An Efficient Sub-graph Isomorphism Algorithm Based on Breadth First Strategy

Tian Weixin
College of Computer and Information Technology, China Three Gorges University, Yichang, China
t_wxin@hotmail.com

Abstract - Sub-graph isomorphism is an important elemental issue in graph theory. This paper aimed to cope with the fall in performance that the current algorithms meet when the edges of the source graph grow up, and proposed an algorithm based on breadth first strategy. The algorithm sorts the vertices of the two graphs by the degree of out-edge and in-edge and adds all the vertices to the feasible pair according to the connection relations of the current vertex. The onging solution will be discarded and turn to next when any conflicts occur. The experiment shows that it has the better performance than current algorithm when the edges increase.

Index Terms - sub-graph isomorphism, BFS, graph matching, vertices pair

1. Introduction

Graph is one of the most expressive structures in the information science. Applications in many sorts of domains apply graph to organize their data to take the advantage of its flexibility and expressive capability. The graph was used to represent DNA structures in bioscience, to represent molecule structures in chemistry, to represent superficial objects in image process, to represent social networks in virtual community and so on. In the context of graph research, the isomorphism between two graphs means a one-to-one map from each of the vertices of the one graph to that of the other preserving the same topological connections. Sub-graph isomorphism is an important research topic in the domain with the purpose of finding sub-graphs of a source graph which are isomorphic to a given pattern graph.

The techniques for sub-graph isomorphism can be roughly generalized into two classes[1]. One is the determination algorithm, also called exact algorithm, and the other is inexact algorithm. The exact algorithm, seeking for consistence in the vertices as well as the edges while matching, returns true only when every vertices of the pattern graph have their counterparts in the source graph and each edges between two vertices in the pattern graph have corresponding edges in the source graph regarding the match pair of the vertices. Many literatures fall into this category. For example backtracking based method[2], state space based method[3], group theory based method[4], decision trees[5,6] and etc. Inexact algorithm, with the purpose of reducing computing complexity and gaining speed, makes its sense on the ground of the fact that there will be noise while representing the objects or relations in the real world or the occasion that no exact result is required. This method usually defines a matching cost to measure the difference of the two graphs. The matching process is thus transformed into a procedure of minimizing the matching cost.

Since the exact sub-graph isomorphism on general graphs is inherently a NP problem, many research efforts have been devoted to some restricted classes of graphs such as trees, two-connected outer-planar graphs, and two-connected series-parallel graphs and graphs of bounded tree-width[8]. There are also many methods have been proposed to decrease the calculating expense of the NP sub-graph isomorphism algorithms. However, they all fixed their eyes on the influence by single vertex. In fact, the pair of vertices has more effect in the procedure of seeking the isomorphism because criteria based on this can reduce more than one path once. In this paper, we proposed an algorithm to make advantage of the criteria based on the multi-pair of vertices. Every checking round, we check and add all the edge-end vertices of the current vertex as part of match. The vertex will be skipped when fail to be verified. Experiment shows that it is an efficient strategy when the edges of the source graph grow up.

The rest of the paper is organized as follow. In section 2 we will present some researches related to our research. In section 3 we will present an basic algorithm and an improved one; The section 4 is the experiment done on the graph database. Section 5 we will conclude the paper and also give some possible future work.

2. Related Work

The exact sub-graph isomorphism composes two main parts. One part is the strategy of searching, and the other part is the fitness or pruning function. For a pattern graph with m vertices and a source graph with *n* vertices*(m<n)* ,there will be *P(m,n)* possible matches. So it's necessary to organize the solution space and search it efficiently and orderly. Ullmann's algorithm uses the permutation trees to present the solutions. Every nodes in the tree presents a match between a vertex in the pattern graph and a vertex in the source graph, and the path from the root to the leaf-node means a possible solution[2]. Cordella's algorithm uses state space to organize the solution space, and utilizes the relations of inclusion among the states to sequence the solutions[3]. Many researches adopted those two techniques as their searching strategies. To the second issue, Ullmann etc. defined constraints to get rid of the forward branches before hand, while Cordella etc define and fitness function *F(n,m,v)* to do the same thing. Both techniques are based on the degree of vertices. However, the cordella's is in a deeper lever. It makes use of the relations sufficiently between the vertices and edges of the two graphs. In a more theoretical aspect, that how many edges a vertex has and the checking order of the number of

