Towards a Subgraph/Supergraph Cached Query-Graph Index

Jing Wang, Nikos Ntarmos, Peter Triantafillou
School of Computing Science, University of Glasgow, Glasgow, UK
Email: j.wang.3@research.gla.ac.uk, {nikos.ntarmos, peter.triantafillou}@glasgow.ac.uk

I. Introduction and Motivation

Many modern big data applications deal with graph structured data, such as databases of molecular compounds represented as graphs of atoms and bonds, or "structured interaction networks" in biological and social networks, where nodes refer to entities (proteins, people, etc.) and edges represent their relationships. Central to high performance graph analytics over such data, is to locate patterns in dataset graphs. Informally, given a graph dataset and a query (a.k.a. pattern) graph g, the goal is to return stored graphs that contain g (subgraph querying) or are contained in g (supergraph querying). These operations are costly, as they entail the NP-Complete subgraph isomorphism problem[1]. This is further aggravated when the dataset consists of a large number of graphs, as testing g for subgraph isomorphism against all of them would require a very large amount of time.

The prominent solution to subgraph/supergraph querying follows a *filter and verify* paradigm. Figure 1 illustrates the case of subgraph querying, comprising three major stages:

1) Indexing: Dataset graphs are reduced to their features (i.e., substructures of the graph, such as paths, trees, cycles, or arbitrary subgraphs), which are then indexed in an appropriate data structure (index).
2) Filtering: Query graph g is decomposed into its features, which are then compared against the index, so as to drop stored graphs that do not contain all features of g and produce the query's candidate set $CS(g)$.
3) Verification: Filtering can produce false positives – i.e., graphs in $CS(g)$ do contain all features of g but not necessarily g itself; thus each of these graphs needs to undergo an expensive subgraph isomorphism test vs g.

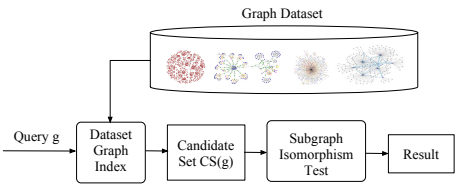

Figure 1. The *filter and verify* paradigm

In most real-world scenarios the verification stage is by far dominating the query processing time, and thus related work focuses on reducing the candidate set size. Approaches can be characterized by the way they produce features (e.g., data mining for frequent and discriminative features, exhaustive enumeration of all features up to a limit feature size) and feature types (e.g., paths, trees, cycles, general graphs, etc.). Mining based methods ([2], [3], [4]) utilize data mining techniques to produce features. gIndex[5] and FG-Index[6] mine for frequent (sub)graphs among the dataset graphs, while Tree+Δ[7] and TreePi[8] mine for and index frequent trees (Tree+Δ can also index (sub)graphs). Such approaches tend to mine for complex structures, which are expected to have higher filtering power and thus pay off the expensive mining cost of the indexing stage. Enumeration based methods SING[9], GraphGrep[10], GraphGrepSX[11] and Grapes[12] list all paths of dataset graphs, whereas CT-Index[13] indexes trees and cycles. Exhaustive enumeration can yield a large amount of features and thus make the index too large to fit in memory. As a result, all exhaustive enumeration approaches limit the size of features, usually to a small number of edges (i.e., 10 or less). In addition, some approaches (e.g., [9], [12], and [8]) manage to utilize location information stored in the index (for those graphs that survive the filtering process) to achieve further pruning. Essentially, all these approaches are utilizing the knowledge of dataset graphs characteristics to answer queries.

The key observation for our work is the realization that in real applications, queries share subgraph or supergraph relationships with each other. Think of the hierarchy of queries in protein-interaction datasets, where query graphs can vary from aminoacids, and proteins, to protein mixtures and proteins of uni- or even multi-cell bacteria and organisms. Up to now, this natural relationship among queries has not been exploited.

