Accelerating Subgraph Search by using synchronous update

Siyu Yuan 17307110448

Abstract

Recently, subgraph searching have been widely used in the fields of biomedicine and social networking. However, Most researches on subgraph searching focus on subgraph searching over graph database. In fact, subgraph searching on a single graph is also very valuable. In this paper, I propose an algorithm to accelerate subgraph query on a large graph by using synchronous update which effectively relieves the memory pressure and speeds up the query. The experimental results show that the synchronous update method is better than the non-synchronous update method both in offline and online.

Keywords: Subgraph Search, synchronous update, Pruning

1 INTRODUCTION

In recent years, subgraph search has been applied in more and more fields such as social network, semantic web and biomedicine. There are two types of targets for subgraph queries. One is to search for specific query maps on a graph database, for example, searching for compounds containing specitic substructures related to biological activity from a compound database, where each compound corresponds to a smaller data map. The other is to search for a specific query graph on a single large graph. For example, given a structure, locating its appearance on a large biological network[1].

Up to now, there are many mature algorithms to solve this problem, such as subgraph isomorphism, graph simulation, strong simulation and so on. However, graphs in real life pose some challenges for subgraph search. On one hand, The size of a real network graph is very large. For example, Facebook reached 1 billion users on October 4, 2012. Another example is Yahoo Web graphics, which includes 1.4 billion vertices and 6.6 billion edges[2]. They can easily reach PB size. On the other hand, the time complexity of subgraph search is very high. For a largh graph $G = (V, E)$ and a largh query Q, it is even NP-hard to find the matching subgraph of Q in G when using the subgraph isomorphism algorithm.

In this paper, I propose SSSU, an algorithm to accelerate subgraph search by using synchronous update. All in all, I have made the following contributions in this paper,

(1) I propose a method that speeds up subgraph queries by combining synchronous update ideas.

(2) I explore the effect of reducing search space on speeding up subgraph queries.

(3) I associate the Clique query method with the subgraph query problem, and at the same time, fuse the previous algorithms to effectively optimize the subgraph query problem.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 gives some important definitions about the problem and makes preliminaries. Section 4 explains our synchronous update. The experimental results are presented and discussed in Section 5. Finally I conclude this paper in Section 6.

2 RELATED WORK

In theory, the subgraph search problem belongs to the NP-hard problem. Therefore, how to solve the subgraph search problem efficiently has aroused widespread concern in the academic community. The current subgraph search algorithm can be divided into the following three types,

Subgraph isomorphism algorithm: It transforms the subgraph search problem into a subgraph Isomorphism problem. The Ulman[3] algorithm and the VF2[4] algorithm are both typical algorithms of subgraph Isomorphism problems. The former uses a matrix and the latter uses a point pair to transform a graph problem into a numerical problem. These two algorithms have simple ideas, but they have high time complexity. QuickSI[5] efficiently prune the search through synchronous depth-first traversal.

Feature-Based algorithm: Since sequential scan is not scalable, an indexing mechanism is required. Obviously, if the graph G contains query graph Q, G should contain any substructure of Q. Therefore, I can index substructures of a query graph to prune graphs that do not contain these substructures. GraphGrep[6] is a Path-Based indexing algorithm. Although GraphGrep can effectively pruning, due to the simple path, the structural information will lose during the search process[7]. It is sensible to use structures instead of paths, but how to choose discriminative structures is still a controversial issue. The SQBC[8] algorithm uses the Clique as the discriminative structures to achieve effective pruning. However, in the offline phase, the algorithm needs to calculate a large number of clique codes, which brings redundant calculations.

No Feature-Based Algorithms: This type of algorithm does not rely on graphic features, but uses Vertex Signature and Edge Label Encoding to encode graphic information[9,10,11]. For example, NOVA[12] uses the neighborhood labels of the vertex as its own index. SPath[13] proposes a vector indexing method based on the n-step shortest path around the vertex. However, building index will put pressure on memory and cause increased space complexity.

