Optimized Graph Search using Multi-Level Graph Clustering

Rahul Kala[1], Anupam Shukla[2], Ritu Tiwari[3]

[1, 2, 3] Department of Information Technology, Indian Institute of Information Technology and Management Gwalior, Gwalior, MP, INDIA
[1] rahulkalaiiitm@yahoo.co.in, [2] dranupamshukla@gmail.com, [3] rt_twr@yahoo.co.in

Citation: R. Kala, A. Shukla, R. Tiwari (2009) Optimized Graph Search using Multi Level Graph Clustering, *Proceedings of the Springer International Conference on Contemporary Computing*, Noida, India, pp 103-114.

Final Version At: http://link.springer.com/chapter/10.1007%2F978-3-642-03547-0_11?LI=true

Abstract. Graphs find a variety of use in numerous domains especially because of their capability to model common problems. The social networking graphs that are used for social networking analysis, a feature given by various social networking sites are an example of this. Graphs can also be visualized in the search engines to carry search operations and provide results. Various searching algorithms have been developed for searching in graphs. In this paper we propose that the entire network graph be clustered. The larger graphs are clustered to make smaller graphs. These smaller graphs can again be clustered to further reduce the size of graph. The search is performed on the smallest graph to identify the general path, which may be further build up to actual nodes by working on the individual clusters involved. Since many searches are carried out on the same graph, clustering may be done once and the data may be used for multiple searches over the time. If the graph changes considerably, only then we may re-cluster the graph.

Keywords: Clustering, Searching, Graph theory, Social Networking Analysis, Web Search Results

1 Introduction

Graphs are defined as a collection of vertices and edges or G(V,E). In the past years a lot of research in this field has led to many algorithms that are highly efficient. Searching in a graph refers to the algorithm of finding out a goal node, starting from a source node. A key emphasis is given on the length of the path from source to goal and the time taken to reach the goal node. For a good algorithm, the

length of the path traversed as well as the time taken should be minimal. Searching is one of the most trivial operations carried out in graphs. Various search algorithms exist like Breadth First Search, Depth First Search, Best First Search, A* Algorithm etc.

Clustering in a graph refers to the grouping of closer nodes to form one cluster. Likewise by constant grouping, various clusters of considerable sizes may be made. Clustering is of ample of use in data analysis in various domains like biomedical, pattern recognition, etc. Various algorithms exist for the clustering of the graph. Some of the prominent ones are C-means fuzzy clustering, k-means clustering, hierarchical clustering.etc.

Graphs are being used to model various problems. They have been extensively used to solve many day-to-day problems and find optimal solutions by various algorithms. One of the major applications is in the formation of social network graphs. This is especially motivated by their ample use at the social networking sites. These sites offer each individual to open their account. He may link to various other people by adding them as friends and build his social network. We consider each such user as a vertex of graph. If a person x adds a person y as a friend, we represent it as an edge between x and y. Hence this forms a massive graph of the order of millions of edges. Here we assume that a person links to only those people whom he knows reasonably well. In the absence of this assumption, the number of edges would become very high and the performance of Breadth First Search may exceed all known algorithms.

Another application of the algorithm is the search performed by the search engines. The search engines crawls the web to find results that match the given keyword. A typical search engine ranks the web pages on the basis of their closeness to the entered search query. This may give various kinds of pages. We may also cluster the search results. Very similar web pages may be put in one cluster. E.g. the search ‘sun’ might refer to sun Microsystems, solar system, sun java etc. These are the various classes the search result may be clustered in. We may hence represent the entire search result as a graph with the edges between any two pages depending upon their closeness to each other. Hence we would be able to form clusters. Initial results may show pages from cluster that is closer to the search query. As the user clicks some of the pages, we would get to know his interests. Then we may show more results from his preferred cluster. Some results may also be shown from the nearby clusters.

Whichever problem we take, we find that many times a series of search operations are called quite frequently, without considerable change in the graph. Every time the search algorithm tries to reach the goal node starting from the source node. In this paper we propose the graph to be clustered once at the starting. The entire graph is clustered to form clusters of some size. These clusters are replaced by new vertices. The edges are placed from this new vertex to any other vertex, depending whether it was connected in any manner to that vertex.