II. Our Proposal

We propose a solution, coined iGQ, which can identify whether a new query g is a subgraph or supergraph of previously issued queries, and reuses the (positive and negative) results of the latter to expedite g. The goal is for iGQ to be incorporated into any subgraph or supergraph query processing framework. It will piggyback on said framework (let us call it method $\mathbb{M}$) and augment it with the iGQ index $\mathbb{I}$. The latter consists of two components, $\mathbb{I}_{sub}$ and $\mathbb{I}_{super}$, usable for answering subgraph and supergraph queries respectively.

Figure 2 illustrates how the system operates. Method $\mathbb{M}$ is used to index the dataset graphs, while $\mathbb{I}$ starts off empty and is populated as new queries are processed. Upon the arrival

of a query g, the query processing proceeds in three parallel threads. One thread employs method M to decompose g into its features and to produce the candidate set $CS(g)$. The two remaining threads operate on $\mathbb{I}_{sub}$ and $\mathbb{I}_{super}$ to identify related previous queries and retrieve their results. These three components combined, lead to significant reductions in the number of subgraph isomorphism tests by further pruning $CS(g)$.

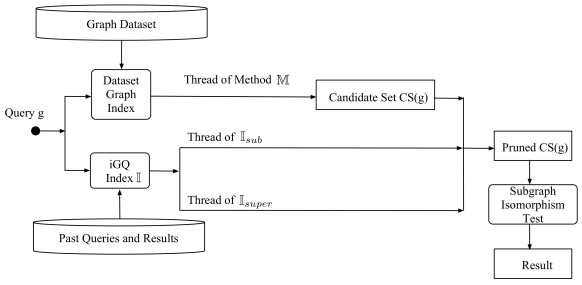

Figure 2. The iGQ framework

Operations on $\mathbb{I}_{sub}$ and $\mathbb{I}_{super}$ actually represent a microcosm of the original problem, i.e., subgraph/supergraph query processing. Rather than indexing the dataset graphs, here we index previous query graphs and derive the subset of them which contain the query graph (subgraph querying) or are contained in the query graph (supergraph querying). For this reason, any approach from the related works can be adapted to process $\mathbb{I}$.

III. Performance Evaluation

We tested the performance gains of an initial implementation of iGQ on several high performance subgraph querying methods, including GGSX[11], Grapes[12], and CT-Index[13], over real datasets. For Grapes, we present two alternatives: Grapes and Grapes(6), with the latter using 6 threads to improve query processing with parallelism.

The two real-world datasets we employed exhibit different characteristics. AIDS[14] contains topological structures of 40,000 molecules. Graphs in AIDS are small, with on average ≈45 vertices (std.dev.: 22, max: 245) and ≈47 edges (std.dev.: 23, max: 250) per graph. PDBS[15] is a database of 600 graphs representing DNA, RNA and proteins, with on average ≈2,939 vertices (std.dev.: 3,217, max: 16,431) and ≈3,064 edges (std.dev.: 3,264, max: 16,781) per graph.

We create the workloads, following the method for generating queries as found in the related works using these datasets[16]. Here, queries are generated from the original dataset graphs using random walks as follows. First, a graph is selected uniformly at random from the dataset graphs. Then an edge is selected uniformly at random within this graph. Finally, a random neighboring edge is selected uniformly at random and this continues so until the chosen number of edges is reached. Once a set of query graphs of a specific size is generated, then it is used to derive the query graphs of the next smaller size, etc. For AIDS dataset,

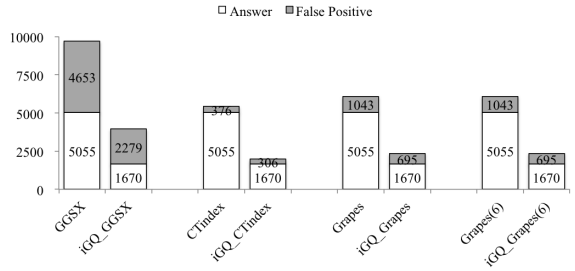

Figure 3. Average number of subgraph isomorphism tests for AIDS.

