

Scalability for Social Network Analysis Algorithms

Tor Kvernvik
Ericsson Research,
Färögatan 6,
164 80 Stockholm, Sweden
+46107147646

Tor.kvernvik@ericsson.com

Fredrik Hildorsson
Uppsala University,
Box 337,
751 05 Uppsala Sweden
+46702498876

Fredrik@Hildorsson.org

ABSTRACT

A Social network is a social structure made up of individuals called “nodes” which are tied by “edges”. One example of a source for social network analysis is traffic in a mobile telephony network where the subscribers are ‘nodes’ and the ‘edges’ are derived from the telecom traffic in the mobile telephony network. Mobile networks and other social networks of today contain millions of subscribers and the growth is expected to continue for the foreseeable future. Many of the social network analysis algorithms are considered to be computationally expensive to execute. In this paper we describe the analysis and conclusions from our studies related to scalability for social network analysis algorithms. For some algorithms scalability is improved by introducing approximations in the existing algorithms. Many social network algorithms perform the analysis globally on the complete network. By performing the analysis for the closest neighbors to the target node, the ego network, the scalability is improved. Solutions for parallelized execution of the algorithms are also described. The most challenging part was to efficiently partition the network in smaller segments to prepare for parallel execution of the algorithms. A novel idea for handling pre-partitioning based on geographic location of the subscribers in a mobile network is described. As a proof of concept a prototype was implemented. Evaluation of our prototype implementation and ideas for further improvements are also described. Performance tests to compare the original with the improved social network algorithms and the parallelized solutions shows considerable improvements related to scalability. The trade off between capacity and quality of the results for these solutions and algorithms is also evaluated.

Categories and Subject Descriptors

General Terms

Algorithms, Experimentation, Performance, Verification.

Keywords

Social Network Analysis, Scalability, Partitioning.

1. INTRODUCTION

The number of subscribers and traffic is increasing rapidly in the mobile telephony networks. There are social network analysis methods available that can help to derive valuable information from traffic data in telecom networks.

Examples of usage:

- Identifying influential subscribers e.g. gateways
- Identifying highly connected groups of subscribers.

One of the main challenges is to provide scalable solutions that can handle networks with millions of nodes with reasonable computational resources and execution time. Many of the social network algorithms are mainly designed for smaller networks with a few thousands of nodes.

This paper analyzes how to handle social network analysis for larger networks with million of nodes:

- How scalable are the different social network algorithms executing in serial?
- How can scalability be improved by more efficient algorithms e.g. approximation algorithms? How does this impact quality?
- How can the scalability be improved by executing the algorithms in parallel? How does this impact quality i.e. some information may be lost when the graph is split on different processors?

The paper is organized in the following sections. Section 2 gives an introduction to social network analysis, the most used algorithms and how they can be optimized for scalability. Section 3 describes how the scalability can be improved by parallelized execution. Section 4 describes the results from the prototype implementation and the performance tests. Section 5 describes the conclusions from the study.

2. SOCIAL NETWORK ANALYSIS METHODS & ALGORITHMS

We have focused on two of the main methods for social network analysis, Centrality and Clustering.

2.1 Centrality

Centrality is used to find how central a node is in the network. A central node can mean a number of things and therefore there are many different definitions of centrality.

Ego network analysis

A network can be analyzed from different viewpoints i.e. ego network or complete network. With complete network analysis all nodes in the network may be considered. In ego network analysis the analysis is viewed from a single node and only the nodes in the closest neighborhood are considered. The ego network may have different ranks depending on how many steps away from the ego the analysis is performed. Figure 1 shows an ego network of rank one. This means that it consists of an ego node and the neighbors’ one step away from the ego.

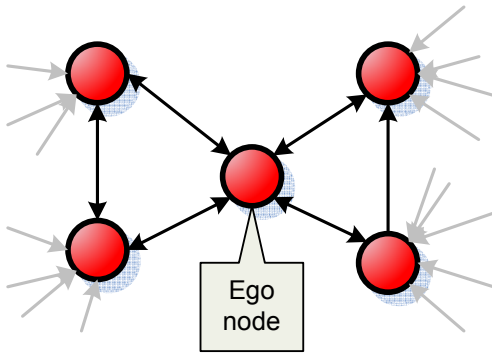


Figure 1. Ego node and ego network (rank=1)

2.1.1 Degree Centrality

Degree centrality measures the influence of a node from and to its closest neighbors. The degree of a node is estimated by adding the number of neighbors the analyzed node has one step away. Degree scales linearly with the number of nodes and is thereby scalable, i.e. with time complexity $O(n)$ where n is the number of nodes, and is suitable for very large networks.