We cluster the graph in a multi layer method. Once the graph is clustered, to give us a smaller graph, we apply the clustering algorithm again to give us a still smaller graph. This process is repeated multiple times unless it is not possible to further cluster the graph (the individual cluster cannot be larger than the desired threshold size).

Instead of applying the searching to the entire graph, we apply it onto the clustered graph, to find the appropriate path. Once we know this, we iterate down level-by-level to every cluster. In these steps, we further come to the actual nodes rather than the ones generated while clustering. In the end we reach the first level. At this level the entire solution is a collection of points in the actual graph given as input.

The clustering may be done once and may not be repeated every time. Even if we wish to add nodes or modify them, we may easily do the same in the various levels of clusters formed. This would ensure that we do not have to re-cluster the graph for a long time ahead.

2 Related Work

With the ever increasing applications of graph and graph theory, it is evident that a lot of research is being carried out in this field. We find various algorithms being developed and various optimizations being carried out in order to improve the searching technique [2, 5, 10, 15]. The traditional searching involves algorithms like Breadth First Search, Depth First search, Heuristic Search etc.

Clustering is another rapidly developing area. In the past few years a lot of work has been done to improve the clustering algorithms and applying them to the graphs [1, 3, 8]. The metrics for the performance evaluation of graph clustering is another major area of research [6, 11]. We find various clustering algorithms like k-Means Clustering, C-Means Fuzzy Clustering, etc. that are being widely used and improved.

Social Networking has got a huge importance because of the ever increase in the number of users [13]. Various kinds of work are being done for their analysis and developing searching techniques. Similarly a lot of study is going on in the field of web searches and website navigation [4, 7, 9]. People are trying to cluster the results using various forms of optimizations [14]. There are works going on to build optimized graphs for the searching of results through keywords. Though this paper is not only limited to the these networks, but we will make a special mention as these networks are the motivation and provide the basic design of the search algorithm.

It may be noted that we assume here that the vertices of the graph are connected to many, but limited number of vertices. If we get vertices that are connected to most of the vertices, the total length of any path from source to destination would be very small. In such a situation, the Breadth First Search would perform very well. In the case of social networking sites, we assume that a person is linked to the other person, only in case he knows the other person relatively well. The absence of this case has lead researchers to conclude social networking graphs as low width graphs.

In this paper, rather than using an already built clustering algorithm, we build a separate algorithm. The reason for this is that we do not need precise clustering, where clusters need to be neatly well apart. We know the kind of application areas in which these algorithms would be put into. Hence we can get a lot of facts to develop faster clustering algorithms that take a lot less time.

The search algorithm is a simple breadth first search, but it is applied on the clustered graphs. This reduces the size of the problem and hence gives faster results.

We need to apply the breadth first search at every level of clustering. When we finish, we get the final path that is almost the most optimal path.

3 Algorithm

The whole process of this algorithm is divided into 2 parts. First we cluster the graph. In the second part, we use the multi level clustered graph and search for the goal node starting from the source vertex. We discuss both of these in the next sections.

3.1 Graph Clustering

The first step is to form clusters of the given graph. While forming clusters, we do not lay stress on the quality of the clusters produced, as is the case with many other clustering algorithms. Rather stress is given on the speed of the algorithm. We also have no initial idea as to how many clusters would be formed. This further urges the need of a new clustering algorithm. We discuss clustering in next sections. First we have a look at the multi layer concept and then we present the way we form clusters.

Multi Layer Clustering: Here we form multi layered clusters. The given graph is clustered to bring together vertices that lie close to each other. Each of the clusters is replaced by a single vertex. Hence after clustering we get a graph that is smaller in size as compared to the initial graph. Many vertices are clustered and replaced by one vertex. In the graph that comes out, we again apply the same algorithm. This process of applying algorithm goes on until it is not possible to cluster the graphs up any further, subjected to a maximum of A times. The general procedure of the multi layered clustering is given below. The algorithm is also summarized in figure 1.