edges are crucial to the performance. Fedor V. etc. studied the influence of the tree-width and path-width to an isomorphism algorithm, and had showed that if the tree-width of pattern graph is at most t, then there is an algorithm for the sub-graph isomorphism running in a certain less time[12]. And even more, when the edges of the pattern graph is proportion to the vertices on the function of $O(k\log n)$, there is more simple algorithm to find whether this pattern graph is isomorphic to a source graph with bounded tree-width. According to this, MohammadTaghi etc. presented a polynomial-time sub-graph isomorphism algorithm under these restrictions[8].

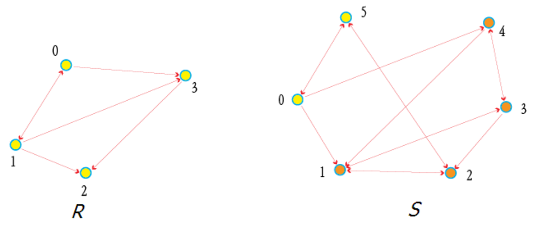
Apart from those researches exploring the relation between vertex and edge, there are also other attempts to lift the efficiency of the sub-graph isomorphism algorithm. One of the threads is to seek for parallel solution. M. Patwary etc. views the graph matching algorithm as a sparse matrix algorithm, and presented a parallel version by making use of sparse matrix partitioning methodology[14]. [7]describes a sub-graph type-isomorphism matching algorithm running on MapReduce platform. Another thread of researches is to apply some transformation algorithms. Minsu Cho propose a novel progressive framework which combines probabilistic progression of graphs with matching of graphs, which efficiently re-estimates in a Bayesian manner the most plausible target graphs based on the current matching result, and guarantees to boost the matching objective at the subsequent graph matching[15]. Jaeun Choi etc. proposed a multi-objective genetic algorithm for the sub-graph isomorphism problem. It designed a new fitness function which not only considers directly-visible characteristics of current solutions, but also considers the potential for being an optimal solution[16]. Those can be a potential weapon to address the sub-graph isomorphism, while the efficiency still left for verifying when facing the large scale graphs.

When setting out to attack the sub-graph isomorphism problem, the first issue will be faced is to choose the type of graphs. Some literatures chose to study the labeled graphs[10,13]. As the more recently research shows that compared with unlabeled graph, the algorithm on labeled graph will gain more efficiency for the constraint effect of the labels. But in the research perspective, this effect will be a distraction to the sub-graph isomorphism study[11]. In this paper we focus our devotion to the pure unlabeled graphs.

3. Algorithm

3.1 Preliminary

Graph can be mainly divided into undirected graph and directed graph according to their edge type. In this paper, we addressed the directed graph, which is defined as $G=(V,E)$ .Given two graphs $A(V_a,E_a)$ and $B(V_b,E_b)$ $(|V_a|\le|V_b|)$, the exact sub-isomorphism between *A* and *B*, is to find out all the sets of vertex pairs $\{(v_i,v_j)\mid v_i\in V_a, v_j\in V_b\}$ each set is called an isomorphism solution, in which every vertex in graph *A* has an one-to-one mapped vertex in graph *B*. Fig.1 shows two graphs and two of isomorphism solutions.



*the pair set $\{(0,4),(1,3),(2,2),(3,1)\}$ is an isomorphism

Figure 1 An Example of Sub-graph Isomorphism of R and S

To get the isomorphism solutions, it is nesessary to match all the possible vertex pairs between the two graphs and its corresponding edges. For the two graphs R,S with number of vertex $N_r,N_s(N_r\le N_s)$ respected, there will be $A(N_s,N_r)$ different solutions. Those solutions can be organized as a tree. Each solution then can be represented as a path from the root to the leaf of the tree. the node in the path denotes a match of vertex pair (v_i,v_j) $v_i\in V_a, v_j\in V_b$. Thus an unqualified pair is more close to the root, it will eliminated more unqualified solutions when it is checked. So sorting the pairs sounds to be of help to the match performance. One natural way to do the match is to sort the pairs first and then check every pairs systematically. In each round of the check, impose the constraint conditions on it to abandon the unqualified pairs. It is doubtless that all the solutions will be returned by this way. However it is not bound to have a nice performance because not all the pairs deserve a checking round. It's not a trivial cost when the number of graphs becomes large. Experiment shows that generating the pairs along the edges between the vertices is a relative better strategy than those systematical ones.