In this paper, I propose an algorithm to accelerate subgraph query by using synchronous update, which effectively relieves the memory pressure and speeds up the query. In the offline phase, given the connection between the query graph and the target graph, I preprocessed the target graph effectively reducing the search space. In the query phase, I utilize synchronous update to realize the query and pruning simultaneously. In the meanwhile, the vertex neighbor information and the Clique feature are used to effectively validate the candidate set.

3 PROBLEM DEFINITION AND PRELIMINARIES

In this section, I first give some basic definitions of our problem. Table 1 lists commonly used symbols throughout the paper.

Notations	Definition and Description
MTG	Minimum target graph after offline phase
GVS	the vertex set in the targe large graph
QVS	the vertex set in the query graph
GES	the edge set in the targe large graph
QES	the edge set in the query graph
GLS	the label set in the targe large graph
QLS	the label set in the query graph
L(v)	label of vertex v
L(e)	label of edge e
INNB(v)	In-Neighbors of vertex v
OUTNB(v)	Out-Neighbors of vertex v
INND(v)	In-Neighbors' degree of vertex v
OUTND(v)	Out-Neighbors' degree of vertex v
freq(L_i)	Frequency of label L_i in the graph
MRS	Matching result set
TCS	target condidate sutset

Definition 1 (Target graph). A large target network graph can be defined as a labeled directed graph $G = (V, E, L)$ with a vertices set V, an edge set E, and a label function L, where each vertex $u \in V$ represents an entity in the network, each edge $e \in E$ represent the relationship between two entities, and L is assigned a label for each each vertex. In fact, each label can represent attributes of an entity, such as name, value, etc.

In the example graph of Figure 1 (a), the number is used to represent a unique vertex's ID, and the alphabet is a vertex's label. Note that two different vertices in the graph can have the same label.

Definition 2 (Query graph): Query graph $Q = (V_Q, E_Q, L_Q)$ is a labeled directed graph with a set of query nodes V_Q, a set of query edges E_Q, and a label function L_Q, which is assigned a label for each query node $v \in V_Q$.

Problem Statement: Given a target graph G and a query graph Q, the problem in this paper is to find all matches of Q to G.

Figure 1 (b) shows the subgraph match.

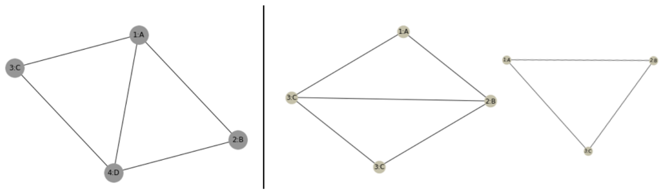

Figure 1: (a) an example graph (b) an example for subgraph match

Then, I will give some basic definitions about the graphs.

Definition 1 (Graph Isomorphism[14]): Graph G_1 is isomorphic to G_2 if there exists a bijection ϕ such that for every vertex $v \in V_1$, $\phi(v) \in V_2$ and $attr(v) = attr(\phi(v))$, and for every edge $e = (v_1, v_2) \in E_1$, $\phi(e) = (\phi(v_1), \phi(v_2)) \in E_2$, and $attr(e) = attr(\phi(e))$

Definition 2 (Subgraph Isomorphism[14]): Given two graphs Q and G, Q is a subgraph isomorphic to G if and only if Q is isomorphic to at least one subgraph G_0 of G, and G_0 is called a match of Q in G.

Definition 3 (clique[14]): A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent

4 SUBGRAPH SEARCH

I divide the algorithm into offline and online phases. The offline phase mainly processes the target large graph G to reduce the search space, and the online phase performs subgraph search. To better illustrate the algorithm, I will use Figure 2 as an example, of which Figure 2 (a) is the query graph, and Figure 2 (b) is the target large graph.