2.1.2 Eigenvector Centrality

Eigenvector centrality is similar to degree centrality but takes a wider part of the network into consideration and thereby gives a better prediction of influence than degree centrality. A node with few neighbors can still have a high eigenvector centrality if its neighbors have a high level of centrality. The Eigenvector algorithm scales almost linear with the number of nodes. For a sparse graph it scales with time complexity $O(n + m)$ where n is the number of nodes and m the number of edges. This means that it can be scaled up to a large network with millions of nodes.

2.1.3 Betweenness Centrality

Betweenness is one of the most used centrality methods. Nodes with high betweenness have a high level of control of the information floating between the different groups of nodes in the network. Subscribers in a telecom network with high betweenness are usually considered to be suitable for targeted marketing.

The original betweenness algorithm i.e. global betweenness has poor scalability. The reason is that the calculation considers the path for all nodes in the network through the target node. With some optimization algorithms it can be calculated with time complexity $O(n \times m)$.

A more capacity efficient alternative is ego network betweenness. According to M. Everetta and S. P. Borgatti [5] there is a strong correlation between the result of betweenness in the ego network and betweenness in the entire network. The largest graph they evaluated was a 500 node network. In the ego network version of betweenness centrality it is only the closest neighbors, i.e. one step away from the target node, that are considered in the calculation and the algorithm is thereby more scalable.

Ego network betweenness scales with time complexity $O(n)$.

2.1.4 Closeness Centrality

Closeness is a measure of how close each node is to all other nodes in the network. Most usually the geodesic distance is used but also other variants exist. The original closeness algorithm scales very poorly. Closeness in a large network does not make sense as described by M. Everetta and S. P. Borgatti [5] and is therefore not further elaborated in this paper.

2.2 Clusters

A group of nodes that are more closely connected in a network is often called a cluster. We have identified two ways to find clusters, definition driven or hierarchical clustering. In a definition driven cluster there are rules that defines what the relationship between the nodes within each cluster is. The main difference between the two is that each node in a network can only belong to one cluster created from hierarchical clustering but to several definition driven clusters at the same time. We have analyzed both cliques as an example of definition based clustering method and communities as hierarchical clusters.

2.2.1 Cliques and extensions

In the original clique definition all nodes in one clique needs to be directly connected to each others. With this very stringent definition usually very few large cliques exists in a telecom network. Usually only a few cliques consist of more than three nodes. All maximal cliques can be calculated with the Bron-Kerbosch algorithm [3]. The time complexity is close to linear with the number of cliques. Since the number of cliques increases rapidly with the size of the graph the algorithm is not scalable for large networks.

There are several more inclusive extensions to this definition i.e. K-plex, K-core or N-cliques. With these looser definitions more large cliques are found. All these algorithms are computationally expensive to calculate and not scalable in a large network.

There are a number of approximation algorithms with varying trade off between performance and quality. Most of them are based on the simple greedy algorithm also called local search. The local search algorithm starts out with an ego node. Then the neighbors with the highest degree are added to the clique. This is repeated until there are no more nodes to add to the clique. Some extensions to the local search algorithm are described by R. Battiti and M. Protasi in [2] & D.-Z. Du et al. [4]. The Clique analysis algorithm that we implemented in the prototype and tested originated from the greedy algorithm and it scales almost linearly with the size of the graph with time complexity $O(n)$.

2.2.2 Communities

Community structure also refers to the occurrence of groups of nodes in a network that are more densely connected internally than with the rest of the network. The difference to cliques is that communities are based on hierarchical clustering. The hierarchical clustering can be either top-down or bottom-up. The top-down algorithm starts with the whole network as a single cluster and then splits up the network in smaller and smaller parts. Most community algorithms are based on using the betweenness algorithm in order to separate the network in parts. This means that the betweenness for all nodes in the network is calculated and the nodes or edges with the highest betweenness are removed. This procedure is repeated several times until the network is separated

in several parts i.e. communities. The procedure is repeated until the network is separated in a predefined number of communities or until a certain quality of the clustering is reached. The scalability for this algorithm is very poor, mainly due to that the betweenness algorithm has to be executed several times. As mentioned in section 2.1.3 the global betweenness is expensive to calculate. The time complexity is approximately $O(n^3)$.

By replacing the global betweenness with ego network betweenness as described in section 2.1.3 the computation time is decreased considerably.

