Efficient Subgraph Mining in

Multi-Relational Networks

Pu-Jen Cheng (鄭卜壬)
Assistant Professor
Department of Computer Science and Information Engineering
National Taiwan University

R97922134 TingChu Lin (林庭竹)

Introduction & Motivation:

Subgraph searching problem [1] is an interesting research direction in the Web or many other social environments. However, there are many problems in the searching of graph. One of the most difficult problems is the cost of computation. When we apply more conditions in the searching algorithms, the computational time is longer and more costly due to there is a huge number of links and vertices in a graph. Thus, we want to apply the OpenMP [2] parallel method to speed up the computation of subgraph searching.

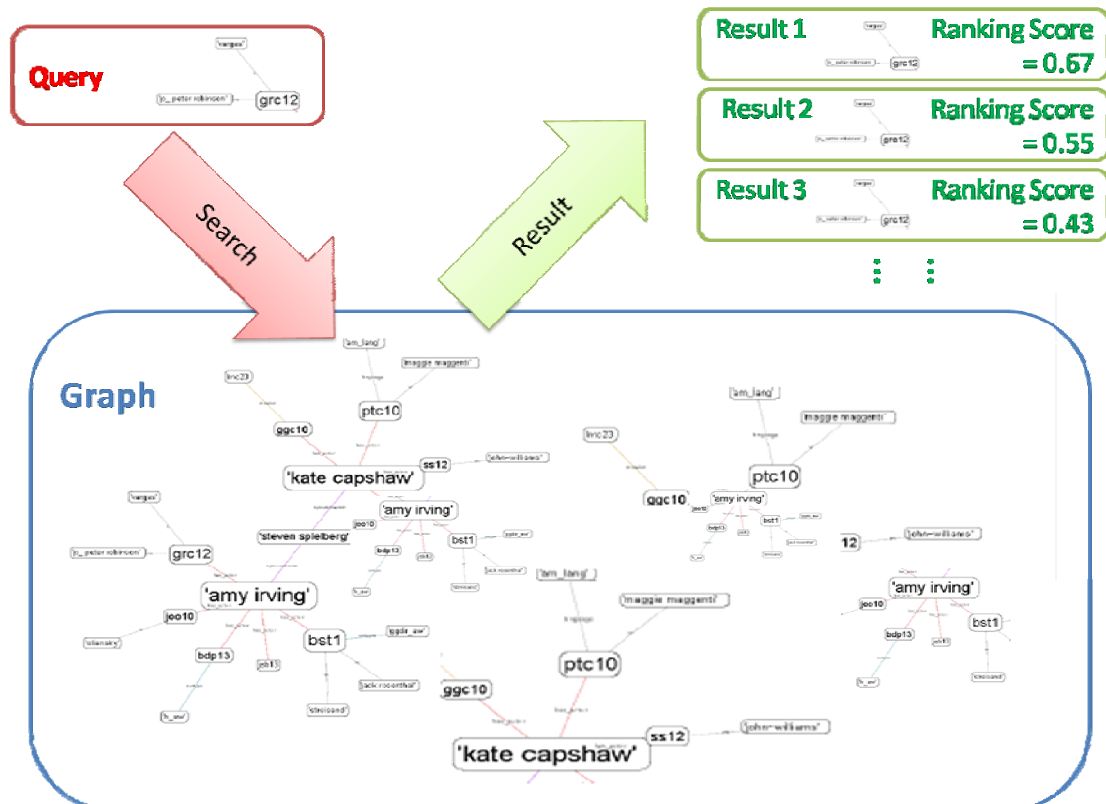


Figure 1. The graph representation of the DBLP dataset.