MultiLayeredClustering
Step1: g ← original graph
Step2: for i = 1 to A
Step3: g ← makeCluster(g)
Step4: if there is no change in g
Step 5: break
Step 6: add g to graphs

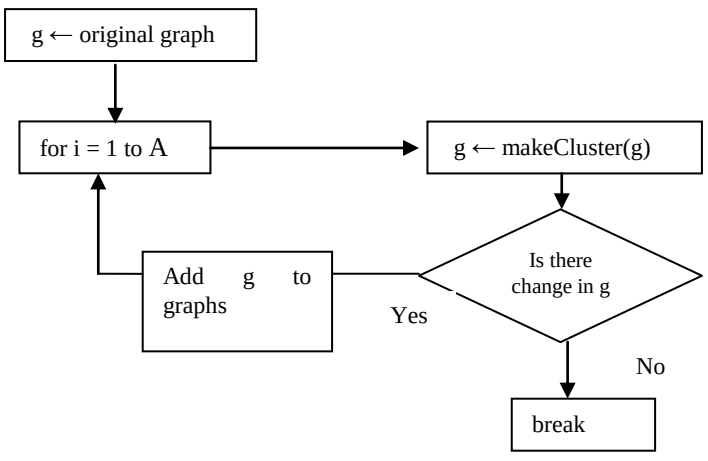


Fig 1. Algorithm of Multi Layer Clustering

The entire concept of multi-layer clustering can be easily understood with the help of the special case of social network graphs. At the lowest level each user is a vertex. We know that all users who belong to same department in an institute would be connected to each other very well. Hence this may be taken as a cluster separately. Like this we would be having many separate clusters each belonging to some particular department.

We also know that in the resultant graph, all departments of the institute would be strongly connected to each other. Hence we form clusters of every institute. Hence at the second level, graph represents various institutes as vertices. Likewise at the third level, clustering may be in form of state, in fourth level by country. In any general graph, clustering in this manner will reduce the graph size.

Making Clusters: Graph is given as an input to the clustering algorithm. The algorithm divides it into clusters of sufficient size. After we have finished forming clusters one after the other, some vertices are left. These vertices do not belong to any of the clusters. Each cluster is deleted from the graph. Here we remove all the edges and vertices that lie within this cluster. This is then replaced by a single new vertex that represents the entire cluster. This new vertex is connected by edges to all the vertices in the new graph that were connected to any of the vertex in the cluster in the original graph. E.g. figure 2 shows a clustered graph. The graph formed after the vertices and the edges of the new graph have been found out is given in figure 3.

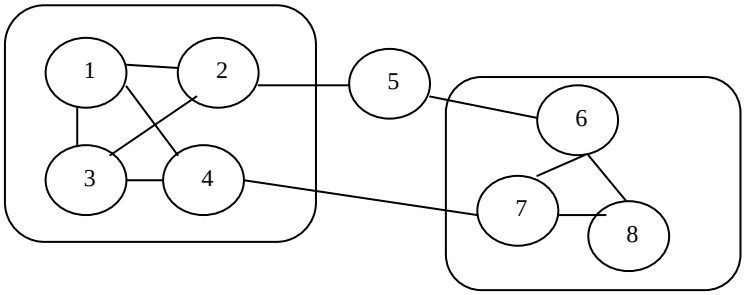


Fig. 2. Original graph

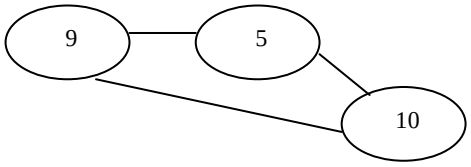


Fig. 3. Graph after clustering

The algorithm for making clusters is as given below.

Make Clusters()

Step1: While more clusters are possible
Step2: c ← getNextCluster()
Step3: for each vertex v in c
Step4: Delete v from graph and delete all edges from/to it
Step5: Add a new unique vertex v_2
Step6: Add edges from/to v_2 that were deleted from the graph
Step7: Add information of cluster v_2 to set of clusters in the particular level

We need to keep a record of all the vertices a cluster contains. Every cluster has a unique identity that is the identity of the vertex that substitutes the cluster. We register the cluster by mentioning all the vertices it contains. This information would be needed later in searching process.

