# Implementation and Behavioural Analysis of Graph Clustering using Restricted Neighborhood Search Algorithm

Mohit Kumar K. K. Agrawal Dr. Deepak Arora Reena Mishra

Department of Computer Science and Engineering,
Amity University, Lucknow,
Uttar Pradesh, India.

## ABSTRACT
Restricted Neighborhood Search Algorithm or RNSC is a cost-based clustering technique for clustering the graph into separate clusters, where each cluster has some similar properties. The properties considered in this case are low inter-connectivity and high intra-connectivity in clusters. This is implemented only for un-weighted and undirected graphs. This algorithm applies a heuristic approach in which we only consider the moves in the restricted neighborhood of previous move, i.e., after a move has been made the next move will only be allowed in the neighbor clusters of that move. After a fixed number of moves we apply diversification moves to avoid the solution reaching local-minima in-spite of global solution. A tabu list is also maintained to avoid same moves which it has made in the recent past. This technique reduces the run-time of clustering algorithm many-folds, as it skips those redundant cases which occur multiple times and don't improve the cost of the function. The plus point of this algorithm is improvement in run-time due to the data-structure it is maintaining to update the clustering. The updating of the clustering is very fast and thus makes the algorithm fast. The paper proposes an effective behaviour analysis of some parameters of this algorithm which may help in the future modification of this algorithm. For better analysis of RNSC algorithm we have used Random Scaled-Free graphs having more than 10,000 nodes. Thus in our approach RNSC algorithm is successfully implemented in C++ for a graph of more than 10,000 nodes.

## General Terms
Graph Clustering, Data mining et. al.

## Keywords
Graph clustering, Tabu search, Destructive diversification, Shuffling diversification, naïve move, Scaled move, Move cost, Neighborhood search, RNSC.

## 1. INTRODUCTION
Restricted Neighborhood Search Algorithm (or RNSC) is a cost-based clustering algorithm. It uses the local search method to improve the clustering. The goal is to improve the clusters by grouping nodes of high intra-connectivity inside a cluster and to keep the inter-connectivity among different clusters minimum. It can also be described as dense intra-connectivity of nodes in same cluster and sparse inter-connectivity of nodes in different clusters. The cost of the current clustering is computed at every step and we try to minimize the cost to obtain better clustering. After making a move, it searches only in the neighborhood of the move to search for the next move. It selects the best move available in the neighborhood and takes that move under certain parameters, which will be discussed later. The neighborhood of the move is described as those moves which either originate or terminate at either the source or destination cluster of that move. The main advantage of this algorithm is its large list of data structure which decreases the run-time. The memory requirement of this method is O $(n^2)$. This is due to storing the graph in adjacency list format which has worst case memory requirement of O $(n^2)$ when the graph is completely connected. Also it searches the next move according to the performance criteria, i.e. the cost function. Other algorithms first compute the clustering then compute the performance criteria (like in MCL [7]). It has also implemented some features to avoid local-minima by taking some diversification moves, i.e. moves which are not in accordance with the local searching technique and making a diversification move to scatter the current cluster, as we use in search annealing. It also uses a tabu list which contains the list of all the nodes recently visited in the running called tabu nodes and those nodes are avoided for some time as retracing those nodes will not improve much the clustering. For further details on the tabu search refer the paper referred in bibliography. This algorithm uses the restricted neighborhood search heuristic or also called the variable neighborhood search. In the current case the two clustering are considered neighbour if one can be reached from one cluster to another just by moving a single vertex from first cluster to second or vice-versa. The moves are allowed to only those clusters in which it already has an adjacent node, otherwise that move is discarded because it will not improve the cost of the cluster much. Also some to empty clusters are made at regular interval, called ghost moves to avoid local minima. RNSC is a type of local search algorithm, because once a cluster is made, it will only looks at a new clustering that it can reach by moving a single node. The clustering algorithm uses a local search technique so an optimal solution is not guaranteed but we expect better results by running the algorithm multiple times on the same graph and taking the best clustering out of those experiments. The advantage of this algorithm is it first computes the moves and then makes the best move available as compared to other algorithms where first the move is made then the cluster cost is computed. Using this method many moves are avoided which increase the clustering cost. The first segment of the paper depicts an overview of RNSC algorithm and the Scaled-Free graph generator which is used in our comparison. The next segment describes all the parameters to be used for comparison. In the last section we have provided all the results and discussions.

