SING: Subgraph search In Non-homogeneous Graphs

Raffaele Di Natale [1], Alfredo Ferro [1], Rosalba Giugno [1],Misael Mongiovì [1],Alfredo Pulvirenti [1],and Dennis Shasha [2]

[1]Dipartimento di Matematica ed Informatica
Università di Catania
95125 Catania, Italy
{dinatale,ferro,giugno,mongiovi,pulvirenti}@dmi.unict.it

[2]Courant Institute of Mathematical Sciences
New York University
New York 10012, USA
shasha@cs.nyu.edu

ABSTRACT

Finding subgraphs of a graph database which are isomorphic to a given query graph has many practical applications in several fields, including cheminformatics and bioinformatics. Since subgraph isomorphism is a computationally hard problem, indexing techniques have been intensively exploited to speed up the process. Such systems filter out those graphs which cannot contain the query, and apply a subgraph isomorphism algorithm to each residual candidate graph. However, the applicability of such process is limited to databases of small graphs. When these indexing systems are applied to large graphs, their filtering power degrades and the subgraph isomorphism test is still expensive. In this paper, SING (Subgraph search In Non-homogeneous Graphs), a novel indexing system able to cope with large graphs, is presented. The method uses the notion of *feature*, which can be a small subgraph, subtree or path. Each graph in the database is annotated with the set of all its features. The key point is to make use of feature locality information. This idea is used to both improve the filtering performance and speed up the subgraph isomorphism task. Extensive tests on chemical compounds, biological networks and synthetic graphs show that the proposed system outperforms the most popular systems in query time over databases of medium and large graphs. Other specific tests show that the proposed system is effective for single large graphs.

General Terms

graph search, indexing, subgraph isomorphism, large graphs

1. INTRODUCTION

Graphs naturally model a multitude of complex objects of the real world. A chemical compound can be represented by a graph where atoms are vertices and bonds are edges. Biological networks model the complex of interactions among components in cells, (e.g. proteins, genes, metabolites). Social networks, the web, the water system and the power grid are all represented by graphs. The availability of algorithms and tools to manage and analyze graphs is crucial in various application domains. One of the basic problem is the search of a query graph in a target graph or, more generally, in a database of graphs. Searching a molecular structure in a database of molecular compounds is useful to detect molecules which preserve chemical properties associated with a well known molecular structure. This can be used in screening and drug design. Searching subnetworks in biological networks allows to identify conserved complexes, pathways and motifs among species, and assist in the functional annotation of proteins and other cell components.

The problem of searching for a query graph in a target graph is called *subgraph isomorphism* and is known to be NP-complete. Since the subgraph isomorphism test is expensive, screening all graphs of a large database can be unfeasible. Recently, indexing techniques for databases of graphs have been developed with the purpose of reducing the number of subgraph isomorphism tests involved in the query process. In a *preprocessing* phase the database of graphs is analyzed and a data structure called index is built. A query is processed in two phases. In the *filtering* step the index is used to discard the graphs of the database which cannot contain the query, producing a small set of *candidate* graphs. The set of candidates is then verified (*verification* step) by a subgraph isomorphism algorithm and all the resulting matches are reported.

Most graph indexing tools are based on the concept of *feature*. Depending on the particular system, a feature can be either a small graph [8, 1, 7], a tree [5] or a path [4, 3]. The filtering property is based on checking whether the features of the query are contained in each target graph. In the preprocessing phase the database of graphs is scanned, the features are extracted from each graph and stored in the in-

dex data structure. During the filtering phase, the features are extracted from the query and the index is probed in order to discard all graphs which do not contain some feature of the query.

Existing indexing techniques are effective on databases of small graphs but they become unfeasible when applied to huge graphs. Indeed such graphs generally contain numerous features spread over them. Consequently, each feature of a given query is very likely to be less frequent than it is in the large graph. This implies that filtering systems based only on the presence or number of features are not effective for large graphs. Moreover the subgraph isomorphism test over a large graph is considerably expensive. On the other hand, alternative indexing systems which do not make use of features [5, 11] show similar problems on large graphs.

