Lindex: a lattice-based index for graph databases

Dayu Yuan · Prasenjit Mitra

Received: 18 October 2011 / Revised: 31 May 2012 / Accepted: 13 June 2012 / Published online: 15 September 2012
© Springer-Verlag 2012

Abstract Subgraph querying has wide applications in various fields such as cheminformatics and bioinformatics. Given a query graph, \( q \), a subgraph-querying algorithm retrieves all graphs, \( D(q) \), which have \( q \) as a subgraph, from a graph database, \( D \). Subgraph querying is costly because it uses subgraph isomorphism tests, which are NP-complete. Graph indices are commonly used to improve the performance of subgraph querying in graph databases. Subgraph-querying algorithms first construct a candidate answer set by filtering out a set of false answers and then verify each candidate graph using subgraph isomorphism tests. To build graph indices, various kinds of substructure (subgraph, subtree, or path) features have been proposed with the goal of maximizing the filtering rate. Each of them works with a specifically designed index structure, for example, discriminative and frequent subgraph features work with gIndex, \( \delta \)-TCFG features work with FG-index, etc. We propose Lindex, a graph index, which indexes subgraphs contained in database graphs. Nodes in Lindex represent key-value pairs where the key is a subgraph in a database and the value is a list of database graphs containing the subgraph. We propose two heuristics that are used in the construction of Lindex that allows us to determine answers to subgraph queries conducting less subgraph isomorphism tests. Consequently, Lindex improves subgraph-querying efficiency. In addition, Lindex is compatible with any choice of features. Empirically, we demonstrate that Lindex used in conjunction with subgraph indexing features proposed in previous works outperforms other specifically designed index structures. As a novel index structure, Lindex (1) is effective in filtering false graphs, (2) provides fast index lookups, (3) is fast with respect to index construction and maintenance, and (4) can be constructed using any set of substructure index features. These four properties result in a fast and scalable subgraph-querying infrastructure. We substantiate the benefits of Lindex and its disk-resident variation Lindex+ theoretically and empirically.

1 Introduction

Graphs are widely used to model structures and relationships of objects in various scientific and commercial fields. For instance, chemical molecules are modeled as graphs [16], and three-dimensional mechanical parts are stored and manipulated as attributed graphs in a CAD-mechanical-components database [8]. Graphs are also used in pattern recognition and have broad applications in computer vision and image processing [6].

A popular method of retrieving graphs from graph databases is by performing a subgraph query. Given a graph database, \( D \), and a query graph, \( q \), a subgraph-querying algorithm retrieves all graphs \( g \in D \) containing \( q \) as a subgraph. Deciding whether one graph is a subgraph of another is referred to as the subgraph isomorphism problem; the problem has been shown to be NP-complete [7]. Consequently, for large databases, an index is necessary to enable efficient query processing. Typical graph indices are sets of key-value pairs. The key is a subgraph of a graph in the database, and the value consists of a list of database graphs that contain the subgraph. When the query graph is a key in the index, the value can be returned directly as the answer. Otherwise, the
index is used to return a candidate set \( C(q) \) of graphs that may potentially contain the query. \( C(q) \) is typically larger than the answer set \( D(q) \), \(|C(q)| \geq |D(q)|\). A subgraph isomorphism test is performed to check that \( q \) is contained in each candidate graph in \( C(q) \). Thus, \(|C(q)| \) subgraph isomorphism tests must be performed to eliminate graphs that are not in \( D(q)(D(q) \) will also be referred to as the supporting set of \( q \) in the rest of the paper). Existing methods for subgraph indexing and querying use such a filtering + verification paradigm [5,13–15,19,21,23]. Previous works have mainly focused on mining “good” substructure features for indexing.\(^1\) A good feature set improves the filtering power by reducing the number of candidate graphs, which leads to a reduction in the number of subgraph isomorphism tests in the verification step [5,15,19]. Subtree features are also mined for indexing, and they are less time-consuming to be mined in comparison with subgraph features [21,23].