## 2. AN OVERVIEW OF RNSC GRAPH CLUSTERING ALGORITHM AND THE SCALED-FREE GRAPH GENERATOR

In our approach we have used two graph clustering algorithms and two types of graph generator tools which we will be discussing in this segment.

### 2.1 RNSC

Restricted Neighborhood Search Algorithm or RNSC is a cost-based clustering technique for clustering the graph into separate clusters, where each cluster has some similar properties. The properties considered in this case are low inter-connectivity and high intra-connectivity in clusters. This is implemented only for un-weighted and undirected graphs. This algorithm applies a heuristic approach in which we only consider the moves in the restricted neighborhood of previous move, i.e., after a move has been made the next move will only be allowed in the neighbor clusters of that move. After a fixed number of moves we apply diversification moves to avoid the solution reaching local-minima in-spite of global solution. A tabu list is also maintained to avoid same moves which it has made in the recent past. This technique reduces the run-time of clustering algorithm many-folds, as it skips those redundant cases which occur multiple times and don't improve the cost of the function. The plus point of this algorithm is improvement in run-time due to the data-structure it is maintaining to update the clustering. The updating of the clustering is very fast and thus makes the algorithm fast.

### 2.2 Scaled-Free graph

It is also termed as Power-Law [3] and [6] Graph. It is also a graph generator tool but it is different from previous tool discussed. Scaled-Free [3], [4] and [6] graphs are good models for certain type of biological graph, web graphs and other naturally-occurring networks. In this graph the vertex degree follows a power-law distribution, i.e., there are large number of vertices with small degree and few vertices with very high degree (these vertices are also called hubs). The Scale free graph is represented by the notation $G_S$ (n, k), such that, $G_S$ represent the scale free graph, n is the number of vertices in the graph, k is used to construct the graph by the following method:

Algorithm: 1 Scaled-Free generator (Scaled-Free generator code) [3], [5] and [8]

These graphs are good models for certain type of biological graph, web graphs and other naturally-occurring networks. In this graph the vertex degree follows a power-law distribution, i.e., there are large number of vertices with small degree and few vertices with very high degree (these vertices are also called hubs).

The Scale free graph is represented by the notation $G_S$ (n, k), such that, $G_S$ represent the scale free graph, n is the number of vertices in the graph, k is used to construct the graph by the following method:

First take i=1, 2…k vertices and make a set out of these and call this set $G^K$. For i= k+1, k+2 … n we construct $G^{(i)}$ from $G^{(i-1)}$ by adding a vertex i to the graph and joining it to k random vertices in $G^{(i-1)}$, choosing a vertex v with probability proportional to 1+ $\deg_G{}^{(i-1)}(v)$ and not allowing multi edges. Thus $G_S$ (n, k) contains k (n-k) edges.

Because the way we attach this new vertices to old vertices, the order of the vertices with high order are further increased and the vertices with low order remain low-ordered.

## 3. SOFTWARE DESCRIPTION

Restricted Neighborhood search algorithm is implemented in C++ programming language. The project is compiled using make utility. The whole project is divided into modules, each of which is responsible for performing a particular function in the implementation. The input to the algorithm is an adjacency list representation of the graph, which is supplied by input file as a command line argument. The algorithm has many parameters which can be altered as per the need of the program. All the parameters which can be modified are stored in the "definition.h" header file. If we need to change the number of clusters required or the naïve stopping tolerance, just change the value in the definition file and it will be changed in the whole program.