Methodology:

We use the OpenMP API [2] to implement the parallel subgraph search. Figure 2 shows the overview and a simple architecture of our problem. The detail processes of searching in social network are described in the following:

1. Given a query such as "A person who has published two papers", we can form a graph which is constructed by three nodes: one node with the type "author", and other two nodes with the type "paper". Also, there are two edges in this graph, that is, the links between the two paper nodes and the author nodes.
2. After the query graph constructed, we can search the query graph from the large social network about the DBLP data. The corpus (also it is a graph) is constructed before the querying performed. We can perform the searching task by several search methods for network search, such as Breadth-First-Search or Depth-First-Search. For the application, we use the BFS for most of the situations. For all the methods of searching in network, we must traverse the graph once and collect the information we need at every node. As a result, when the dataset is large, the network is also huge due to more nodes added, and the time for searching is extremely costly. For the sake of efficiency, we want to combine the parallel algorithm into searching while the task of traversing nodes is performed.
3. We initially use the PageRank [3] values for every subgraph to do the ranking of search results. When we found the subgraph corresponded with the graph of query, we sum up the PageRank values of all nodes in this subgraph as its score, and search for the next.
4. After all the nodes are traversed, there would be several corresponded subgraphs found. We then perform the ranking methods with respect to the scores (from PageRank value) of these graphs. Then we can further look up the hash table to find the full names of papers or authors for the top k results.

Also, here we have some notes in this project as we apply the steps above to the searching process:

1. The isomorphism of a subgraph. We might get many search results while there is a condition for isomorphism. When a subgraph is simple and well-structured, there might be several corresponding subgraph and the searching and ranking process would be slower. This is an important characteristic in graph theory and we must treat it in a careful way during the search phase.
2. The PageRank value would be a temporarily the ranking criteria. We might use some criterion from the graph theory, such as in-link, out-link, degree, or the link

score from HITS [4], etc. The goal of this project is to utilize the parallel techniques to speed up the time spent on searching a subgraph, thus we might apply the different criterion in the future work.

3. The query types are currently limited to the types of node. However, in a heterogeneous social network, the types of edges would be an important role as we added to the search process. In this way, query would be more complicated and difficult to process when the BFS algorithm applied. Consequently, we current use the type of nodes in this project to make the searching process simpler.

Figure 2. The architecture of subgraph searching problem.

Experiments

Dataset:

We adopt the DBLP dataset, wherein there exists an abundant of information about persons, papers, proceedings, and journals, and these data are modeled into graphs in which nodes and edges are defined. The node type includes "person", "paper", "proceedings" and "journal", whereas the edge type captures possible relationship existing between nodes and contains "author-of", "in-proceedings", "in-journal", and "cites". Detailed description is shown in the table below:

edge (relation)	node – node
author-of	person-paper
in-proceedings	paper-proceedings
in-journal	paper-journal
cites	paper-paper

Each node and edge is assigned an id number for identification. Based on the id number, some detailed description about the component (either node or edge) is also given in the dataset. For example, if the component is an "author-of" edge, the following information shall be provided according to the id number (1190061 in this example):

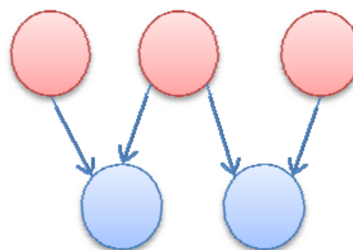
```
1190061 conference      ICALP
1190061 isbn    3-540-27580-0
1190061 publisher       Springer
1190061 series  Lecture Notes in Computer Science
1190061 title   Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings
1190061 volume  3580
1190061 year    2005
```

Finally, for each component (node or edge), a corresponding PageRank score shall be computed and stored for future inference. The network constructed by the DBLP dataset and the simple querying process can be represented as Figure 1.

Experimental Result:

For experiments of this work, we propose several queries for subgraph search, and further test the correctness and time speedup for the parallel methods.

a. **Query 1: Search a person with two different papers, which are co-authored with different two people.**



First of all, we try a query of which complexity is O(|G(Vperson)|^3). For this query,

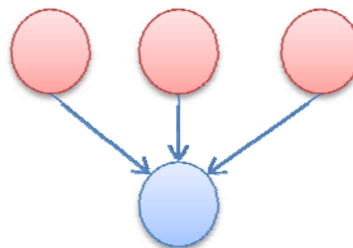
we need to compute who proposed more than two papers at first, and we marked this person as he could be possible subgraph. Second, we search all nodes of person type in corpse to find who the co-author with the marked person was. We need to check the permutation of every marked people with his “two” paper, too. We apply openMP API to parallel this query, and there are two different parallel directives in this experiment, one is marking critical section in result record variables, the other is using reduction to sum up total results.