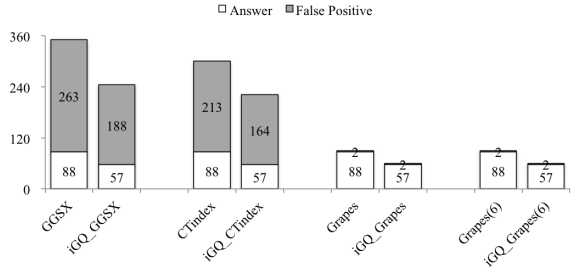

Figure 4. Average number of subgraph isomorphism tests for PDBS.

we generated queries having a predefined size (4, 8, 12, 16, and 20 edges). A similar process was followed for PDBS, except that query sizes consisted of 20, 40, 60, 80, and 100 edges (as the dataset graphs of PDBS are much larger). Each workload consists of 500 queries, 100 per query size.

We first examine the *filtering power*, which is reflected by the number of subgraph isomorphism tests to perform. The following figures show for each algorithm the number of subgraph isomorphism tests performed, broken down into the number of *false positives* (i.e., candidate graphs which were not in the final answer set, shown as "False Positive") and the number of *true positives* (i.e., candidate graphs which were indeed in the final answer set, denoted as "Answer"). Figures 3 and 4 show the case of AIDS and PDBS, respectively. We stress that iGQ reduces both the number of *false positives* and the number of *true positives* that need to undergo subgraph isomoprhism testing.

We now turn to *query processing time*, incurred by the four methods and their iGQ counterparts. Figures 5 and 6 show the query processing time of AIDS and PDBS, in line with corresponding figures on the number of subgraph isomorphism tests. Note that the filtering time overhead of iGQ is always very low, testifying that the overhead of processing queries for the subgraph and supergraph cases is a non-factor in overall query processing time.

Figures 7 and 8 quantify the performance improvements introduced by iGQ over all datasets/workloads and across all four methods. It is instructive to analyze figures 7 and 8 together. The key is that reductions in the number of required subgraph isomorphism tests achieved by iGQ do not translate necessarily into equivalent reductions in the average query times.

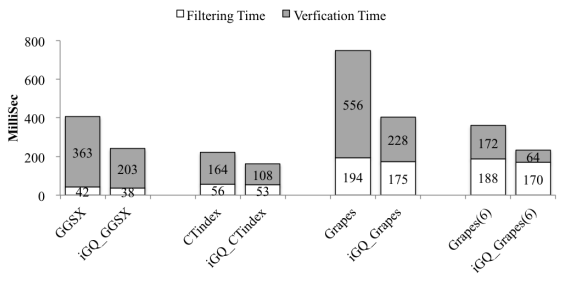

Figure 5. Average query processing time in AIDS.

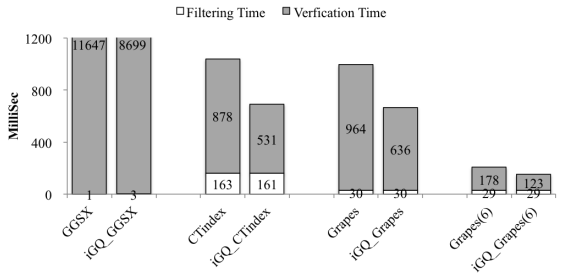

Figure 6. Average query processing time in PDBS.

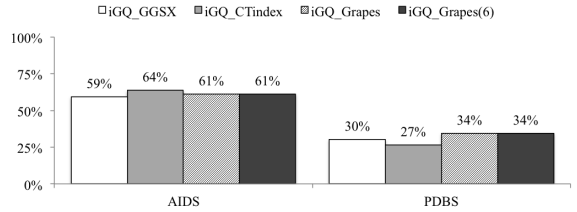

Figure 7. Average subgraph isomorphism tests reduction across workloads over GGSX, CT-Index, Grapes/(6).