The main parameters which we have put in the (definition.h) file are:

- Number of times the experiment to be performed
- Number of clusters required
- Scaled stopping tolerance
- Naïve stopping tolerance
- Tabu length
- Tabu tolerance
- Shuffling frequency
- Shuffling length
- Destructive frequency
- Amount of display information to be generated

These parameters can be modified in the definition.h and their value will change throughout the program. The number of cluster required can be set by the user if he knows the input graph and the clustering can be modified. If the input graph type is not known then set the value of num_cluster to 0 and it will initialize the initial number to the number of nodes. The number of cluster decrease as the algorithm progresses to decrease the cost of the clustering. Display parameter is set to display the amount of information to be generated and shown to the user as output. The amount of details generated is proportional to the value like:

0: minimum details only describing the final clustering and the amount of naïve and scaled steps taken.

1: It displays the above details plus the moves taken at every step in reaching the final solution.

2: It displays the above information and the clustering after every function, i.e., scaled and naïve in each experiment.

3: It displays the complete information of the experiment, including the move list at each step.

- Tabu length [5], [13], [14] and [15], sets the length of tabu list to be maintained. Tabu tolerance is the maximum number of times a node can be present in the tabu list. If a move is present in the tabu list for tabu tolerance times then that move cannot be made.

- Big cost is just a big number which is taken as infinity, so that all the values computed in this experiment are smaller than that number.

- Naive stopping tolerance sets the maximum number of moves the algorithm can take without any change in cost. After these many moves without any change in cost the naïve function stops and passes the control to the scaled function.

- Similarly the scaled stopping tolerance is the maximum number of moves the program can take if there is no change in cost. If the cost has not improved in past these many moves then the scaled function stops and passes the best clustering so far as the final clustering.

- Shuffling frequency is used as the number of moves after which the shuffling moves take place and the cluster is shuffled to avoid local-minima. Shuffling length is the number of shuffling moves it takes after shuffling frequency moves.

- Destructive frequency is the number of moves after which the destructive diversification takes place.

## 4. RESULTS AND DISCUSSIONS

### 4.1 Results for Scaled-Free Graph

This section contains all the results and discussions regarding Scaled-Free graphs.

#### *4.1.1 Number of Nodes vs Scaled Cost with varying $F_D$*

The table contains computed values of Number of Nodes versus Scaled Cost with varying $F_D$ for RNSC algorithm.

**Table 1. Dataset for Scaled-Free Graph**

| For a power-law graph of Alpha=2.5 | | | | |
|---|---|---|---|---|
| **Number of Nodes** | **Cost ($F_D$ =100)** | **Cost ($F_D$ =150)** | **Cost ($F_D$ =200)** | **Cost ($F_D$ =500)** |
| 10000 | 14155461 | 14156131 | 14156486 | 14156441 |
| 11000 | 17276774 | 17276582 | 17277236 | 17276887 |
| 12000 | 20116710 | 20115167 | 20116567 | 20115523 |
| 13000 | 23943056 | 23944700 | 23944636 | 23945804 |
| 14000 | 27679611 | 27912667 | 27678850 | 27680376 |
| 15000 | 31821169 | 31819649 | 31820922 | 31819167 |
| 16000 | 36089490 | 36088571 | 36089134 | 36092491 |
| 17000 | 40946664 | 40946711 | 40948726 | 40946114 |
| 18000 | 45267777 | 45268273 | 45266410 | 45266510 |
| 19000 | 51455782 | 51456500 | 51457340 | 51456430 |
| 20000 | 56336756 | 56337510 | 56337962 | 56336923 |