For each cluster, we also register another parameter. This is the star vertex of the cluster. Every cluster has a star vertex. This is the vertex that is centrally located in the cluster. Hence all the vertices of the cluster are very easily reachable from the star vertex. This vertex is usually the vertex with the maximum edges. It is possible for the graph to have many star vertices in it, but we select only one of them. This is explained in figure 4

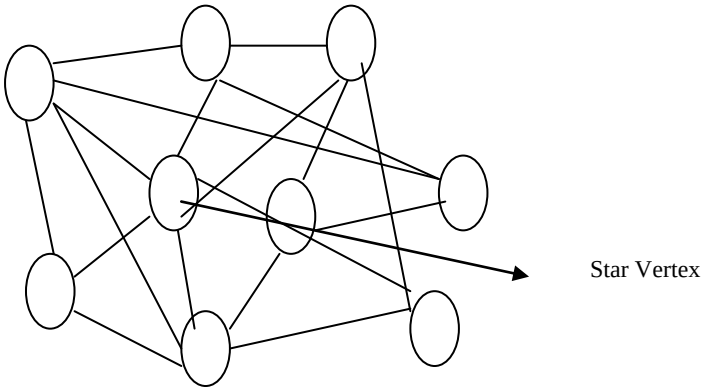


Fig. 4. Example of a cluster

Choosing vertices to form cluster: This is the main part of the algorithm that decide how the clusters would be formed. We lay down criterion that would enable us to select vertices that make up a cluster in the given graph

The general tendency is to traverse the graph and collect the vertices. We try to locate those vertices which have a lot of edges, since the chances of clusters are high at those points. Then we try to grab in points close to these vertices. We try to separate the areas which are well connected within it to the areas which are not so well connected.

In this problem, we define a cluster to be a set of vertices such that almost all the vertices are connected to each other. These vertices form a kind of well-connected architecture. Along with the edges within the members of the cluster, a vertex may have edges pointing to vertices outside the cluster. It is very possible for a vertex to

be members of 2 clusters, but while implementing this algorithm, we do not allow any 2 clusters to have a common vertex.

The algorithm to get a cluster from a graph is as given below. In any step we do not regard the vertices representing clusters (of same level) as normal vertices. They are treated as completely different entities.

getNextCluster()

Step1: Find the vertex v in graph with maximum edges
Step2: If the maximum edges are less than α then return null
Step3: c ← all vertices that are at a maximum distance of 2 units away from v
Step4: Sort c in order of decreasing number of edges of vertices
Step5: Select any 3 vertices v_1, v_2, v_3 in c such that all 3 vertices are connected to β common vertices
Step6: c_2 ← all vertices in c that are connected to v_1 and v_2 and v_3
Step7: Add all vertices in c to c_2 that are connected to at least 4 vertices already present in c_2
Step8: Return c_2

Here we have tried to get appropriate clusters that are distinctly visible from the surroundings, without making complex steps that would have consumed time. The inspiration of such an algorithm lies from the study of the traits of the data on which it is going to be applied.

As explained above, a cluster is a well-connected network of vertices. We first try to find a vertex that may be fit to serve as a star vertex. The best option is to use the vertex with the maximum edges (Step 1). Then we try to collect all possible points that are near to the star vertex. Since we need to keep all the vertices very close to the star vertex, we travel a maximum 2 unit distance (Step 3).

Now we know that our solution is a subset of this collection. It is very possible that the star vertex collected may be part of more than one cluster. If we do not shorten our collection, we are likely to get the union of all clusters as answer.

In order to restrict ourselves to only one cluster, we select any 3 vertices with the hope that they all belong to same cluster (Step 5). If they belong to the same cluster, they would be having many common vertices to which they are connected (Step 5). Also the common vertices to which they are connected would all be members of the cluster (Step 6). Here we assume that the 3 vertices selected all had high connectivity within the graph. The three vertices selected were also probable candidates for the star vertex.

In this manner we may select members of the cluster. In this method, it may be possible that a vertex is having less number of edges. It may be connected to only a few members of the cluster. Hence we again iterate through all the vertices in collection. If any of them is connected to sufficient number of vertices in the cluster, then we accept this vertex also to be a member of the cluster (Step 7). In this manner we are able to distinguish a cluster from its surroundings.