To make the verification phase faster, GraphGrep [4] stores all the feature occurrences of each graph, and discards the part of the graph which does not contain features of the query. It restricts the search to small portions of the target graph. However, this produces a larger index which is more difficult to manage and can lead to a reduction of the filtering performance. Furthermore, the features of the query often occur in many areas of the graphs, limiting the portion which can be discarded.

In this paper, a novel approach to cope with large graphs is proposed. The position of a feature within the graph is considered. This additional information is used to both improve the filtering power and guide the verification phase toward parts of the graphs which can contain the query. The present approach makes use of paths as features. Differently from other systems using more complex features such as subgraphs or subtrees, our index includes all paths of bounded length. This structure is able to capture the topology of graphs and it is shown to perform better than the other systems in terms of query processing time. Moreover the size of the index is comparable to that of other existing systems. An exhaustive experimental analysis on real and synthetic data shows that the proposed system is efficient and effective on both databases of graphs and single large graphs.

The paper is organized as follows. In Section 2 the basic concepts are introduced. In Section 3 both feature-based and non-feature-based graph indexing systems are reviewed. Feature based graph indexing systems are considered in an unified framework and the differences among them are discussed. Section 4 presents the novel graph indexing system. In Section 5 the results of the experimental analysis are reported. Comparisons with the most popular systems over databases of chemical compounds show that our system is faster in processing queries of size greater than 4 on a database including large molecules. Additional results on gene regulatory networks and synthetic data are reported. For these data the proposed system outperforms all the other tools in terms of query time. Section 6 concludes the paper and addresses future directions.

2. PRELIMINARIES

In this paper, undirected node-labeled graphs are considered. However, the concepts introduced in what follows can be easily extended to edge-labeled and directed graphs. An undirected labeled graph (in what follows simply a graph) is a 4-tuple $g = (V, E, \Sigma, l)$ where V is the set of vertices, E, the set of edges, is a symmetric relation on V, Σ is the alphabet of labels and $l : V \to \Sigma$ is a function which maps each vertex into a label. We put $size(g) = |E|$ and indicate with $\mathcal{G}$ the set of all possible graphs. A graph g_1 is said to be a subgraph of g_2 iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$.

Given two graphs $g_1 = (V_1, E_1, \Sigma, l)$, $g_2 = (V_2, E_2, \Sigma, l)$ an *isomorphism* between g_1 and g_2 is a bijection $\phi : V_1 \to V_2$ so that:

- $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$
- $l(u) = l(f(u)) \forall u \in V_1$

A *subgraph isomorphism* between g_1 and g_2 is an isomorphism between g_1 and a subgraph of g_2. A graph g_1 is said to be isomorphic to another graph g_2 if there exist an isomorphism between g_1 and g_2. For the sake of simplicity we say also that g_1 is equivalent to g_2 and write $g_1 \approx g_2$. Notice that $\approx$ is an equivalence relation on $\mathcal{G}$. A graph g_1 is said to be subgraph isomorphic to another graph g_2 if there exist a subgraph isomorphism between g_1 and g_2. In this case we say that g_1 is *contained* in g_2 and write $g_1 \sqsubseteq g_2$.

In this paper, the following two problems will be discussed:

First_query_occurrence problem: Given a database of n graphs $D = \{g_1, g_2, ..., g_n\}$ and a query graph q, executing the query q on D is equivalent to find all graphs g of D such that q is subgraph isomorphic to g. In the following we assume, without loss in generality, that all graphs of D and the query graph, share the same alphabet Σ.

All_query_occurrences problem: Given a database of n graphs $D = \{g_1, g_2, ..., g_n\}$ and a query graph q, executing the query q on D is equivalent to find all subgraph isomorphisms between q and elements of D.

We will make extensive use of the notion of feature. Features are defined formally by the following definition.

