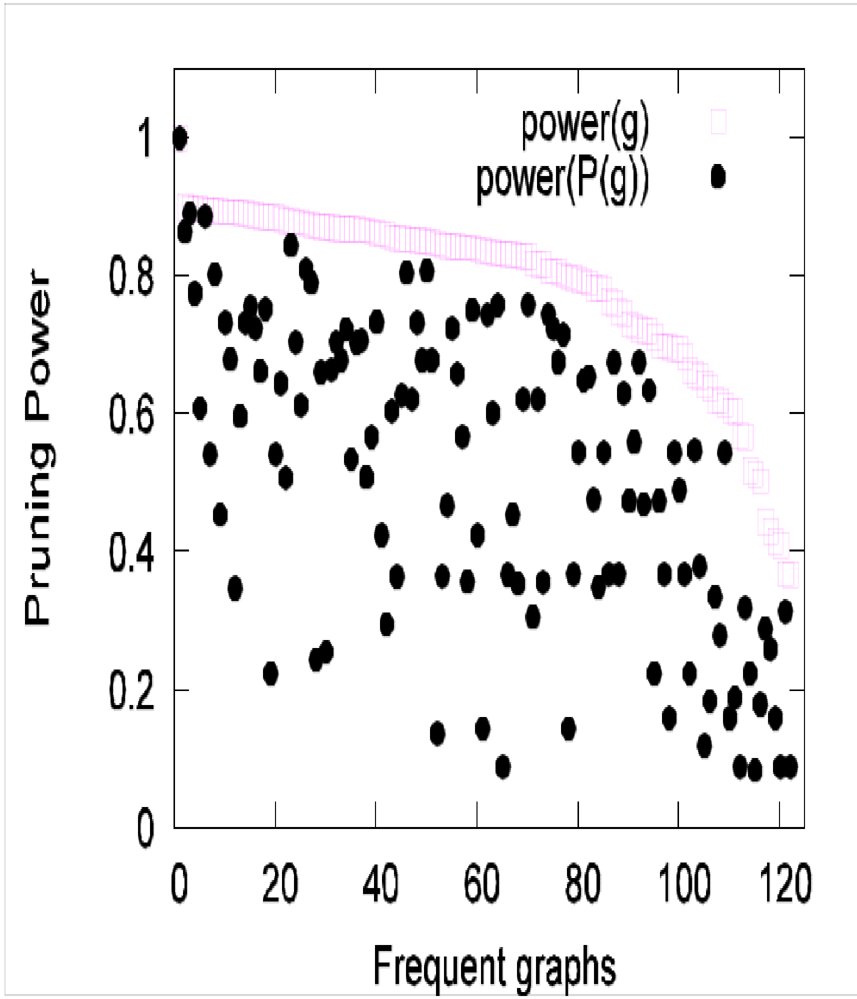
$$\text{power}(S) = \frac{|\mathcal{G}| - |\bigcap_{i=1}^{n} sup(f_i)|}{|\mathcal{G}|}$$

- 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





The Real Dataset (AIDS antivirus screen dataset) N = 1,000, σ = 0.1

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*.

Add Graph Features On Demand

- Consider a query graph *q* which contains a subgraph *g*
  - If **power(*T*(*g*)) ≈ 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, $\varepsilon(g)$, 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 **$\varepsilon(g) \geq \varepsilon_0$**.

Discriminative Graph Selection (1)

- Consider two graphs ***g*** and ***g'***, where ***g*■ *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.

$$\begin{tikzcd}
sup(g)(?) \arrow[rr, "?"] & & sup(g')(?) \\
sup(\mathcal{T}_g) \arrow[u, "\epsilon(g)\geq\epsilon_0"] \arrow[rr, "|sup(\mathcal{T}(g))|\geq\sigma|\mathcal{G}|", "|sup(\mathcal{T}(g'))|\geq\sigma|\mathcal{G}|"'] & & sup(\mathcal{T}_{g'}) \arrow[u, "\epsilon(g')\geq\epsilon_0"']
\end{tikzcd}$$

