Knowledge-Based Systems 72 (2014) 48–59

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Graph indexing for large networks: A neighborhood tree-based approach

Zhen Lin [a,b,*], Yijun Bei [c]

[a] College of Computer Science & Technology, Zhejiang University, Hangzhou 310027, China
[b] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[c] School of Software Technology, Zhejiang University, Hangzhou 310027, China

ARTICLE INFO

Article history:
Received 17 March 2014
Received in revised form 4 August 2014
Accepted 29 August 2014
Available online 16 September 2014

Keywords:
Graph querying
Graph matching
Neighborhood tree
Indexing
Social network

ABSTRACT

Graphs are used to model complex data objects and their relationships in the real world. Finding occurrences of graph patterns in large graphs is one of the fundamental graph analysis tools used to discover underlying characteristics from these complex networks. In this paper, we propose a new tree-based approach for improving subgraph-matching performance. First, we introduce a new graph indexing mechanism known as **N**eighborhood **Tree**s (NTree), which records the neighborhood relationships of each vertex in the large graph to filter negative vertices. Second, we decompose a query graph into a set of neighborhood trees and only a subset of candidate trees, which can properly recover the original query graph. In this way, the tree-at-a-time method is used to obtain the matched graphs. Third, we employ a graph query optimizer to determine the neighborhood tree selection order on the basis of the cost evaluation of tree join operations. Experiments on both real and synthetic databases demonstrate that our approach is more efficient than other state-of-the-art indexing methods.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Graphs are used to model many complex data objects and their relationships in the real world. In recent years, an increasing number of large networks such as social networks, chemical compounds, semantic web, and protein networks have appeared. The sizes of these large graphs can be in excess of millions of vertices and edges. Methods for managing, processing, and analyzing these graph data have become important research topics. Finding the occurrences of graph patterns or subgraphs in large graphs is one of the fundamental uses for graph analysis tools because they denote underlying characteristics in complex networks. Subgraph query or matching approaches have been widely used in areas such as chemical informatics [1], proteins analysis [2], biochemistry [3], web applications [4], and computer vision [5]. For example, given a large protein network, biologists may want to determine all occurrences of structural motifs in 3D proteins by using protein contact maps [6].

Graph searching is an important task in a variety of applications and falls under two scenarios: subgraph querying and subgraph matching. The classical graph-querying problem is to find all supergraphs of the query pattern from a graph database, whereas graph matching involves finding all subgraphs of the database graph, which are isomorphic to the query graph. It is clearly inefficient to perform an exhaustive search in the database because the subgraph isomorphism itself is a non-deterministic polynomial time (NP)-complete problem.

Graph indexing is a common technique for performing searches in large graph databases. Many indexing mechanisms such as gIndex [7], Tree+Δ [8], TreePi [9], FERRARI [10], and frequent subgraph (FG)-index [11] have been developed for subgraph querying processes. In these approaches, frequent features are extracted from graphs and are leveraged to build graph indices. As a result, expensive preprocessing is required due to the frequent pattern mining processes in index construction such as paths, trees, and subgraphs. In addition, non-mining based techniques, such as Closure-tree [12], GCoding [13], LnGCoding [14], and GiS [15], have been proposed for graph queries. For the purpose of indexing, structural information of graphs are mapped into graph signatures such as numerical space in GCoding or line graphs in GiS. However, such indexing was developed mainly for searching graphs in a large number of small graphs.

To solve the problem of graph matching in a large graph, neighborhood signature-based techniques such as GraphQL [16], SPath [17], and GADDI [18] have been proposed. In GraphQL [16], profiles

* Corresponding author at: College of Computer Science & Technology, Zhejiang University, Hangzhou 310027, China.
E-mail addresses: linz@illinois.edu, nblinz@gmail.com (Z. Lin), byj@zju.edu.cn (Y. Bei).

http://dx.doi.org/10.1016/j.knosys.2014.08.025
0950-7051/© 2014 Elsevier B.V. All rights reserved.

around a vertex neighborhood are used for local pruning, and global structural information is leveraged to simultaneously reduce the overall search space. In SPath [17], the shortest paths around a vertex neighborhood are leveraged as basic indexing units, and a more efficient path-at-a-time method is introduced to process graph queries. In GADDI [18], neighboring discriminating substructure (NDS) distance measurement is adopted as the basis of the pruning method, and an index-based graph matching method is proposed for achieving high pruning power and linear size scales. To reduce search space, the cost model and query plan optimizer are both employed in these up-to-date approaches. However, when using neighborhood profiles or paths as indexing units to prune negative vertices, structural information around the vertices may be lost. As a result, false positive vertices remain as candidates and require further filtering. Moreover, too many join operations are required compared with more complex structures such as trees or subgraphs.