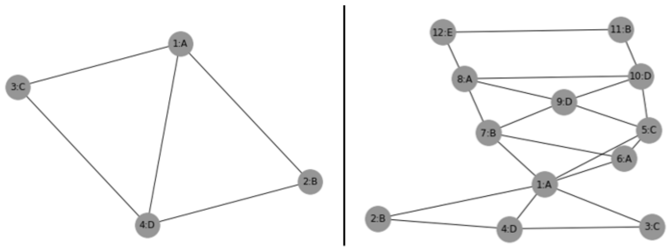

Figure 2: (a) the query graph. (b) the target large graph

4.1 Offline Phases

In the offline phase, the vertex set and edge set of the target large graph are filtered. The filtering principle is that the edges corresponding to the label in the query graph are retained, otherwise the edges are deleted. The pseudo code is shown below (Figure 3),

```
1 Algorithm 1: filter the target large graph
1:  input: the targe graph $G$, the query graph $Q$
2:  output: MTG
3:  for each $v \in GVS$ do
4:      if $L(v) \notin QLS$ then
5:          remove $v$ for $G$
6:      end if
7:  end for
8:  for each $e \in GES$ do
9:      if $L(e) \notin QES$ then
10:         remove $e$ for $G$
11:     end if
12: end for
13: return $G$
```

Figure 3: Filter The Target Large Graph

After filtering, it can be clearly seen that in the Figure 4, the vertices and edges of the target graph G are significantly reduced, effectively curtailing the search space by 77.8%.

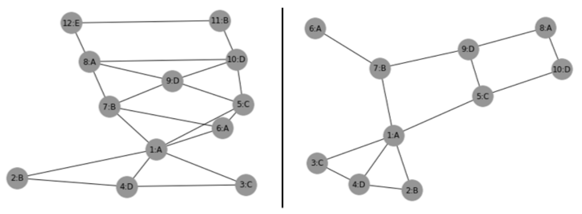

Figure 4: (a) the target graph before filter. (b) the target graph after filter

4.2 Online Phases

4.2.1 Candidate sets Construction

In the process of constructing the candidate set, the following filter criteria can be adopted.

(1) The label of the candidate vertex on the target lager graph G is the same as the label of the corresponding vertex on the query graph Q. That is,

$$L_G(v) = L_Q(v)$$

(2) The in-degree and out-degree of the candidate vertex on the target lager graph G must be greater than the in-degree and out-degree of the corresponding vertex on the query graph Q. That is,

$$INND_G(v) >= INND_Q(v)$$

$$OUTND_G(v) >= OUTND_Q(v)$$

(3) The number of neighbor labels of the candidate vertex on the target lager graph G is greater than the number of neighbor labels of the corresponding vertex on the query graph Q.[12]

(4) The number of vertices in the maximum clique containing the candidate vertex on the target lager graph G is greater than the number of vertices in the maximum clique containing the corresponding vertex on the query graph Q.[8]

4.2.2 Synchronous Update

According to the formulated filtering criteria, the candidate set of each vertex is obtained, and then the target candidate set is constructed through combination. In order to achieve the final matching result, each element in the target candidate set needs to be verified, which will bring a redundant calculation process. This is avoided with synchronous updates. Synchronous updating implements filtering and validation simultaneously, thus greatly reducing time complexity. Specific examples are as follows,

Assume that the current candidate set for each vertex is shown in Figure 5, where vertices on the query graph Q are represented by v_i and vertices on the target large graph G are represented by u_i, then the number of elements in the combined target candidate set is 8.

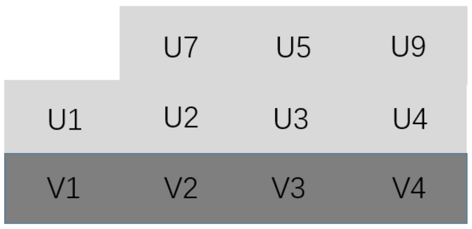

Figure 5: The Candidate Set of Each Vertex

When the synchronous update method is adopted, the candidate subsets will be gradually formed. When adding the next candidate vertex on the target large graph G, first check whether the vertices in the candidate subsets satisfy,

(1) the corresponding edges exist.

(2) the labels of the edges on the candidate subsets are the same as the labels of the corresponding edges on the query graph Q.