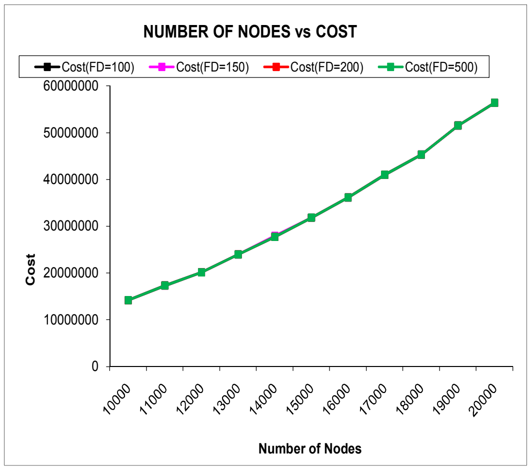

**Fig 1: A line graph representing Number of nodes vs Scaled Cost with varying $F_D$.**

**Discussion:** Fig 1 shows a line graph representing Number of Nodes versus Scaled Cost with varying $F_D$ for a graph of more than 10,000 nodes. This graph shows that there is not much change in the cost due to changing the shuffling diversification frequency. Changing the shuffling frequency changes the scaled moves as shown in the next graph, but there is not much change in the cost due to the change in the shuffling frequency. The cost is almost similar in all the cases of varying the shuffling frequency. But we can observe that as the number of nodes in the graph increases then the scaled cost of the clustering increases linearly with the number of nodes. This can be shown by the cost computation formula, which shows the contribution of each node in computing the scaled cost. According to the formula, as the number of nodes increase in the graph the corresponding scaled cost in the graph also increases. ($F_D$- Shuffling diversification frequency)

### 4.1.2 Number of Nodes vs Scaled length by varying Shuffle frequency

The table contains computed values of Number of Nodes versus Scaled Length by varying Shuffle Frequency for RNSC algorithms.

**Table 2. Dataset for Scaled-Free Graph**

| For a power-law graph of Alpha=2.5 | | | | |
|---|---|---|---|---|
| **Number of Nodes** | **Moves(Shuffling Freq.=100)** | **Moves(Shuffling Freq.=150)** | **Moves(Shuffling Freq.=200)** | **Moves(Shuffling Freq.=500)** |
| 10000 | 5352 | 2636 | 3276 | 2110 |
| 11000 | 3501 | 2704 | 2611 | 2111 |
| 12000 | 4749 | 3311 | 2658 | 3025 |
| 13000 | 5051 | 3027 | 3611 | 2359 |
| 14000 | 5420 | 3370 | 2814 | 2381 |
| 15000 | 6150 | 3305 | 3457 | 2597 |
| 16000 | 4501 | 5919 | 3217 | 2745 |
| 17000 | 5350 | 4420 | 3236 | 2756 |
| 18000 | 5000 | 3860 | 3728 | 2946 |
| 19000 | 5500 | 3839 | 3460 | 2917 |
| 20000 | 5705 | 4124 | 3672 | 3138 |

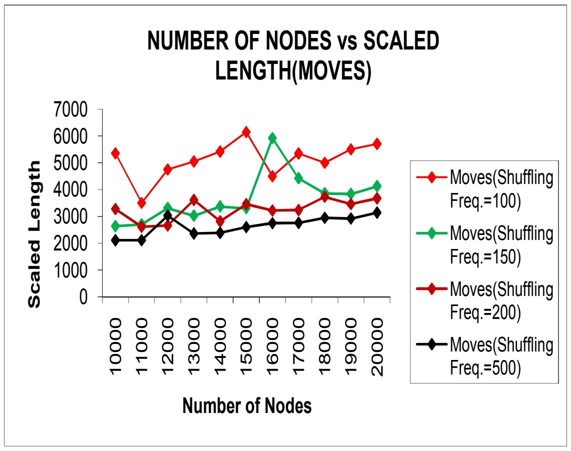

**Fig 2: A line graph representing Number of Nodes vs Scaled length by varying Shuffling frequency.**

**Discussion:** Fig 2 shows a line graph representing Number of Nodes vs Scaled length by varying Shuffling frequency for a graph of more 10,000 nodes. For small values of shuffle frequency, the scaled length of the RNSC algorithm increases. The scaled length is directly proportional to the run-time of the algorithm. If we keep the shuffling frequency small then diversification takes place after a very small number of steps, therefore it takes more steps to reach a global maxima as the diversification disturb the process after very small number of steps. On the other hand if the shuffling frequency is large then diversification takes place after a large number of moves. In that case the scaled moves is not increased to that much extent as it was due to small shuffling frequency.

### 4.1.3 Number of Nodes vs Run-time by varying Tabu Length

The table contains computed values of Number of Nodes versus Run-time by varying Tabu Length for RNSC algorithms.