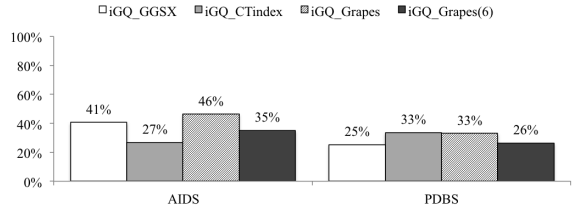

Figure 8. Average query processing time reduction across workloads over GGSX, CT-Index, Grapes/(6).

Finally, it is important to note that in the PDBS case, Grapes(6) outperforms CT-Index (see Figure 4 and 6), whereas the reverse holds in the AIDS case (Figure 3 and 5). This leads to an important conclusion: Among the best performing methods in the literature, no method is a clear winner in all datasets/workloads. So the elusive goal is to achieve solid performance regardless of the dataset/workload.

IV. Conclusion and Challenge

In a nutshell, iGQ has shown great potential to exploit sub/supergraph relationships among queries and utilize the results of past queries to expedite future query processing. iGQ acts as an accelerator of method $\mathbb{M}$ with impressive performance gains, by removing unnecessary subgraph isomorphism tests. Such unnecessary tests are not only caused by false positives, but also graphs that are true positive. Remaining challenges entail the formal proofs of correctness for iGQ, efficient algorithms and structures of its main threads, and the definition of proper algorithms for the management of iGQ contents. The latter crucially contains a replacement policy for (query) graphs, which fundamentally differs from standard cache replacement policies.

References

[1] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[2] C. Chen *et al.*, "Towards graph containment search and indexing," in *Proc. VLDB*, 2007.

[3] G. Zhu *et al.*, "Prefindex : An efficient supergraph containment search technique," in *Proc. SSDBM*, 2010.

[4] S. Zhang, J. Li, H. Gao, and Z. Zou, "A novel appropre-findexach for efficient supergraph query processing on graph databases," in *Proc. EDBT*, 2009.

[5] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *Proc. ACM SIGMOD*, 2004.

[6] J. Cheng, Y. Ke, W. Ng, and A. Lu, "FG-index: towards verification-free query processing on graph databases," in *Proc. ACM SIGMOD*, 2007.

[7] P. Zhao, J. X. Yu, and P. S. Yu, "Graph indexing: tree + delta >= graph," in *Proc. VLDB*, 2007.

[8] S. Zhang, M. Hu, and J. Yang, "TreePi: A Novel Graph Indexing Method," in *Proc. IEEE ICDE*, 2007.

[9] R. Di Natale *et al.*, "Sing: Subgraph search in non-homogeneous graphs," *BMC bioinformatics*, vol. 11, no. 1, 2010.

[10] R. Giugno and D. Shasha, "GraphGrep: A fast and universal method for querying graphs," in *Proc. IEEE ICPR*, 2002.

[11] V. Bonnici, et al., "Enhancing graph database indexing by suffix tree structure," in *Proc. IAPR PRIB*, 2010.

[12] R. Giugno *et al.*, "Grapes: A software for parallel searching on biological graphs targeting multi-core architectures," *PloS One*, vol. 8, no. 10, 2013.

[13] K. Klein, N. Kriege, and P. Mutzel, "CT-index: Fingerprint-based graph indexing combining cycles and trees," in *Proc. IEEE ICDE*, 2011.

[14] NCI - DTP AIDS antiviral screen dataset, "http://dtp.nci.nih.gov/docs/aids/aids_data.html."

[15] Y. He, et al., "Structure of decay-accelerating factor bound to echovirus 7: a virus-receptor complex," *PNAS*, vol. 99, pp. 10 325–10 329, 2002.

[16] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu, "iGraph: A framework for comparisons of disk-based graph indexing techniques," *PVLDB*, vol. 3, no. 1-2, pp. 449–459, 2010.