3. PARALLEL EXECUTION OF THE SOCIAL NETWORK ALGORITHMS

Increased efficiency of the social network analysis algorithms is the primary choice to increase scalability. With the expected subscriber and traffic growth in the telecom networks, parallel execution of the algorithms will also be required to even further reduce the execution time. By separating the graph into multiple partitions each partition of the graph can be executed on different processors in parallel. The number of edges between the nodes in the different partitions, the edge cut, needs to be minimized in order to enable parallel execution with minimal interaction between the partitions and thereby minimize the quality degradation or capacity loss due to the need of inter processor communication.

Figure 2 shows a small partitioned social network graph. The nodes correspond to subscribers (end users) in a telecom network. The nodes are connected via edges that correspond to connections, derived from e.g. telephone calls in a mobile network, between the nodes. The dotted lines show the partition borders. Usually the partitions shall be balanced, roughly the same size.

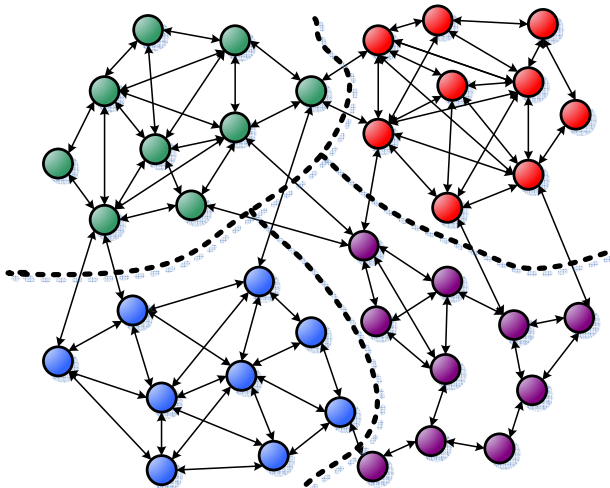


Figure 2. A partitioned social network graph

There are a large number of alternative partitioning methods. One of the most efficient methods for large networks is a k-way multilevel partitioning method developed by A. Abou-Rjeili, V. Kumar and G. Karypis [1][7]. This method was implemented in the prototype. The partitioning consists of three basic steps before

the social network algorithms are executed. Figure 3 shows an overview of our prototype and the different steps in graph partitioning. Most partitioning algorithms including k-way multilevel partitioning starts with the network initially separated in partitions and then the algorithm iteratively improves the partitioning by moving nodes between the partitions. This initial separation of nodes may be random but in order to make the execution of the partitioning algorithm quicker it is necessary to perform a pre-partitioning prior to the real partitioning. Figure 3 describes the basic steps of social network analysis and partitioning. Each step is described in more detail in the following sections 3.1 & 3.2.

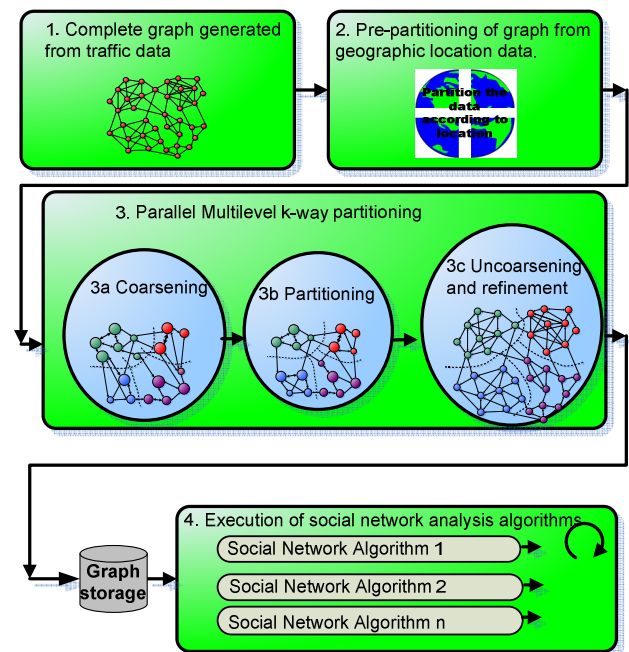


Figure 3. Preparation of social network graph for parallel execution.

1. In the first step a graph corresponding to the whole data set is generated from the traffic data e.g. Call Detail Records (CDRs) in a telecommunication network.
2. As a novel idea for this prototype the geographical data from the CDRs records is used to perform pre-partitioning of the graph (section 3.1.)
3. The parallel multi-level partitioning (section 3.2) is performed in three steps.
 - a. The graph is compressed via a coarsening algorithm.
 - b. A detailed (inner) partitioning is executed.
 - c. The graph is un-coarsed and the partitioning is refined for the un-coarsed graph.
4. The social network analysis algorithms are executed on the different partitions of the graphs in parallel. Several different Social Network Algorithms can be executed on the same data set without having to redo the partitioning.