3.2 Searching

After the clustering is over, we should be able to perform the search operation. Just like the clustering was in multi levels, the searching also would be conducted separately in the entire levels one after the other. We would start with the highest level and reach the lowest level.

The first job in this regard is to find out the source and the destination at the various levels. We know that a source might have been replaced by a cluster at the first level. This new cluster might have been replaced by another cluster at the second level and so on. Hence we need to start from the basic given graph and trace the source and the destination at each level. At the end of this process, we would get to know at each level the points from where searching needs to start and the point where we end. The algorithm for finding out these points is as given below. It can even be summarized as shown in figure 5.

PointSearch
Step1: Level ← 0
Step2: While there are graphs in current level
Step3: If source is a member of any cluster of this level
Step4: Source ← Cluster number where it was found
Step5: If goal is a member of any cluster of this level
Step6: Goal ← Cluster number where it was found
Step7: Add source, destination to point set
Step8: Level ← Level + 1

Now we know the source and the goal node at each and every level. This means that our task is now to start the search from the source and try to reach the goal. We take a solution vector that would be storing the solutions of any level. When we find out the solution at any level, this is stored in this solution vector. When we move from a higher level to a lower level, then we pass this solution vector between levels. The lower level works on the solution vector generated from the higher level graph. It tries to put the points found in its graph, removing the ones found in the higher level clusters.

The algorithm for the search is given by the following algorithm

Search()
Step1: Solution ← null
Step2: For each (source, destination) in point set
Step3: $Solution_2$ ← start + all vertices in solution + destination
Step4: If any vertex in $solution_2$ is a cluster of the higher level
Step5: Replace that vertex with the star vertex of that cluster
Step6: Solution ← null
Step7: For all adjacent vertices (v_1,v_2) in $Solution_2$, taken in order
Step8: Solution ← Solution + bfs(current level graph,v_1,v_2) – v_2
Step9: Solution ← Solution + destination

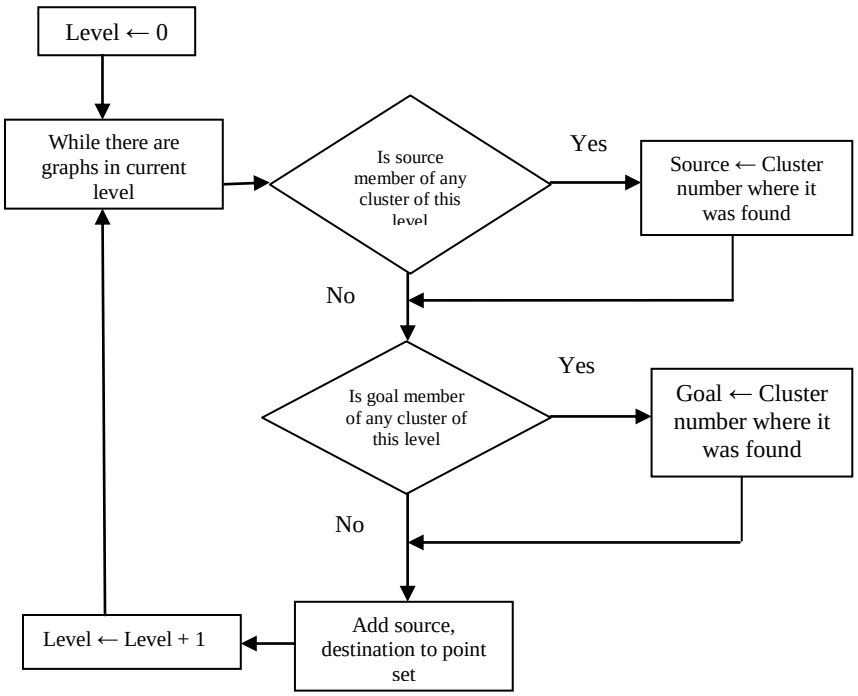


Fig 5. The point searching algorithm

Here function "bfs(current level graph,v_1,v_2)" refers to the standard breadth first search algorithm that acts on the current graph, takes the source as v_1 and destination as v_2. This algorithm returns the collection of vertices traversed from v_1 to v_2. In case it returns null, this means that the path is not possible. In our system if we get a null in the BFS algorithm, we break out of the complete algorithm, stating that the path was not possible.