3.2 The Basic Breadth First Algorithm

```
;R[],S[] is the vertex array of graph R,S respected, sorted by the degrees of the out-Edge and in-Edge, vertex Vi's out-Edge or in-Edge degree are denoted as Dout(Vi) or Din(Vi);
 For match between R[0] and each member of S[]
 $H=\varnothing$          ;H is a feasible set of pairs.
 If feasible(0,i) do
 Add (0,i) to feasible pairs set
 While(Exist non-visited node of SVex(H)) do
 ;$SVex(H)=\{i\mid(*,i)\in H, i\in Vertex(S)\}$
 Visited(i)=true
 While(getPairs(H)) do
 Add the vertex pairs(m,n) to H if feasible(m,n).
 If($|H|\ge N_r$) output the solution.
 End while
 End while
 End for
```

**The feasible(m,n) function was defined as Dout(m)<Dout(n) and (Din(m)<Din(n).*

**The getPairs(H) function fetch a pair(m,n) from H in turn and generated all the possible match pairs according to the in and out edges of m and n.*

The algorithm starts with a vertex pair that has the maximum edges, then extends the pairs set by adding the both ends of out edges of the vertex of pair after checking its feasibility. Each pair added into the set will serve as a start node to extend new pairs. If all the nodes in graph R have been taken into the pairs, a solution is obtained. Otherwise, there must be a node in graph R that cannot be matched in the rest vertex of graph S and the algorithm will change to another start node. The algorithm ends when all the possible start nodes have tried.

Because the vertex are sorted by the degree of out-edges and in-edges, the matching process can be ended as earlier as the time it first meets the unmatched pair.

We adopted the uncompressed adjacent matrix to store the graph. Fig.2 shows the structures of the two example graphs mentioned before.

0	1	0	1
1	0	1	1
0	0	0	0
0	0	1	0

0	1	0	0	1	1
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	0	1	0
0	1	0	1	0	0
1	0	1	0	0	0

Figure 2 Adjacent Matrices for R, S

3.3 The Improvement Breadth First Algorithm

This section, we present an improvement algorithm based on the former one to reduce the memory cost without much decrease in performance.

In a graph, for every pair of vertex, we can see four types of relations between the two vertices. They are no edge, one edge in forward direction, one edge in backward direction and two edges in bi-direction. In former algorithm, we record those information by putting down 0 or 1 in different place. In fact, to the algorithm presented ahead, it's only crucial to provide a way to visit the edges by index and the place as well as how to present the relation types is not very important. So we compress the adjacent matrix by only reserving the place for pairs that have the ascendant order for the two vertices. In the place where has two possible values of 0 or 1 before, now has four optional values of 0-3. We use 0 to present no edge, 1 to the forward direction edge, 2 to the backward direction and 3 to the bidirectional edge. The representation which applies to the example graphs come the following Fig. 3.

For example when we need to check whether there is a edge from v_i to v_j, in the previous algorithm, we just need to check whether the value of array *V[i][j]* is 1. Now in the current algorithm we do the checking like this: $If\,(i > j)$,then check whether the value of array *V[i][j]* is more than 2, otherwise, check whether the value of array *V[i][j]* is more than 1.The memory cost in this algorithm is about half that of the previous one.

3	0	1
	1	1
		2

1	0	0	1	3
	3	3	2	0
		2	0	3
			0	3
				0

Figure 3 Compressed Representation for R,S

4. Experiments

We do the experiment on the Graph Database realized by SIVALab of the University of Naples "Federico II". This database has been well applied as a benchmark in the graph isomorphism research community[17]. The graphs have been randomly generated according to six different generation models, each involving different possible parameter settings. As a result, 84 diverse kinds of graphs are contained in the database. Each type is represented by thousands of pairs of graphs for which an isomorphism or a graph(sub-graph) isomorphism relation holds, for a total of 143,600 graphs. There are mainly three parameters of the graphs in the databases. One is the number of vertex. There are 10 classes numbers in the database from 20 to 1000. The other is proportion that the sub-graph contains nodes in the full graph, the three of which are 0.2,04 and 0.6 respectively. The third is the average branches that the full graph possesses, which named as η.