3.1 Geographical location based pre-partitioning

Before the detailed partitioning an initial pre-partitioning is performed. This should be a fast algorithm to be used as a starting point for a more detailed partitioning scheme. We developed a novel method i.e. the geographical partitioning method as illustrated in Figure 4.

This method originates from the assumption that there is a strong relation between the number of calls between two subscribers and the geographical distance between the subscribers i.e. a majority of the calls are executed locally in most telephone networks.

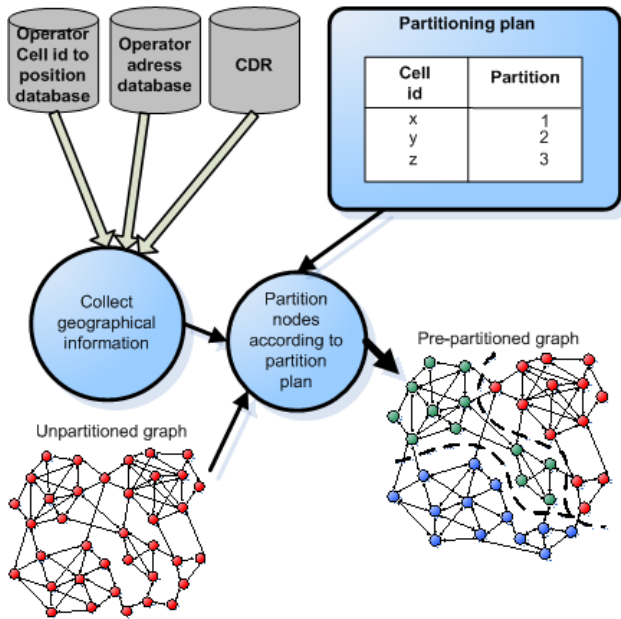


Figure 4. Geographical location based pre-partitioning.

R. Lambiotte et al. [9] shows that the probability that subscribers are connected is proportional to the distance between them up to a distance (d) of 40 km. The probability that two nodes are connected follows a power law decrease $\sim 1/d^2$.

The geographical location of the subscriber can be retrieved via information in the CDR. The CDRs are stored in a database in the telecommunication operator's network and contains information about the traffic in the telephony network including the geographic location of the subscribers. Other alternative sources for the location are the operator's address data base or other location databases.

A partitioning plan is prepared by an experienced network planner with knowledge about the demographics and the traffic patterns relation to the geographic location of the subscribers. The partitioning plan is used to map between the main geographical location e.g. cell identity of the subscribers and the partitioning they shall be located to.

The result is a graph that is pre-partitioned according to the main geographic locations of the subscribers. This means that for example subscribers that have the same city as their main location are likely to be located in the same partition.

Since there were no data from a real telecom operator available we had to simulate the geographical location based pre-partitioning in the prototype by adapting the data set. See section 4.1.

3.2 Multi level partitioning

We have implemented and tested a variant of the multi level k-way partitioning developed by A. Abou-Rjeili, V. Kumar and G. Karypis [1][7]. This is an efficient solution for partitioning of large social network graphs. It consists of three basic steps coarsening, inner partitioning and Un-coarsening. Figure 5 shows an example of multi level partitioning. Figure 3 shows the implemented version and the complete set of procedures with step 3 displaying the multilevel partitioning.

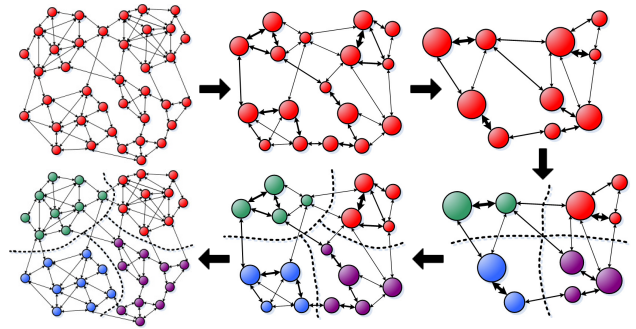


Figure 5. Multi-level k-way partitioning example

3.2.1 Coarsening

Prior to the inner partitioning the graph is coarsened, step 1-3 in figure 5. This means that the graph is compressed to only a few nodes and edges. The coarsening is done in several steps and in each step the nodes that are closely located are merged. A method called coloring is used to make sure that each node is only merged once during each step. The merged nodes get the summary of the weights of the edges of the merged nodes. This process is repeated several times until only a few nodes remains. In the implemented prototype it was decided to perform the coarsening locally on each partition to enable parallel execution of the coarsening.