The synchronous update for the example is shown in Figure 6. Note that, the $Turbo_{iso}$ algorithm[15] is used to determine the order of candidate subset formation. The specific formula is as follows,

$$Rank(v) = \frac{freq(L_G(v))}{INND_Q(v) + INND_G(v)}$$

$Turbo_{iso}$ is more inclined to select vertices with a lower Rank value. This algorithm can effectively speed up the screening process.

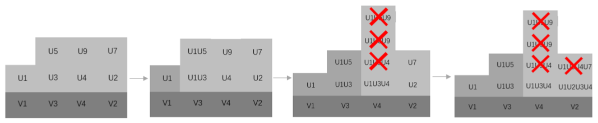

Figure 6: Synchronous Update Process

Synchronous update algorithm pseudo code is as follows (Figure 7),

1 Algorithm 2: Synchronous update for Subgraph Search

```
1: input: the targe graph $G$, the query graph $Q$
2: output: MRS
3: for each $v \in GVS$ do
4:     for each $u \in QVS$ do
5:         if $L_G(v) = L_Q(u)$ and $INND_G(v) >= INND_Q(u)$ and $OUTND_G(v) >= OUTND_Q(u)$ and v satifies the NOVA and the SQBC condition then
6:             put v into the condidate set of u
7:         end if
8:     end for
9: end for
10: while $i < len(QVS)$ do
11:     combine the condidate set of i-th vertex in QVS with the condidate set of the first i-1 vertices in QVS
12:     check the TCS
13:     i = i+1
14: end while
15: return TCS
```

Figure 7: Synchronous Update For Subgraph Search

5 EXPERIMENT

In this section, I evaluate our algorithm (represented as SSSU) on a synthetic dataset. I compare the SSSU with the optimization method that does not take synchronous updates (represented as UN-SSSU) in this paper. SSSU and UN-SSSU has been implemented in Python. All experiments are performed on a laptop with a CPU frequency of 2.6GHz and a memory capacity of 32GB under Windows.

5.1 Datasets

In order to confirm the effectiveness and efficiency of my algorithm, while taking into account the limited memory of the laptop, I adopt EM model to generate the dataset in the experiments.

The ER model defines a random graph as N vertices connected by M edges. These vertices are randomly selected from $\frac{N(N-1)}{2}$ possible edges. In the experiment, I generated a large graph and a set of small graphs. The large graph consists of 1000 vertices and 7,000 edges. The size of the thumbnails varies from 2 to 5.

5.2 Experimental results

According to the datasets described above, I evaluated the method proposed in this paper, and analyzed offline performance and online query response time based on the size of the query graph.

The offline consumption time and query time of the above two algorithms, namely SSSU and UN-SSSU, are summarized in Figure 8. The abscissa indicates the size of the query graph, ranging from 2 to 5. The ordinate of Figure 8(a) is the offline consumption time, and the ordinate of Figure 8(b) is the query time. It can be seen from the figure that SSSU is less than the other algorithms in both the offline consumption time and the query time. There is no doubt that the performance of SSSU is better than UN-SSSU. In addition, the performance line of SSSU is almost horizontal, which shows that our algorithm has good scalability as the query graph size increases.