In related works [5,13,19,23], different graph index structures have been used for different kinds of features; no index structure is general enough to support all kinds of substructure features. For example, glIndex [19] cannot index the \( \delta \)-TCFG features (which is designed specifically for FG-index [5]) because it does not support the search for the closest \( \delta \)-TCFG supergraph of the query as needed by the FG-index method. At the same time, FG-index cannot index the discriminative and frequent subgraph features of glIndex efficiently, since no apriori pruning can be made during the index lookup [19]. For the same reason, glIndex cannot support MimR features [15] efficiently. We show in Sect. 7 that using glIndex to index MimR features results in a significant increase in the index-lookup cost, which dominates the overall query-processing time when the query is a relatively large graph. The index structure of TreePi [21] only supports subtree features. The iGraph framework benchmarked existing indexing methods and concluded that “there is no single winner for all experiments” [9]. This observation motivates the need for a graph index that can be implemented in a graph database management system and is independent of the features being indexed. The DBMS can have an extensible library of feature-selection strategies that are application-dependent and any of them can be chosen to use with the index structure.

We propose and evaluate the lattice-structure index, Lindex.\(^2\) In Lindex, each node is associated with a key-value pair. A key is a (substructure) feature, say \( sg \), and its value set \( V \), as in an inverted index, is the set of database graphs (IDs) that contain \( sg \). In Lindex, an edge between two index nodes indicates that the key in the parent node is a subgraph of the key in the child node. In Fig. 1, we show a simple Lindex. The root node in the Lindex has a key \( sg0 \), the empty graph (a graph with no nodes or edges). The node \( sg0 \) has two children nodes with keys \( sg1 (sg0 \subset sg1) \) and \( sg3 (sg0 \subset sg3) \).

We show in detail how Lindex is designed and implemented independent of the choice of index features. Lindex provides the following: (1) high filtering power (Sect. 4), (2) fast index-lookup strategies (Sect. 5), (3) compact memory consumption (Sect. 3), and (4) fast index construction and maintenance (Sect. 6.3).

**High filtering power:** In previous methods [5,13–15,19,21,23], when a query graph was not a key in the index, the number of subgraph isomorphism tests needed to answer a query \( q \) was at least \(|D(q)|\). Can we design a method that may require fewer subgraph isomorphism tests than the size of the answer set \(|D(q)|\)? We can do so using Lindex. Lindex utilizes the fact that database graphs that contain a supergraph of a query \( q \) are guaranteed to be in the answer set for \( q \); those graphs do not need to be checked for subgraph isomorphisms. Consider the example in Fig. 1. Let \( q \) be the query. Given the Lindex, \( maxSub(q) \) contains the maximal-subgraph features of \( q \), which are also direct parents of \( q \), namely, \( sg2 \) and \( sg3 \). The intersection of their value sets \([a, b, c, e, f]\) and \([a, b, d, e, f]\) is \([a, b, e, f]\), which is the candidate set of answers. Our algorithm also finds the minimal-supergraph features of \( q \), \( minSup(q) \). In our example, the minimum supergraph of \( q \) (direct child of \( q \)) is \( sg5 \) whose value set is \([b, e, f]\). From the Lindex, by construction, we know that \( b, e, f \) contain \( sg5 \). Therefore, the database graphs \( b, e, f \) are guaranteed to contain \( q \). Hence, we can directly put \([b, e, f]\) in the answer set resulting in saving three costly subgraph isomorphism tests. In our method, only the graph \( a \) has to be checked, while all previous works have to check all of \([a, b, e, f]\). No indices in previous works support the minimal-supergraph-feature lookup, except for the FG-index. However, FG-index returns only one supergraph of \( q \) (closed \( \delta \)-TCFG supergraph) [5], while our proposed Lindex returns all minimal-supergraph features of queries.

---

1 Substructure features will be simply referred to as features in the rest of the paper.
2 A preliminary version of Lindex was reported in an online-only workshop proceeding [20].