3.2.2 Inner partitioning

The inner partitioning, step 4 in figure 5, is performed on the coarsened graph. We implemented a very simple partitioning algorithm where the nodes closest to the borders of each partition are evaluated and may be moved to the closest partition. The algorithm was based on the Kernighan-Lin partitioning algorithm [8]. After each move the edge cut is calculated. This procedure is repeated until a minimal edge cut is reached.

3.2.3 Un-coarsening & refinement.

After the partitioning the graph is un-coarsened, step 5-6 in figure 5. This means that the nodes and edges are split to their original structure again. Due to the coarsening the partitioning of the compressed graph is a bit rough. During the un-coarsening the partitioning is refined by evaluating movement of the border nodes. The refinement is based on the Kernighan-Lin partitioning algorithm [8].

4. PERFORMANCE TESTS

As a proof of concept a prototype solution, following the principles described in figure 3, was implemented to verify our ideas. We performed a number of tests in order to evaluate the selected methods and algorithms regarding scalability and quality of the result.

First a number of tests were executed with the original algorithms and then the same tests were performed with the approximating algorithms. The execution time, memory consumption and potential quality degradation of the results were measured.

The execution time of the graph partitioning procedures and the performance of the social network analysis algorithms executing on the partitions in parallel was also tested

The tests were performed on several graph sizes and data sets generated from different versions of graph generator algorithms.

4.1 Data sets

The lack of large data sets from real telecom traffic forced us to generate graphs that as much as possible reflects the characteristics of a real telecom network. Several graph generation algorithms were tested i.e.

- **Barabasi Albert** Scale free graphs
- **Watts Beta** Small world graphs
- **Eppstein Power Law** Scale free graphs
- **Kleinberg** Small world graphs
- **Random graph** The neighbors are totally random.

The main issue with the data sets was that none of these tools is able to generate one graph with all the characteristics that are typical to a social network graph originated from real telecom traffic data. As noted in [10] the telecom call graph is both scale-free and small world at the same time. A social graph is also more clustered than what can be explained by a random model as noted in [10] and a social graph also have assortative mixing (i.e. positive correlation) between the degrees of adjacent nodes. As mentioned none of the available tools is able to generate a graph with all these characteristics.

Most of the performance tests were performed on a randomly generated graph. The creation of edges was weighted with a Gaussian distribution so that closer nodes are more likely to become a neighbour. This gives a simple data locality which the geographical partitioning (3.1) is assumed to have given if that had been implemented and data from real telecom traffic had been available.

4.2 Results

Some of the most important results from the performance tests are described.

4.2.1 Social Network Algorithms

The scalability and the quality of the results for the original social network algorithms were tested. For the less scalable original algorithms a more optimized algorithm was also tested.

In table 1 the results are compared between the original algorithms centrality global betweenness and the corresponding more scalable algorithms ego network betweenness. A corresponding comparison is performed for the original clique (K-plex) algorithms and the greedy clique (K-plex) algorithm. Both

the execution time and the quality are compared. The approximations are executed on two cores. The tests in table 1 were performed on a graph with 100K nodes. The time for the graph partitioning is not included in table 1.

Degree centrality was not tested since it scales linearly with the number of nodes and does not cause any scalability issues.

Table 1. Comparison between original algorithms and approximation algorithms executed on a randomly generated graph with 100K nodes

	Scalability	Time complexity	Execution time example:	Quality
Global Betweenness centrality	Not scalable	$O(n \times m)$ $O(n^4)$	~ 10 hours with 1 core	Exact betweenness value for all nodes.
Ego network betweenness centrality	Scalable	$O(n)$	1.3 seconds with 2 cores	Good at finding nodes with high betweenness.
Bron-Kerbosch Clique algorithm	Not scalable	$O(\#cliques)$	~ 6 min with 1 core	Finds all cliques in the graph.
Greedy K-plex(clique) algorithm	Scalable	$O(n)$	3.1 seconds with 2 cores	Finds most of the large cliques in a graph.