Definition 1. Let $\mathcal{G}$ be the set of all possible graphs in a given alphabet of labels. A set $\mathcal{F}$ is a *set of features on* $\mathcal{G}$ iff there exists a binary relation $is\_a\_feature \subseteq \mathcal{F} \times \mathcal{G}$ such that the following property holds (*graph upward monotonicity*):

$$\forall f \in \mathcal{F}, q, g \in \mathcal{G},$$
$$is\_a\_feature(f, q) \wedge q \sqsubseteq g \rightarrow is\_a\_feature(f, g)$$

Every set of features defines a *pruning rule* for the subgraph isomorphism problem. Indeed if $is\_a\_feature(f, q)$ and $\neg\ is\_a\_feature(f, g)$ then q cannot be subgraph isomorphic to g.

Examples of set of features are:

- The set $Paths_{\leq k}$ of all labeled paths of length $\leq k$. Here a labeled path is the sequence of labels.

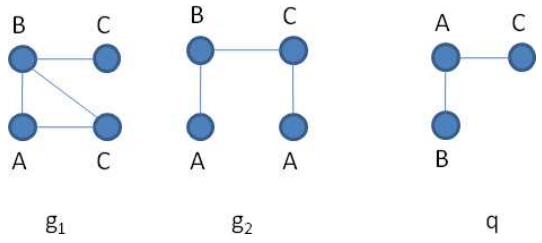

Figure 1: A database of two graphs g_1, g_2 and a query q. $q \sqsubseteq g_1$ but $q \not\sqsubseteq g_2$.

- The set $Subtrees_{\leq k}$ of all labeled subtrees of depth $\leq k$.
- The set $Subgraphs_{\leq k}$ of all labeled subgraphs of size $\leq k$.

In this paper the set of features $Paths\_occ_{\leq k}$ of pairs (p, n), where p is a labeled path of length $\leq k$ and n is a lower bound in the number of occurrences of p in the given graph, is considered. The corresponding pruning property asserts that if the query graph q contains at least n occurrences of a given labeled path p and g does not contain at least n occurrences of p, then q cannot be subgraph isomorphic to g and g can be pruned.

Notice that in all above examples if a feature f is a subfeature of a given feature f' of g then f' is also a feature of g. The following definition formalizes this notion.

A *downward monotonic* set of features is a partially ordered set of features $(\mathcal{F}, \preceq)$ such that:

$$\forall f, f' \in \mathcal{F}, g \in \mathcal{G},$$
$$f \preceq f' \wedge is\_a\_feature(f', g) \rightarrow is\_a\_feature(f, g)$$

For instance $Paths_{\leq k}$ is a downward monotonic set of features with respect to the subsequence relation between labeled paths. $Paths\_occ_{\leq k}$ is downward monotonic with respect to the number of occurrences. However it is not downward monotonic with respect to the subsequence relation. Indeed in Figure 1, (ABC,2) is a feature of g_1 but (AB,2) is not a feature of g_1.

A downward monotonic set of features allows an additional optimization in the pruning process. Indeed, the pruning rule can be restricted only to maximal features f in the query. This means that no other feature f' in the query can be strictly greater than f in the partial order of features.

3. RELATED WORKS
4. A NEW APPROACH BASED ON FEATURE LOCATION
5. EXPERIMENTAL RESULTS
6. CONCLUSIONS
7. ACKNOWLEDGMENT
8. REFERENCES

[1] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. *Proceedings of ACM SIGMOD international conference on Management of data*, pages 857 – 872, 2007.

[2] L. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[3] A. Ferro, R. Giugno, M. Mongiovi, A. Pulvirenti, D. Skripin, and D. Shasha. Graphfind: enhancing graph searching by low support data mining techniques. *BMC Bioinformatics*, (9), 2008.

[4] R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs, 2002.

[5] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 38, Washington, DC, USA, 2006. IEEE Computer Society.

[6] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[7] D. W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 976–985, 2007.

[8] X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *ACM Transactions on Database Systems*, 30(4):960–993, 2005.

[9] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. *Proceedings of IEEE 23rd International Conference on Data Engineering*, pages 181–192, 2007.

[10] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: tree + delta $\leq$ graph. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 938–949. VLDB Endowment, 2007.

[11] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 181–192, New York, NY, USA, 2008. ACM.