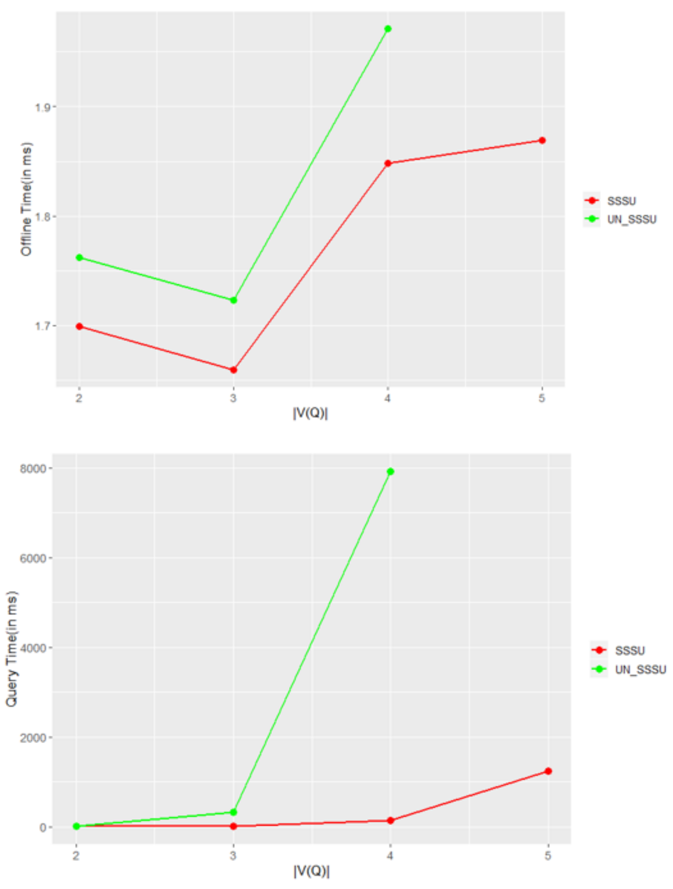

Figure 8: (a) The Offline Time (b) The Query Time

6 CONCLUSION

In this paper, I propose a new subgraph search algorithm on a large graph, called SSSU. It uses synchronous updates to speed up searching. It performs filtering and verification simultaneously, and combines optimization strategies such as SQBC and NOVA to effectively optimize the subgraph search process. It can be seen from the experiments that the performance of SSSU is significantly higher than the algorithms without synchronous updates. However, due to the limitations of the device, I hope to further verify the results on a larger data set in the future. In addition to the issues raised in this paper, research on approximate subgraph matching, subgraph queries on probabilistic graphs, and subgraph queries on frequently updated graphs are meaningful, interesting, and challenging. In future work, I will focus on these issues.

7 REFERENCE

[1]Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu.Graph pattern matching: from intractable to polynomial time. Proceedings of the VLDB Endowment, 3(1-2):264–275, 2010.

[2]Yahoo webscope. yahoo! altavista web page hyperlink connectivity graph.

[3]Ullmann, J.R. An Algorithm for Subgraph Isomorphism[J]. Journal of the ACM, 1976, 23(1):31-42.

[4]Cordella L P, Foggia P, Sansone C, et al. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(10):1367-1372.

[5]H. Shang, Y. Zhang, X. Lin, J.X. Yu, Taming verification hardness: an efficient algorithm for testing subgraph isomorphism, PVLDB 1 (1) (2008) 364–375.

[6]Giugno R , Shasha D . GraphGrep: A fast and universal method for querying graphs[C] Pattern Recognition, 2002. Proceedings. 16th International Conference on. IEEE, 2002.

[7]Yan X, Yu PS, Han J. Graph indexing based on discriminative frequent structure analysis[J]. ACM Transactions on Database Systems, 2005, 30(4):960-993.

[8]W. Zheng et al., SQBC: An efficient subgraph matching method over large and dense graphs, Inform. Sci.(2013)

[9]He H, Singh A K. Closure-Tree: An Index Structure for Graph Queries[C] Data Engineering, 2006. ICDE ’06. Proceedings of the 22nd International Conference on. IEEE, 2006. 30(4):960-993.

[10]Zhang N, Ilyas IF, Aboulnaga A. FIX:feature-based indexing technique for XML documents[C] International Conference on Very Large Data Bases. VLDB Endowment, 2006.

[11]Zou L, Chen L, Yu J X, et al. A novel spectral coding in a large graph database[C] EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008, Proceedings. DBLP, 2008.

[12]K. Zhu, Y. Zhang, X. Lin, G. Zhu, W. Wang, Nova: a novel and efficient framework for finding subgraph isomorphism mappings in large graphs, in: DASFAA (1), 2010, pp. 140–154.

[13]P. Zhao, J. Han, On graph query optimization in large networks, PVLDB 3 (1) (2010) 340–351.

[14]W. Zheng, L. Zou, D. Zhao, Answering subgraph queries over large graphs, in: WAIM, 2011, pp. 390–402.

[15]Han W S, Lee J, Lee J H. TurboISO: Towards ultrafast and robust subgraph isomorphism search in large graph databases[C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013.