Eigenvector centrality

The eigenvector algorithm scales well. Our tests show that the execution time increases only slightly faster than linear compared to the size of the graph. This means that the eigenvector algorithm is well suited for large networks. The eigenvector algorithm was implemented to execute in parallel on the processors. This did not cause any quality degradation since the eigenvector values of the border nodes are communicated between the processors prior to each iteration. In figure 6 the execution time for a 500 K nodes graph is presented e.g. 31 seconds on eight cores.

Betweenness centrality

The global betweenness does not scale well. It is not scalable for large networks. The test on a 100 k nodes graph took approximately 10 hours with one core (table 1).

The ego network betweenness was tested as a more scalable alternative. As expected it scales almost linear to the size of the graph. The execution of a 100 k nodes graph used 1.3 seconds executing on two cores (table 1). Tests (figure 6) showed that it takes approximately 10 seconds to execute the analysis of a 500k nodes graph on 2 cores. The quality of the result, between the approximation and the original algorithm, correlates well for

nodes with high level of betweenness. The highest ranked 10 % of each of the two betweenness algorithms consists to 90% of the same nodes. However for nodes with lower level of betweenness the ego network betweenness did not correlate as well. This can be improved by increasing the rank of the ego network betweenness. That is for example to include neighbor nodes of rank 2, i.e. neighbors up to two steps away from the ego, instead of rank one that was used in the prototype.

Cliques (K-plex)

The original algorithm for calculating cliques, Bron Kerbosch [3][4] was tested. The time complexity for the original algorithm is bad. The execution time increases linearly with the number of cliques. The number of cliques increases rapidly with the size of the graph and thereby did the execution time not scale well with the size of the graph.

A greedy clique approximations algorithm was implemented and tested as a more scalable alternative. It scales well with the size of the graph, almost linear with the number of nodes.

In table 1 a comparison is made between original Bron-Kerbosch algorithm executing on one processor and the greedy clique algorithm executing on two processors. The results show a considerable reduction in execution time i.e. 3.1 seconds compared to 6 minutes.

Since the implemented algorithm treats each partition as a separate graph, cliques that cross the partition borders is not found. The greedy clique algorithm was good at finding large cliques but only finds a part of the smaller, less than 4 nodes cliques. For the larger cliques with more than three nodes the approximation found 90 % of the cliques. There are several ways to improve the accuracy but that will be to the cost of more computation time. The algorithm may be improved to also identify cliques that cross the partition borders but this requires more inter processor communication and will impact the scalability.

In most cases the network operator is mainly interested in finding the larger cliques and thereby it is most efficient to use the greedy clique algorithm developed in the prototype.

Communities

The original community algorithm has a high time complexity $O(n^3)$ where n is the number of nodes. The main reason is that it is based upon the global betweenness algorithm. The average execution time for a graph with 100 nodes was 6 seconds and for a graph with 1000 nodes almost 2 hours. It was not possible to test on larger graphs than 1000 nodes since it took too much time to execute. This means that it is not usable in large networks with million of nodes.

By replacing the global betweenness algorithm with the ego network betweenness algorithm the execution time should be reduced considerably, approximately about 5 times. The execution time would be even further reduced since only the affected nodes needs to be calculated after each removal of a node.

Modularity can be used to compare the quality degradation of the result. Modularity can be seen as the proportion between edges inside the cluster and edges going out from the cluster. A high modularity means a good clustering.

Due to time constraints we were unfortunately not able to implement the community algorithm with the ego network betweenness algorithm.

Execution time for parallel processing of Social network algorithms

Figure 6 shows the execution time for our implementation of some of the more scalable social network analysis algorithms and how they scale with the number of processors for a 500K nodes graph.

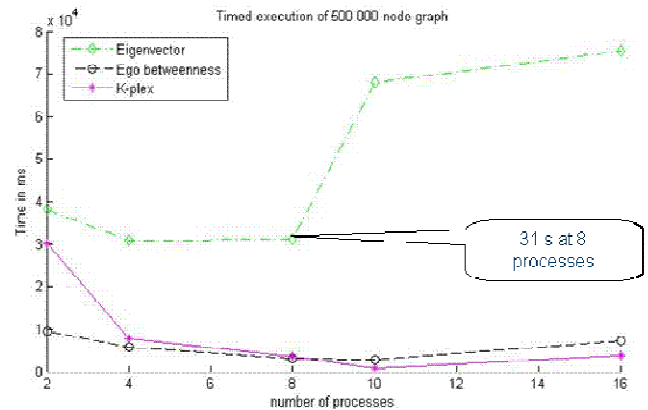


Figure 6. Scalability for the social network analysis algorithms. Randomly generated graph

When the execution of the algorithm is distributed on more processors the execution time is expected to be reduced. However the inter processor communication is also increased with an added number of processors. This reduces the added execution power gained from the increased number of processors. The tests (figure 6) showed that reduction of the execution time is significant for up to 4 parallel processors but for more processors the gain is reduced. The eigenvector algorithm needs inter processor communication for edges that crosses the partition border. This is the main reason for the dramatic increase for the eigenvector algorithm, since the number of edges crossing the partition borders increase with the number of processors.