# of threads	**Sequential**		**Critical**		**Reduction**	
	Time	Usage	Time	Usage	Time	Usage
4	14.81s	96.7%	11.4s	191.0%	11.17s	207.5%
8			12.13s	321.5%	11.57s	320.8%
16			11.04s	361.7%	10.75s	344.6%

Table 1. The time speedup table of query” Search a person with two different papers, which are co-authored with different two people”.

There is about 20% speedup that parallel program runs faster than sequential program. However, the increase of number of threads only increases the usage of CPU but no apparent improvement on time. The reason is we use a lot of privacy memory, and there are lots of inner loop of the parallel loop, so its cost of content switch overwhelms parallel speedup. In the same time, we could not see obvious improvement on critical or reduction.

b. Query 2: Search a paper co-worked by three people



We further try a simple query that described as “a paper is co-worked by three people”. For this query, the Breadth-First-Search (BFS) algorithm is again applied in the search process. First try to find a person with a paper, and find another two authors own such paper. Due to the links of the graph are all directed, the time complexity of sequential program is $O(n^3)$. We then apply the OpenMP API to implement the parallel program. The experimental results are shown in table 2 below. The number of threads is the

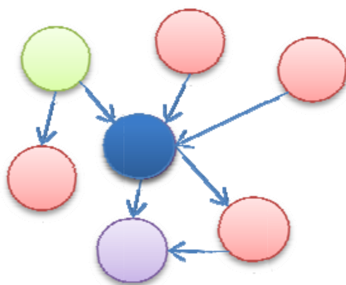
control factor, which are set to increase from 4 to 16.

# of threads	**Sequential**		**Parallel**	
	Time	Usage	Time	Usage
4	14.31s	99.9%	4.60s	297.8%
8			3.17s	533.4%
16			3.68s	456.7%

Table 2. The time speedup table of query ”Search a paper co-worked by three people”.

From the table, we could easily found that the parallel version might enhance the speedup up to 351% when 8 threads applied. However, the result is not much better when 16 threads applied (than 8 threads). We might consider it from the reason that when 16 threads are set and distributed to 8 CPU core to finish the task, the computer might switch the processes for finishing the tasks. Moreover, the usage rate of CPU when 16 threads assigned does not exceed the usage rate as 8 threads assigned, too. As a result, it is much better that to assign n threads when we have only n CPU cores in small number time spent of tasks.

c. Query 3: Search a person with two papers, and one of the papers is co-worked with other two people. Furthermore, the paper cites two other papers and one of these two cited papers cite another.



We then try another query for a more complicated subgraph. In this subgraph, there are seven nodes with two different types as the query 3 described. We use the Breadth-First-Search (BFS) algorithm again, and try to search the subgraph if there is such graph exists. For the rank result we use the summation of all of the PageRank value from the nodes again, and try to find out the time efficiency before and after parallel technique applied. Table 3 shows the result of query 3.

# of threads	Sequential		Parallel	
	Time	Usage	Time	Usage
4	14.48s	99.8%	4.31s	327.1%
8			3.85s	475.0%
16			4.09s	417.3%

Table 3. The time speedup table of query "Search a person with two papers, and one of the papers is co-worked with other two people. Furthermore, the paper cites two other papers and one of these two cited papers cite another.".

From the result, we can easily find that the parallel version does a very large improvement as the speedup can be reach to about 276%. We again see the result that using 16 threads is slightly worse than using 8 threads. In addition, from query 1 to query 3 we found that it is not always slow when the query size is much larger and is more complicate. The isomorphism of graph is introduced for explaining the result. From a simple query like query 1, we might get a huge amount of subgraphs from the large network and the permutation must be considered when the candidate subgraph added. Consequently, it got more time in query 1 but not so much cost for searching the query 3.

d. Query 4: Search for a given length x as the shortest path between two given

Repeat with [paper, proceedings]
until length equal to X.

- **Input (Type-H, Type-T, x, [Pattern])**
  - Type-H: Type of head node
  - Type-R: Type of tail node
  - X: length of the Path(shortest path)
  - [Pattern]: a string of path path's node type, repeat if necessary).