In this paper, we propose a new tree-based approach for improving subgraph searching performance. First, we introduce a new graph-indexing mechanism known as **N**eighborhood **Tree**s (NTree), which records the neighborhood relationships of each vertex in the large graph to filter negative vertices. Because trees contain more information than paths or vertex profiles, the NTree has stronger pruning power. Second, we decompose a query graph into a set of trees, and only a subset of candidate subtrees that can properly recover the original query graph are selected. These candidate subtrees are then joined to reconstruct the query graph. In this way, the tree-at-a-time method is used to obtain the matched graphs. Third, we employ a graph query optimizer to determine the vertex searching order on the basis of the cost evaluation of tree joining operations. Our work has the following contributions:

1. We propose a structural pattern-based graph-matching framework. First, candidate vertices are identified. Then, a joining process by comparing relationships of candidate vertices in the large graph to the original query graph is used to further verify these candidates. Both vertex pruning and query reconstructing abilities of different structural patterns such as neighborhood paths, trees, and graphs are evaluated. As a result, neighborhood trees are selected to be the most feasible candidates (Section 3).
2. To reduce the number of candidates, we propose a new cost-effective graph indexing technique, NTree, which makes use of trees around the vertex neighborhood for pruning purposes. Canonical unordered trees are leveraged, and the string comparison technique is used to accelerate the subtree containment process (Section 4).
3. To address the second step, we propose an efficient searching method for joining the neighborhood trees and reconstructing the original query graph. In addition, we design a graph query cost model on the problem of neighborhood tree selection to optimize the search order (Section 5).
4. We conduct extensive experiments by using both real and synthetic databases. We compare our indexing method, NTree, with state-of-the-art path-based indexing methods. The results show that our method outperforms these indexing methods in terms of graph matching performance (Section 6).

2. Related works

The subgraph isomorphism test [19,20,5,21] is a well-known NP-complete problem that has been widely studied in recent years. For subgraph searching, a large number of index-based graph matching and searching frameworks have been proposed including gIndex [7], TreePi [9], FG-index [11], NB-index [22], LW-index [23], Tree+Δ [8], GCode [13], GPTree [24], Closure-tree [12], Turboiso [25], SODA [26], SING [27], and GiS [15]. These graph-indexing approaches have been designed mainly for performing subgraph querying from a graph database consisting of many small- or medium-sized graphs. Most of these indexing approaches such as gIndex [7], Tree+Δ [8], and FG-index [11] all make use of frequent patterns in the graph database as the basic indexing structure. Consequently, expensive preprocessing is required to mine frequent patterns when constructing graph indices. The encoding method GCode [13] assigns a signature known as level-n path tree to each vertex based on its local structures. As a result, graph codes are obtained by combining all vertex signatures. SING [27] uses the concept of features and makes use of feature locality information. gStore [28] transforms a Resource Description Framework (RDF) graph into a data signature graph and uses the vertex signature (VS)*-tree index with light maintenance overhead. A filtering rule in gStore is also developed for answering exact SPARQL queries in a uniform manner. In this paper, a different encoding scheme is introduced. We make use of the level-wise signature that records the neighborhood edge to construct the neighborhood tree as the graph index. Neighborhood vertices arranged in the form of rings are recorded as well as the occurring number of edges in each level.

To address the inexact matching problem, Grafil [29] clusters features according to their selectivities and applies a multi-filter strategy. Edge deletions are transformed into feature misses, and an upper bound is used on the maximum number of allowed feature misses for graph filtering. Zhu et al. [30] proposed a novel search paradigm, TreeSpan, to conduct similarity all-matching that conforms a similarity threshold θ by first conducting exact all-matching on a minimal set of spanning trees. A rigid theoretic analysis shows that this approach can significantly reduce the time required for conducting exact all-matching compared with the existing techniques. In SAGA [31], Tian et al. proposed a more flexible indexing approach that can support both vertex insertions and deletions. A flexible graph distance model is employed to measure similarities between graphs, and matched fragments are assembled into large matches. However, in this similar graph searching approach, only a subset of approximate matching results is obtained. Ness [32] is another tool for inexact matching that focuses on the $top-k$ approximate matches. In this method, a neighborhood-based similarity measure is proposed that avoids costly graph isomorphisms and edits distance computation. SLQ [33] is a framework enabling schemaless and structureless graph query that can automatically learn an effective ranking model with no manual preprocessing. This method returns matches by using graph sketches and belief propagation. The simB method [34] is proposed for edit distance-based similarity search problems, whereby a lower bound based on the branch structure is proposed to reduce the search space, and the b-tree index is adopted to facilitate the query processing. GSimSearch [35] is another efficient algorithm for graph similarity query. Unlike the simB method, GSimSearch exploits the number of common fixed-length paths between pairs of graphs and also adopts degree-associated structural information to enhance runtime performance.

In addition to the searching subgraph problem from a large number of small graphs, methods used to search a subgraph in a large graph such as a social network are also addressed [16,17,36,37,18,38,39]. Several up-to-date approaches such as GraphQL [16], SPath [17], and GADDI [18] are proposed to obtain all occurrences of a query in a large graph. In GraphQL [16], neighborhood profiles are first employed to prune vertices individually. Then, the overall search space by considering all vertices in the pattern is simultaneously reduced. In SPath [17],

Download English Version:

https://daneshyari.com/en/article/403579

Download Persian Version:

https://daneshyari.com/article/403579

Daneshyari.com