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

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

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

because **ε(g) ≥ ε_0** and **ε(g') ≥ ε_0**. recall: $\sigma_x = |\sup(x)| / |G|$

Discriminative Graph Selection (4)

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

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

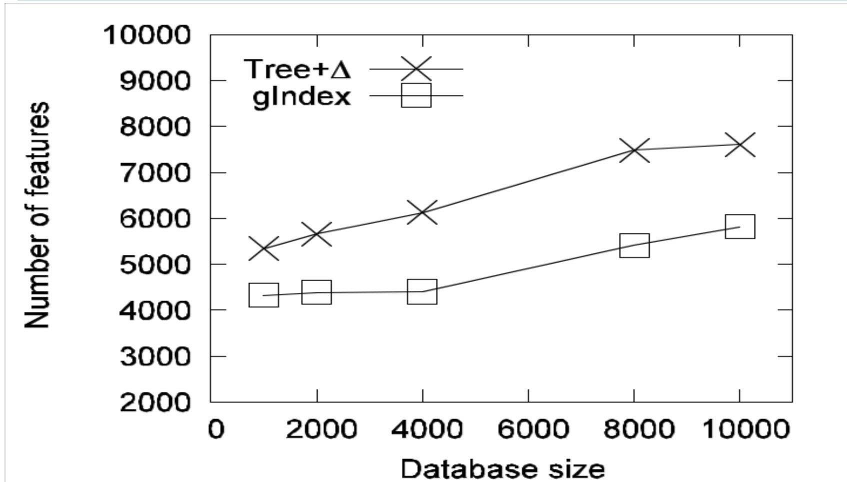
$$max\{\epsilon_0, \sigma\} \leq \sigma_{T(g')} \leq \sigma + (1-\sigma)\epsilon_0$$

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

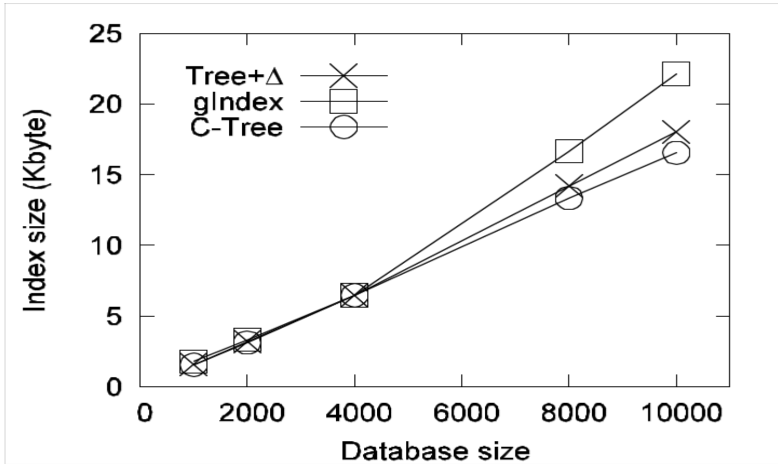
An Experimental Study

- 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.

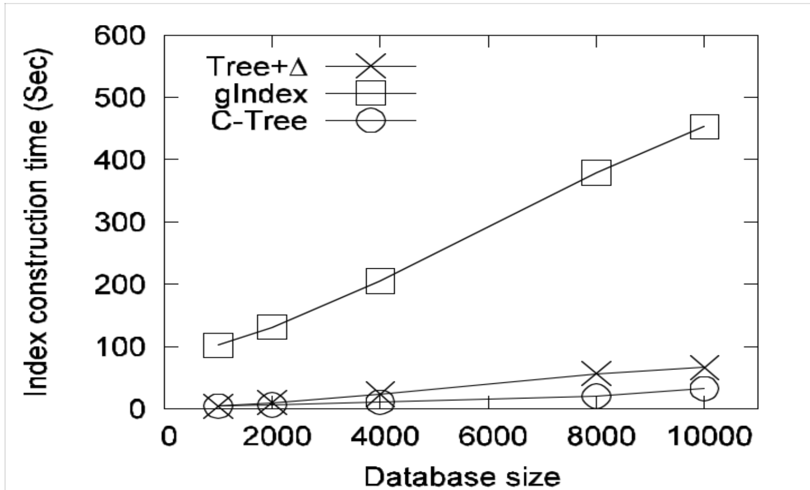
Index Construction (Real Dataset)