In the algorithm we have purposefully deleted all destinations of the BFS algorithm. This is because the destination of the current step would become the source of the next step and would be repeated.

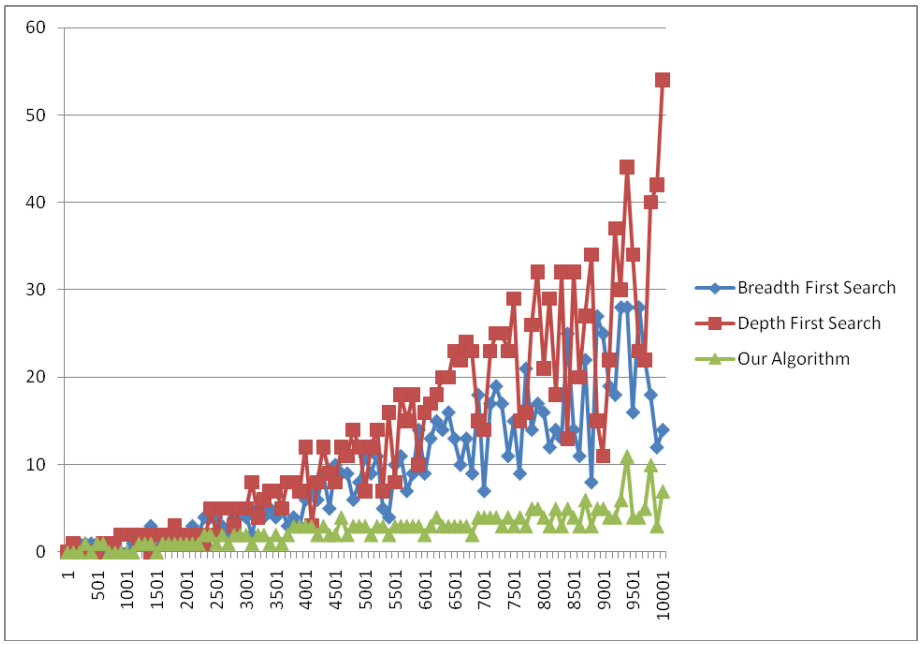
4 Results

In order to test the working of the algorithm, we coded the algorithm using JAVA and made the program to run in various test cases. Synthetic data was generated to test the algorithm. This data was generated keeping in mind the conditions over which the algorithm would be applied. Multiple search queries were fired to the algorithm. For all of these we found that the algorithm was able to solve the problem

and come up with a path from the source to the destination. In order to validate the results, we even coded the standard breadth first search and depth first search and compared the solutions generated by these three algorithms. It was observed that the breadth first search came up with the shortest paths. The depth first search used a lot of nodes and came up with complicated paths. Our algorithm came up with the shortest paths as well, but the algorithm does not guarantee the paths being the shortest.

When we increased the input size, the difference between the breadth first search and the depth first search and our algorithm was clear. The depth first search kept complicating. Our algorithm kept following the paths of optimal solutions and generated good results.

Time saving was the key point of the algorithm. We also studied the time taken by the three search algorithms. In the starting there was not much difference in the running times. As we increased the input size, the depth first search time increased dramatically. It was clear that this technique badly failed to generate results in time. Breadth First Search was comparatively much better and performed well in even quite large input sizes. But the time taken by it for large input sizes was very high as compared to our algorithm. Our algorithm proved to be the best of all three which performed very well when the graph had huge number of nodes.



We plotted the time taken by the three algorithms with respect to input size. This is given in figure 6. It may be noted here that the search nodes were opted randomly rather than those being the most distant or normally located. In the graph we have

Fig 6. Time v/s input size graph for the 3 algorithms

omitted the results which we got from the points that were found to be too close to the goal.