We choose a subset of the graph database to perform the experiment. On each category of the graphs with the same parameters we run 10 times on different graphs and get the average performance. To avoid inconsistence caused by the computing platform, we normalize the cost time to some scales. For the initial algorithm we denote it as BBF, and for the improvement one, we denote it as IBF. We choose VF2 as the reference algorithm for it is the most popular algorithm nowadays and many other researches have chosen it as the reference algorithm. The rows of the tables are for different proportions that the sub-graph contains nodes in the full graph. The columns of the tables are the number of vertex in the full graphs.

We use three tables to put down the experiment results, for $\eta = 0.001, 0.005, 0.01$ respected. From the tables, the IBF is slower than the BBF almost in every situation. However, their time costs are in the same order of magnitudes. In some memory sensitive occasion, the IBF can be taken as an alternative algorithm for it diminishes the memory cost markedly. When the number of vertices is not very large, the VF2 performs better than BBF and IBF. But with the scale of the graph growing, the proposed algorithm gets better than VF2, especially the case of the algorithm BBF. And with number of vertices of the target graph getting close to that of the source graph, the superiority of the proposed algorithms looks like more stable. For maximum number of vertices is 1000 and the number of edges is also not very much in the graph database, further experiment on larger database is required to demonstrate the observation.

Table 1 Time Costs of Three Algorithms at $\eta = 0.001$

		20	60	80	100	200	400	600	800	1000
0.2	BBF	0.89	1.48	1.67	2.02	5.31	10.78	28.09	38.28	176.57
	IBF	1.17	1.74	2.03	2.87	5.98	15.43	44.96	51.08	217.09
	VF2	0.25	0.75	0.80	1.52	4.95	14.91	48.52	65.74	228.91
0.4	BBF	0.93	0.85	1.23	1.53	4.17	8.46	27.06	43.76	67.84
	IBF	1.30	1.13	1.64	1.75	4.86	10.32	31.44	52.31	78.69
	VF2	0.17	0.33	0.52	0.88	3.92	10.15	32.62	50.81	81.53
0.6	BBF	1.23	2.65	2.43	3.07	8.78	23.16	70.39	113.41	154.86
	IBF	1.54	3.17	2.84	3.65	10.02	26.88	72.50	120.82	179.88
	VF2	0.18	1.46	1.15	2.45	5.56	26.43	64.74	116.23	182.14

Table 2 Time Costs of Three Algorithms at $\eta = 0.005$

		20	60	80	100	200	400	600	800	1000
0.2	BBF	0.68	2.98	5.87	10.63	8.32	70.38	312.47	376.94	434.80
	IBF	1.05	3.79	6.33	11.52	9.20	90.19	393.51	426.13	482.17
	VF2	0.07	2.21	5.14	9.8	7.56	96.78	421.12	538.27	441.55
0.4	BBF	0.87	1.21	1.28	1.37	6.48	38.24	43.87	123.41	578.90
	IBF	1.07	1.52	1.49	1.53	6.97	42.91	54.30	158.35	633.19
	VF2	0.12	0.45	0.65	1.21	6.27	40.19	48.27	151.92	712.83
0.6	BBF	0.78	1.03	1.87	2.54	8.62	30.19	60.54	134.93	198.28
	IBF	0.93	1.36	2.01	2.79	9.31	34.85	66.21	153.21	217.33
	VF2	0.19	0.75	1.21	2.83	8.05	33.15	73.78	132.67	206.85

Table 3 Time Costs of Three Algorithms at $\eta = 0.01$

		20	60	80	100	200	400	600	800	1000
0.2	BBF	1.43	3.28	7.33	10.19	27.84	107.93	182.32	505.89	1078.91
	IBF	1.93	4.16	8.93	12.82	31.25	146.08	217.83	693.21	1245.87
	VF2	0.51	4.67	9.78	12.3	25.37	128.79	253.9	658.79	1164.34
0.4	BBF	0.81	1.08	1.36	1.92	18.78	76.59	192.83	453.22	1168.45
	IBF	1.10	1.48	1.87	2.43	23.92	96.08	267.35	526.62	1324.06
	VF2	0.13	0.38	0.71	1.32	20.83	92.42	281.67	578.26	1303.62
0.6	BBF	0.87	1.21	1.55	3.08	20.48	44.39	179.03	421.83	579.43
	IBF	1.02	1.47	1.73	3.54	25.03	49.77	198.32	460.65	643.21
	VF2	0.21	0.96	1.67	2.43	16.37	53.96	168.81	464.44	626.71