As an example the eigenvector algorithm required 31 seconds to execute on 8 processors on a 500 k nodes graph.

4.2.2 Graph partitioning

Since our tests has shown that the faster social network algorithms may be too slow to execute in a network with million of nodes the execution of the algorithms must be parallelized. This means that the social network graph must be partitioned. The partitioning was the most challenging part of the prototype. We implemented and tested a variant of the multi level k-way partitioning [1][7]. Figure 3 shows the different steps in the implementation. Since the multilevel k-way partitioning is quite complex and consists of many different algorithms there are a lot of different settings required in the algorithms. In the tests we used the values recommended by G. Karypis and A. Abou-Rjeili [1].

The execution time for the detailed partitioning was tested on several levels of partitioning, e.g. 2-16 processors, and several sizes of the network. The graph in Figure 7 shows the time that was required to perform the partitioning of a 500K nodes graph. The graph was generated randomly with simple data locality 4.1.

The total time and the time for the different steps of the partitioning were measured. The total partitioning time was expected to increase close to linear with an added number of processors. As shown in figure 7 the execution time was increasing almost linear up to 8 processors. For more processors the execution time increased faster than expected. The inner partitioning contributed to the main part of the increase. The lack of efficient coarsening as a part of the multi level partitioning is expected to be the main reason for the performance issues of the partitioning. The coarsening in the prototype was executed locally on each partition and was thereby not able to reach the expected level of coarsening. This resulted in a larger graph than expected for the inner partitioning. The inner partitioning was more time consuming than expected due to the larger graph. As an example total execution time of the partitioning on 10 processors of a 500K nodes graph was 25 minutes and 26 seconds.

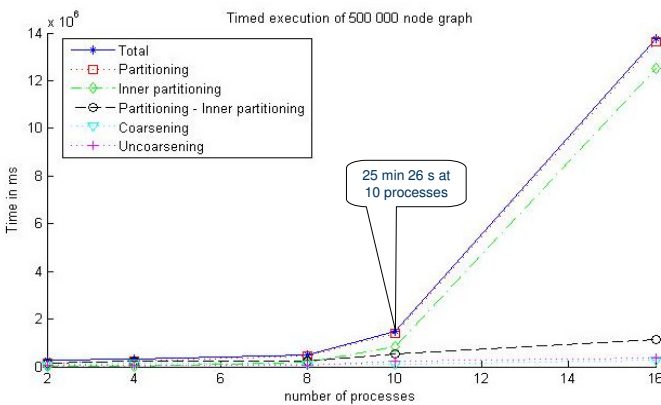


Figure 7. Execution time for the graph partitioning. Randomly generated graph

4.2.3 Total execution time for graph partitioning and social network algorithms

Since the partitioning of the graph is a prerequisite for the parallel execution of the social network algorithm it is essential to compare how the total time of partitioning and social network analysis algorithms scales with the number of processors i.e. step 3 & 4 in Figure 3.

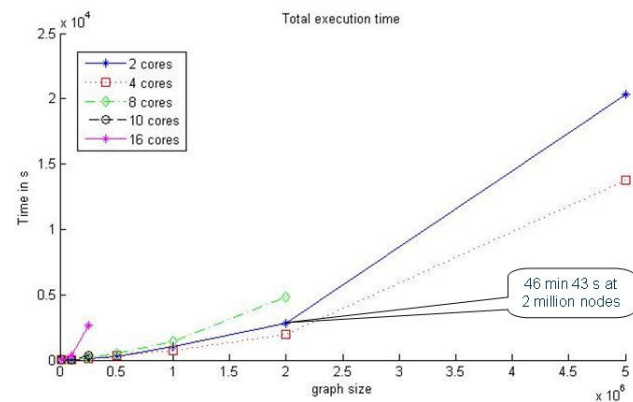


Figure 8 shows how the total execution time scales with the number of processors, randomly generated graph

The tests were performed on different sizes of graphs and the partitioning is followed by the execution of three different social network algorithms Eigenvector, Ego-betweenness and greedy Kplex (clique). The results from the tests are displayed in figure 8.