5 Conclusion

In this paper we have seen that we were able to search the graphs in almost the most optimal way. We used the multi layered clustering of graph that saved a lot of time that we could have wasted in searching the wrong nodes. Clustering shortened the graph to a great extent. Only the good areas, or the areas required were searched rather than the whole graph.

The experimental results clearly prove the efficiency of the algorithm. When we coded the algorithm and matched its efficiency to the existing algorithms, we clearly saw that our algorithm was very efficient and solved the problem very efficiently.

The basic motive of the algorithm is to apply it to huge amount of data in which searching may be a common activity. The algorithm would prove to be very useful in such situations. The algorithm of clustering is quite flexible and may be customized while implementing to match the needs of the particular database.

The basic construction of the algorithm is made keeping in mind the situation in which it would be applied. It may be noted that the algorithm would work for any general graph as well with better efficiencies. The application of this algorithm in other area needs to be studied. As well as clustering algorithms may be developed to adapt to the specific requirements.

A better study of the individual problems and the algorithm adaptation to these problems may be done in the future. Further we have not modeled the insert and delete operation that may affect the search operation. These operations need to be incorporated in the model and may be done in the future.

References

1. Anders Karl-Heinrich, 'A Hierarchical Graph-Clustering Approach to find Groups of Objects', In ICA Commission on Map Generalization, 5th Workshop on Progress in Automated Map Generalization (2003)
2. Arcaute Esteban, Chen Ning, Kumar Ravi, Liben-Nowell David, Mahdian Mohammad, Nazerzadeh Hamid, Xu Ying, 'Deterministic Decentralized Search in Random Graphs', Proceedings of the 5th Workshop on Algorithms and Models for the Web-Graph, (2007)
3. Brandes Ulrik, Gaertler Marco, Wagner Dorothea, 'Experiments on Graph Clustering Algorithms', In Proceedings of the 11th Annual European Symposium on Algorithms. Lecture Notes in Computer Science, vol. 2832. 568--579 (2003)
4. Craswell Nick, Szummer Martin, 'Random Walks on the Click Graph', SIGIR Conf Research and Development in Information Retrieval, 239—246 (2007)
5. Goldberg Andrew V., Harrelson Chris, 'Computing the Shortest Path: A* Search Meets Graph Theory', In Proceedings of SODA, 156—165 (2005)
6. Gunter Simon, Bunke Horst, 'Validation indices for graph clustering', Pattern Recognition Letters 24, 1107--1113 (2003)

7. He Hao, Wang Haixun, Yang Jun, Yu Philip S,' BLINKS: Ranked Keyword Searches on Graphs', in the ACM International Conference on Management of Data (SIGMOD), Beijing, China.(2007)
8. Hlaoui Adel, Wang Shengrui, 'A Graph Clustering Algorithm with Applications to Content-Based Image Retrieval', Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, (2003)
9. Kacholia Varun, Pandit Shashank, Chakrabarti Soumen, Sudarshan S., Desai Rushi, Karambelkar Hrishikesh, 'Bidirectional Expansion For Keyword Search on Graph Databases' ACM Proceedings of the 31st international conference on Very large data bases Trondheim, Norway (2005)
10. Najork Marc, Wiener Janet L., 'Breadth-First Search Crawling Yields High-Quality Pages', ACM Proceedings of the 10th international conference on World Wide Web Hong Kong, (2001)
11. Rattigan Matthew J, Maier Marc, Jensen David, 'Graph Clustering with Network Structure Indices', ACM Proceedings of the 24th international conference on Machine learning Corvalis, Oregon, (2007)
12. Tadikonda Satish K., Sonka Milan, Collins Steve M, 'Efficient Coronary Border Detection Using Heuristic Graph Searching' ieeexplore
13. Wang Yonggu, Li Xiaojuan, 'Social Network Analysis of Interaction in Online Learning Communities' ICALT 2007. Seventh IEEE International Conference on Advanced Learning Technologies, (2007)
14. Yushi Jing, Shumeet Baluja, 'PageRank for Product Image Search', WWW 2008 / Refereed Track: Rich Media, (2008)
15. Zhou Rong, Hansen Eric A, 'Sparse-Memory Graph Search', 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico (2003)