5. Conclusion

This paper proposed algorithms to seek the isomorphism from one smaller pattern graph to another larger source graph. We based our work on the attempt to explore the relations of the vertices and edges. Experiment on the graph database shows that the proposed algorithm performed better than the benchmark algorithm when the vertices and edges of the graphs grow up to some scale. As we pointed out that exploring relations of the vertices and edges and seeking in parallel are the two main trend for the sub-graph isomorphism problem, there are two aspect of work worth to do in the future. One is do some experiment on larger scale graphs to improve our algorithm, the other is to develop a parallel version of the proposed algorithm.

Reference

[1] Conte, D.,Foggia, F.,Sansone, C.&Vento, M. Thirty years of graph matching in pattern recognition. International Journal of Pattern Recognition and Artificial Intelligence 18(3), 2004, 265–298.

[2] J. R. Ullmann, An algorithm for subgraph isomorphism, J. Assoc. Comput. Mach. 23, 1976, 31-42.

[3] L. P. Cordella, P. Foggia, C. Sansone and M. Vento, An improved algorithm for matching large graphs, in Proc. 3rd IAPR-TC15 Workshop Graph-Based Represen- tations in Pattern Recognition, 2001, 149-159.

[4] B. D. McKay, Practical graph isomorphism, Congressus Numerantium 30, 1981, 45-87.

[5] C. Irniger and H. Bunke, Graph matching: filtering large databases of graphs using decision trees, in Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition, 2001, 239-249.

[6] M. Lazarescu, H. Bunke and S. Venkatesh, Graph matching: fast candidate elimination using machine learning techniques, in Proc. Joint IAPR Int. Workshops SSPR and SPR, 2000, 236-245.

[7] Todd Plantenga. Inexact subgraph isomorphism in MapReduce. J. Parallel Distrib. Comput. 73, 2013, 164–175.

[8] MohammadTaghi Hajiaghayi, Naomi Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. Journal of Computer and System Sciences 73, 2007, 755–768.

[9] V.Bonnici, R. Giugno, A. Pulvirent, D. Shash, A. Ferro.A subgraph isomorphism algorithm and its application to biochemical data. BMC Bioinformatics 2013, 14, 7-13.

[10] Cordella, L. P., Foggia, P., Sansone, C., Vento, M.: An Efficient Algorithm for the Inexact Matching of ARG Graphs Using a Contextual Transformational Model. In: Proc. 13th ICPR, vol. III, (1996).180-184

[11] Ullmann, J. R. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. ACM J. Exp. Algor. 15, 1, 2011, 1.6.

[12] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, B.V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. Journal of Computer and System Sciences 78, 2012, 698–706.

[13] Markus Weber, Marcus Liwicki, Andreas Dengel. Faster subgraph isomorphism detection by well-founded total order indexing. Pattern Recognition Letters 33, 2012, 2011-2019.

[14] M. M. A. Patwary, R. H. Bisseling, and F. Manne. Parallel greedy graph matching using an edge partitioning approach. In Proceedings of the fourth international workshop on Highlevel parallel programming and applications, HLPP '10, pages 45-54, New York, NY, USA, 2010. ACM.

[15] Minsu Cho and Kyoung Mu Lee. Progressive Graph Matching: Making a Move of Graphs via Probabilistic Voting, Proc. Computer Vision and Pattern Recognition (CVPR), 2012.

[16] Jaeun Choi, Yourim Yoon, Byung-Ro Moon. An Efficient Genetic Algorithm for Subgraph Isomorphism. Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference. GECCO '12. 361-368.

[17] P. Foggia, C. Sansone and M. Vento, A Database of Graphs for Isomorphism and Sub Graph Isomorphism Benchmarking, Proc. Third IAPR TC-15 Int', l Workshop Graph Based Representations, pp. 176-188, 2001.