Feature Size

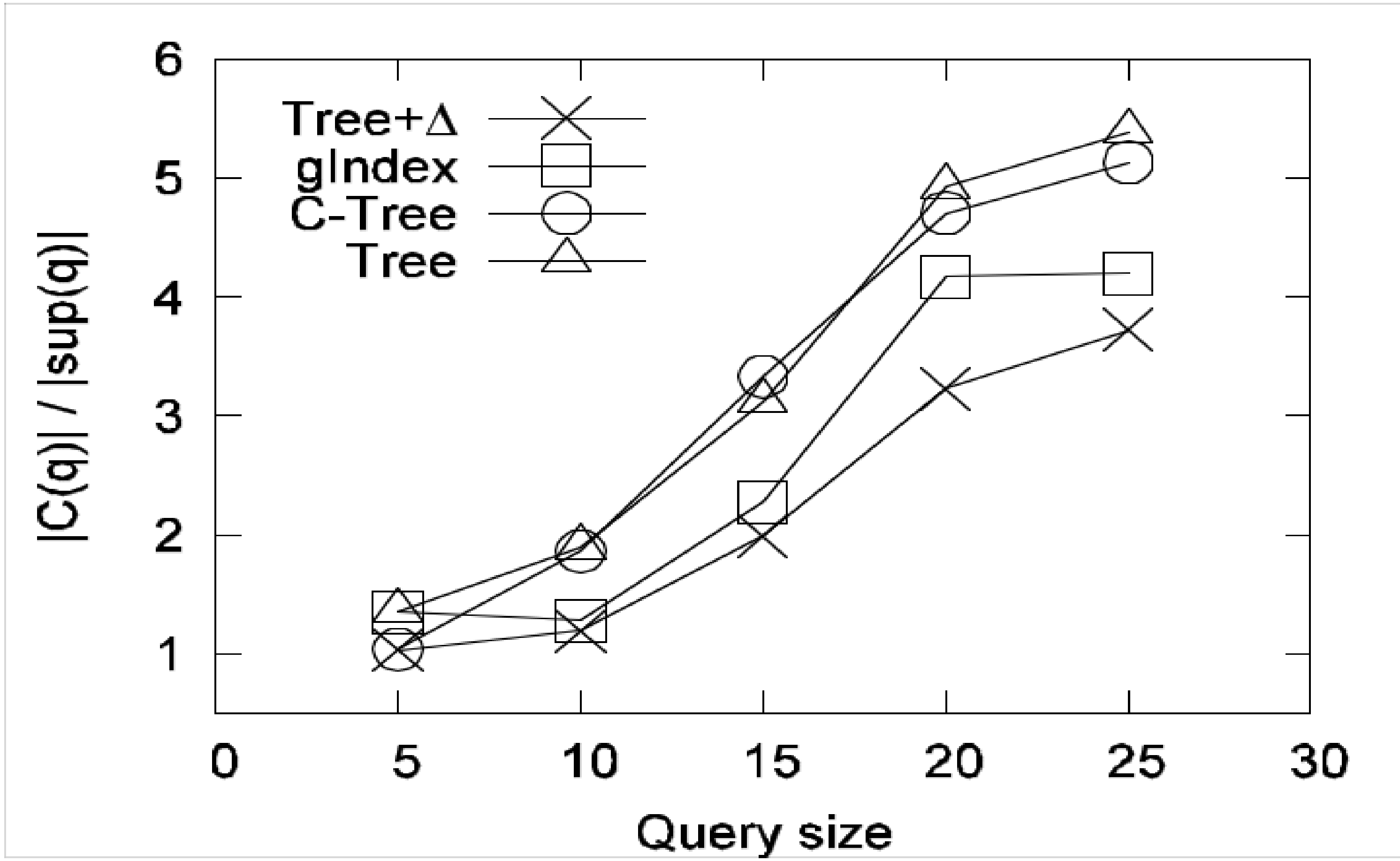


Index Size



Construction Time

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



N=1,000

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.