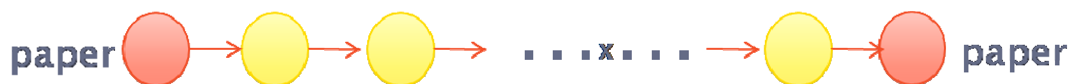
For this query, the Depth-First-Search (DFS) algorithm is applied in the search process. First try to find all nodes with a Type-H, then DFS algorithm is applied to each of them to find all nodes reachable, the parallel process start from here. In

addition, we do some check for back edges so the path form root is shortest, and the types of nodes in the path should follow the input pattern , in the end the, the type of tail node must be verified. Due to the links of the graph are all directed, the time complexity of sequential program is $O(n^3)$. We then apply the openmp API to implement the parallel program. The experimental results are shown in table below. The number of threads is the control factor, which are set to increase from 2 to 16.

# of threads	**Sequential**		**Parallel**	
	Time	Usage	Time	Usage
2	0.85s	93.0%	0.45s	184%
4			0.20s	296%
8			0.36s	432%
16			0.29s	429%

Table 4. The time speedup table of query "Search for a given length x as the shortest path between two given".

e. Query 5: Search for the subgraph has the longest shortest path between two given type of node.



Repeat with [paper, proceedings]

- **Input (Type-H, Type-T, [Pattern])**
  - Type-H: Type of head node
  - Type-R: Type of tail node
  - [Pattern]: a string of path path's node type, repeat if necessary).

For this query, the Depth-First-Search (DFS) algorithm is applied again in the search process. First try to find all nodes with a Type-H, then DFS algorithm is applied to each node of them to find all nodes reachable, the parallel process start from here. In addition, we do some check for no back edges so the path for root is shortest, and the types of nodes in the path should follow the input pattern, in the end the, the type of tail node must be verified. Openmp "critical" is used when update the longest length found latest. Due to the links of the graph are all directed, the time complexity of sequential program is $O(n^3)$. We then apply the openmp API to

implement the parallel program. The experimental results are shown in table below. The number of threads is the control factor, which are set to increase from 2 to 16.

# of threads	**Sequential**		**Parallel**	
	Time	Usage	Time	Usage
2	0.83s	91.0%	0.40s	178%
4			0.27s	291%
8			0.41s	462%
16			0.24s	457%

Table 5. The time speedup table of query "Search for the subgraph has the longest shortest path between two given type of node.".

f. Query 6: Calculate the average number of authors for one paper in different conferences or journals

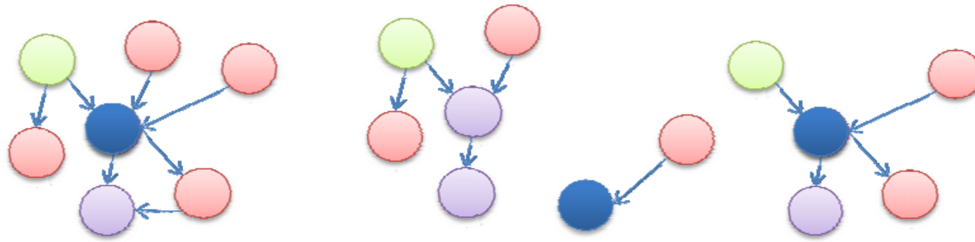
In this project, we might try some different statistic result for the large social network. A simple statistical way is shown here. The goal of query 6 is to find the average number of authors for one paper in different conferences or journals. We apply the Breadth-First-Search (BFS) again, and try to examine the time efficiency for sequential and parallel program.

# of threads	**Sequential**		**Parallel**		**Parallel Nowait**	
	Time	Usage	Time	Usage	Time	Usage
4	14.48s	99.8%	4.31s	327.1%	1.40s	153.5%
8			3.85s	475.0%	1.20s	246.6%
16			4.09s	417.3%	0.61s	167.2%

Table 6. The time speedup table of query "Calculate the average number of authors for one paper in different conferences or journals".