None of the cases scaled linearly. The tests with four processors had the best scalability. A more efficient partitioning algorithm is required to be able to efficiently execute on more than four processors. The tests on 16 processors were aborted for graph sizes above 25K nodes due to too long execution time.

As an example 46:43 minutes was used to partition a two million nodes graph on two processors and execute the three social network analysis algorithms.

5. CONCLUSIONS

The analysis, prototype implementation and performance tests have concluded that it is possible to analyze the social network for large telecom networks with millions of subscribers. As an example the proof of concept prototype proved that it is possible to analyze a 2 million nodes network in less than one hour.

This requires capacity efficient social network algorithms that in some cases use approximations or ego networks. The proof of concept implementations shows that the social network analysis algorithms ego network betweenness, greedy clique and eigenvector algorithms scales well and outperforms the corresponding original algorithms in terms of scalability.

As expected there is a trade off between the quality and performance. The centrality approximation algorithms are good at finding nodes with high centrality i.e. alpha subscribers. The greedy clique algorithm is very efficient to identify larger cliques.

In most cases of network analysis the main focus for the network operators is to find these nodes or larger cliques that really deviate from the average. It is also possible to modify the level of approximation and thereby increase the quality on the cost of longer execution time.

The tests have also proved that it is possible to reduce the execution time even further by using parallel execution. This requires partitioning of the graph. The partitioning was more complicated than expected.

The execution time for the partitioning scaled well for up to 4 processors, but increased rapidly for higher numbers. It is possible to increase the efficiency of the partitioning also for more processors. It requires more efficient partitioning algorithms than what was implemented in the prototype. It is mainly the coarsening algorithm that needs to be improved. Another reason is an underestimation of the number of edges between the partitions that gets cut by the partitioning. A high level of edge cut impacts the execution time since some social network analysis algorithms e.g. eigenvector, needs to fetch information from the other neighboring partitions for the edges that cross the partition borders. The quality of the result for some of the algorithms also get reduced e.g. in case of clusters or cliques that overlaps two partitions.

Due to lack of data from real networks and graph generators not being able to generate all the characteristics of a real network, there is an uncertainty how well the results correlate to a scenario with data from real networks

However the execution time for the partitioning is less critical than the execution time for the social network analysis algorithms since it is only executed once per network data set. Usually several social network analysis algorithms are executed on the same partitioned graphs.

For smaller updates, e.g. updates of the traffic records in a telecom network on a monthly basis; it is possible to perform a quicker partitioning. No coarsening or pre-partitioning is required. It is possible to perform the partitioning with the partitioning results from the last partitioning as starting point. The new or updated nodes are spread between the partitions via the geographical location information in the traffic data. Then the inner partitioning can be efficiently executed without any need for coarsening or pre-partitioning.

6. REFERENCES

- [1] A. Abou-Rjeili and G. Karypis, 'Multilevel Algorithms for Partitioning Power-Law Graphs', 2005
- [2] R. Battiti and M. Protasi, 'Reactive Local Search for the Maximum Clique Problem', *Algorithmica*, Vol. 29, no. 4, 2001, pp. 610-637
- [3] F. Cazals and C. Karande, 'A note on the problem of reporting maximal cliques', 2008
- [4] D.-Z. Du, P. M. Pardalos, 'The Maximum Clique Problem', *Handbook of Combinatorial Optimization: Supplement Volume A*, Springer, 1999, pp. 1-74
- [5] M. Everetta and S. P. Borgatti, 'Ego network betweenness', 2005
- [6] F. Hildorsson, 'Scalable solutions for social network analysis', 2009
- [7] G. Karypis and V. Kumar, 'Parallel Multilevel k-Way Partitioning Scheme for Irregular Graphs', 1996
- [8] B. W. Kernighan and S. Lin, 'An Efficient Heuristic Procedure for Partitioning Graphs', 1970
- [9] R. Lambiotte, V. D. Blondel, C. de Kerchove, E. Huensa, C. Prieur, Z. Smoredac, P. Van Doorena, 'Geographical dispersal of mobile communication networks', *Physical Review E* 68, 2003
- [10] A. A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjee and A. Joshi. 'On the Structural Properties of Massive Telecom Call Graphs: Findings and Implications', *Proceedings of the 15th ACM international conference on Information and knowledge management*, 2006, pp. 435-444
- [11] M. E. J. Newman and Juyong Park, 'Why social networks are different from other types of networks', 2003
- [12] S. Wasserman and K. Faust, *Social Network Analysis – Methods and Applications*, Cambridge University Press, 1994