Summary of TreePi: A Novel Graph Indexing Method

Motivation

Graph databases face a great challenge arising from the emergence of massive complex structural data in biology, chem-informatics, etc. One of the most important research topics is to efficient search and retrieval of graphs using indexing techniques. In the paper, Zhang, etc[1]. used frequent subtrees as indexing structures, because trees can be efficiently manipulated while preserving lots of structural information of the original graph.

Contribution

- Proposed an indexing method using frequent subtrees as feature set.
- Developed a support threshold function to shrink the set of feature trees to save memory
- Introduced the concept of *Center Distance Constraints* to prune the search space.
- Developed a new algorithm that utilized location information to perform subgraph isomorphism tests (The first graph indexing method that uses specific location information).

Methodology

Database preprocessing

Feature Tree Selection

Frequent trees are chosen as index structures for three reasons:

- Trees are more compact forms to preserve the structural information in graph database.
- The isomorphism and normalization on tree are asymptotically simpler than graphs, which is usually NP-complete.
- Many important structures in biology and chemistry applications are trees.

A tree t is σ frequent if $|D_t| \geq \sigma$, where D is the graph database and $|D_t|$ is the size of the subset of D to which t is subgraph isomorphic. The size of feature tree set T_D can grow exponentially, so two methods were developed to make the T_D smaller in size.

First, the frequency threshold σ is not uniformly set for all feature tree set, but rather it is calculated by a supporting function $\sigma(s)$, where s is the edge size of the feature tree set. $\sigma(s)$ is defined as follow:

$$\sigma(s) = \begin{cases} 1 & \text{if } s \leq \alpha \\ 1 + \beta s - \alpha\beta & \text{if } \eta \geq s > \alpha \\ +\infty & \text{if } s > \eta \end{cases} \qquad (1)$$

where α, β, η are positive parameters that can be tuned to specific graph database. The frequency is set to 1 when s is bellow a certain threshold to ensure the completeness of the indexing.

Second, the supporting function $\sigma(s)$ alone is not enough to control the size of the feature tree set, when the graph database is diverse and large. For any feature tree r, we would remove r from the feature tree set when $\frac{|\cap_i D_{r_i}|}{D_r} \leq \gamma$ (γ is called the shrinking parameter)

Heuristics for setting $\alpha, \beta, \eta, \gamma$;

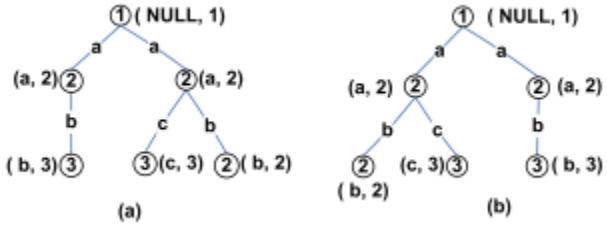
- α: $[\frac{s_q}{4}, \frac{s_q}{2}]$, where S_q is the estimated average size of query graphs.
- β: 1 or 2 when the most frequent substructures is below 10 edges; 5 or 6 when it's above 10 edges.

- η: $\min(s_q, s_D)$, where s_D is the average size of graphs in the database.
- γ: [1, 3]

When the memory size is too small to store all feature trees, we can gradually decrease η and α and increase shrink parameter γ.

Index Construction

The selected feature trees is first converted to its canonical form by sorting the nodes with a predefined sorting order. Every node is labeled by the edge label connecting to parent node and its node label. We first sort by the edge label and then the node label. If they are the same, then we sort by the subtree. An example is given as follow:



After transforming a feature tree to its canonical form, we can use BFS to construct a unique string to represent the feature tree. Then, the paper uses prefix tree based indexing to index the string representations of all feature trees.

Query Processing

Partition

A partition SF_q of query graph q is a set of non-edge-overlapping subgraphs, where SF_q is a set of feature subtrees $\{tf_1, tf_2, ..., tf_n\}, tf_i \subseteq q$.

Filtering & Pruning

Filtering: SF_q is further filtered by intersecting the support set of all feature trees in SF_q, the result is called filtered set P_q. The underlying intuition is that if graph g does not contain any of q's subtree, then g won't contain q either.

Pruning: If there does not exist any set of feature subtrees in graph g, then g will be removed from P_q. Otherwise, all possible sets of feature subtrees satisfying the Center Distance Constraint will be used in final verification. The *Center Distance Constraints* is defined as: If q is subgraph isomorphic to graph g, then at least one set of feature subtrees $TP'_q = \{tp'_1, tp'_2, ..., tp'_m\}$ is embedded in g which satisfies $tp_i \approx tp'_i, 1 \leq i \leq m$, and $d(center(tp_i), center(tp_j)) \geq dg(center(tp'_i), center(tp'_j)), 1 \leq i \neq j \leq m$.

Verification

A new subgraph isomorphic test algorithm is proposed, which utilizes location information. First, a DPS is performed to find the feature subtrees rooted in the center vertices. Second, reconstruct query graph q by joining the subtrees one by one. Each time when the algorithm join a subtree, it checks if the new tree is isomorphic to its counterparts in query graph q. If they aren't isomorphic then move on to the next feature subtree. If the union of all subtrees in T'_q is isomorphic to q, then q is subgraph isomorphic to g.

Conclusion

The authors of this paper proposed a new indexing method for graph databases, which utilizes frequent feature trees as indexing pattern and developed mechanisms to filter and prune the feature tree set. The authors compared the TreePi Index with gIndex [2] which uses frequent structures as index patterns. Compared to gIndex, TreePi Index runs faster in both index construction and query processing.

References

[1] S. Zhang, M. Hu and J. Yang, "TreePi: A Novel Graph Indexing Method," *2007 IEEE 23rd International Conference on Data Engineering*, 2007, pp. 966-975, doi: 10.1109/ICDE.2007.368955.

[2] X. Yan, P. Yu, and J. Han. Graph indexing: a frequent structure-based approach, *Proc of SIGMOD,* 2004.