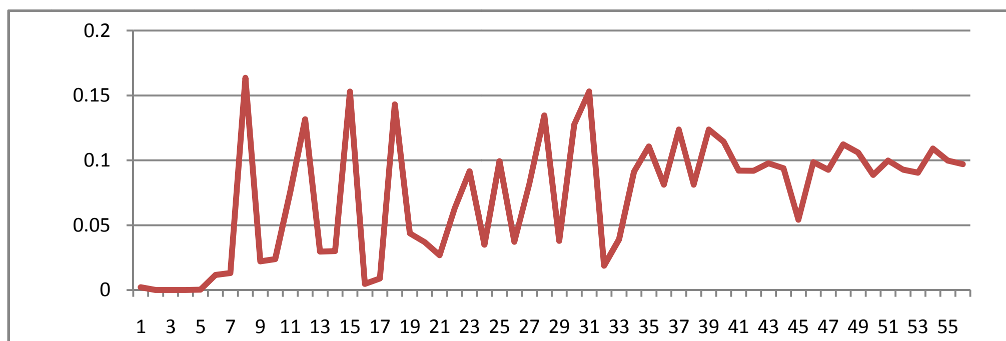
The result is shown in table 6. From the result, it is easy to see that the power of parallel programming reveals of all it. The best result, which appeared when 16 threads and "nowait" command are applied, tells us that we can improve the original program as 2273.77% of efficiency. The parallel programming could be powerful and enhance the speedup in a very huge effort if we use it carefully, thus solve the difficult and time-consuming problem in an efficient manner.

g. Query 7: Connected Component Analysis: Find statistical behaviors in each connected component and overall graph statistics.

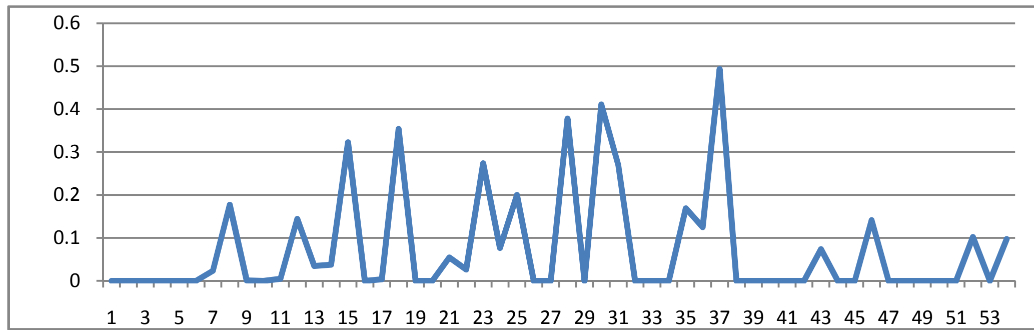


To give an overview of the entire DBLP dataset, we conduct experiments which aim to uncover all existing connected components. Experinetal results demonstrate that there are 13119 connected components, wherein the largest component contains 1163 nodes. This reveals that most of components consist of a small set of nodes; in other words, there are numerous research topics existing in the field of acedamics. Furthermore, we want to realize if there is some specific relation between the size of a component (e.g., how many nodes there are in the component) and the mean PageRank score in that component. Interestingly, the larger the size of a component, the more stable its mean PageRank score is. Similarly, we calculate standard deviation value in each component to realize the degree of agreements among a research group. Results show that standard deviation of some groups is pretty low, while some others have high deviation value. Generally speaking, however, the larger the size of a component, the more consistent their indivisual PageRank score is. Finally, we can see that the most common types of nodes in a component is either "paper" or "person" as expected, whereas there is nearly zero percent of "PhDthesis" in all size of components. We can see all analysis below:

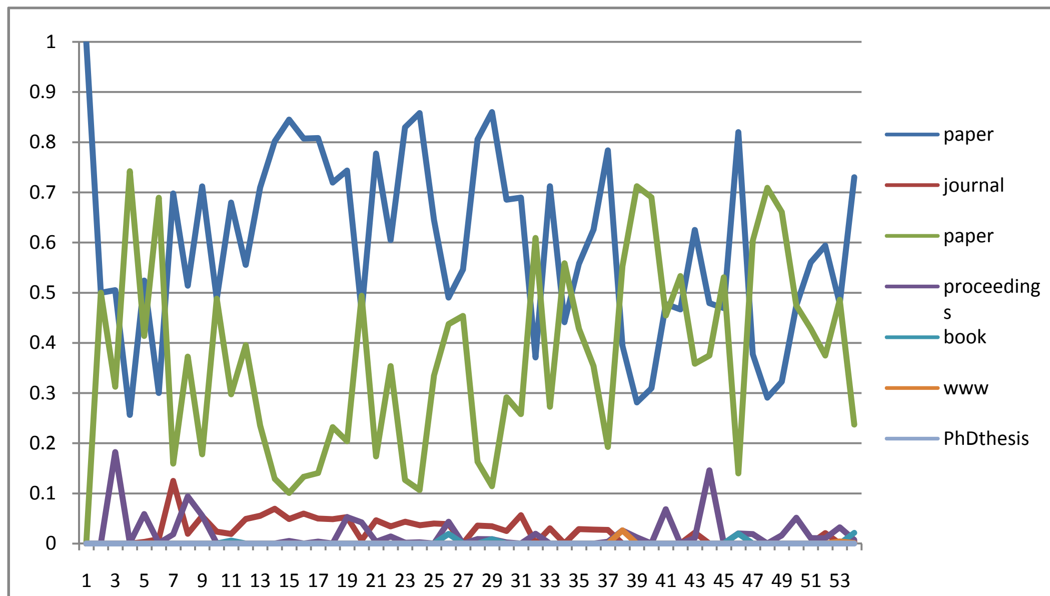
(1) Mean PageRank score vs. Component Size



(2) Standard Deviation vs. Component Size



(3) Type Distribution vs. Component Size



We implement the whole process by either sequential or parallel methods, and we can tell that parallel programs indeed boost the performance of connected components analysis, as shown below,

<table>
<tr><td rowspan="2"># of threads</td><td colspan="2">Sequential</td><td colspan="2">Parallel</td></tr>
<tr><td>Time</td><td>Usage</td><td>Time</td><td>Usage</td></tr>
<tr><td>4</td><td rowspan="3">7.22s</td><td rowspan="3">99.5%</td><td>3.00s</td><td>228.6%</td></tr>
<tr><td>8</td><td>3.13s</td><td>289.7%</td></tr>
<tr><td>16</td><td>2.85s</td><td>241.7%</td></tr>
</table>

Table 7. The time speedup table of Connected Component Analysis: Find statistical behaviors in each connected component, and overall statistics..

Conclusion & Future Work

In this project, we implement a program with the parallel technique and make a deeply look for the subgraph search problem in a efficient way. The contribution of this project is two-fold. From the efficient point of view, we combine the OpenMP parallel API into the project and implement it with parallel programming technique. Moreover, from several experiments, it shows us a very exciting and useful result when we make the subgraph search problem in a parallel way. The efficiency is substantially improved, thus make the problem more applicable and in a less costly way. As the parallel technique is widely used, there would be more solutions for the social network mining with the parallel perspective. Another part of contributions is that we make the application easier. For the graph mining's view, we propose the idea for efficiently mining the network for some interesting, special, or contradictory results, such as three papers which cited each other. The social network for mining could even be applied in different datasets like movie or other types of data. As a result, more social network problems can be solved efficiently, and more research topics in social network would be realized due to the cost of mining is substantially reduced.

We might apply different criterion for ranking subgraphs as discussed previously. Moreover, the edge's type might also be considered in the future work. Different data structure representations for graphs can also be used to solve the more difficult, complicated, and larger graph while the graph is constructed. We believe that with these improvements, we could solve more research and applicable problems in network mining, and make the mining process that in the research of social network much easier and more comfortable.

Reference

1. L. Zou, L. Chen, and Y.Lu. "Top-k subgraph matching query in a large graph". In Proceedings of the ACM Conference on Information and Knowledge Management (CIKM), pages. 139-146, 2007.
2. The OpenMP API Specification for Parallel Programming. http://openmp.org/wp/
3. L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web". In Technical report, Stanford University, 1998.
4. S. Chakrabarti , B. Dom , P. Raghavan , S. Rajagopalan , D. Gibson , and J. Kleinberg. "Automatic resource compilation by analyzing hyperlink structure and associated text". In Proceedings of the seventh international conference on World Wide Web 7, p.65-74, 1998.