**Table 3. Dataset for Scaled-Free Graph**

| For a power-law graph of Alpha=2.5 | | | | | | |
|---|---|---|---|---|---|---|
| **Number of Nodes** | **Run Time ($L_T$=50)** | **Run Time ($L_T$=100)** | **Run Time($L_T$=500)** | **Run Time($L_T$=1000)** | **Run Time($L_T$=1500)** | **Run Time ($L_T$=2000)** |
| 10000 | 0.68 | 0.71 | 0.7 | 0.85 | 0.72 | 0.68 |
| 11000 | 1.15 | 1.4 | 1.11 | 1.07 | 1.09 | 1.16 |
| 12000 | 0.94 | 1.03 | 0.99 | 0.98 | 1.12 | 1.02 |
| 13000 | 1.23 | 1.32 | 1.15 | 1.22 | 1.22 | 1.2 |
| 14000 | 1.92 | 2.02 | 2.23 | 2.17 | 2.27 | 2.21 |
| 15000 | 1.63 | 1.82 | 1.77 | 1.64 | 1.63 | 1.8 |
| 16000 | 1.61 | 1.7 | 1.65 | 1.66 | 1.66 | 1.64 |
| 17000 | 2.29 | 2.06 | 2.16 | 2.29 | 2.06 | 2.16 |
| 18000 | 4.39 | 4.8 | 4.3 | 7.12 | 5.08 | 9.5 |
| 19000 | 3.08 | 3.04 | 3.04 | 3.16 | 2.96 | 3.04 |
| 20000 | 3.38 | 2.92 | 2.88 | 3.24 | 2.79 | 2.88 |

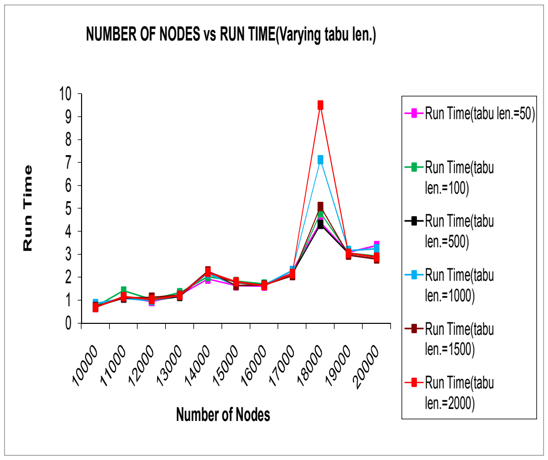

**Fig 3: A line graph representing Number of Nodes vs Run-time by varying Tabu Length.**

**Discussion:** Fig 3 shows a line graph representing Number of Nodes vs Run-time by varying Tabu Length for a graph of more than 10,000 nodes. We notice that as the tabu length decreases the run-time of the algorithm increases. Therefore large tabu length should be used to achieve less time for a particular run of the algorithm. As the tabu-length increases the number of moves which cannot be retraced increases, so there are fewer choices to make for the next move, thus the run-time decreases.

### 4.1.4 Number of clusters vs Run time

The table contains computed values of Number of clusters versus Run time for RNSC algorithms.

**Table 4. Dataset for Scaled-Free Graph**

| Number of Clusters | Run Time(100000 Nodes) | Run Time(150000 Nodes) | Run Time(200000 Nodes) |
|---|---|---|---|
| 5000 | 2.21 | 9.32 | 8.65 |
| 4500 | 1.88 | 10.68 | 10.45 |
| 4000 | 1.68 | 12.8 | 11.76 |
| 3500 | 1.41 | 15.32 | 13.28 |
| 3000 | 1.74 | 17.03 | 14.31 |
| 2500 | 2.2 | 19.99 | 15.38 |
| 2000 | 2.81 | 21.97 | 15.77 |
| 1500 | 3.22 | 23.02 | 16.93 |
| 1000 | 3.78 | 21.8 | 16.14 |
| 500 | 3.8 | 20.57 | 20.35 |

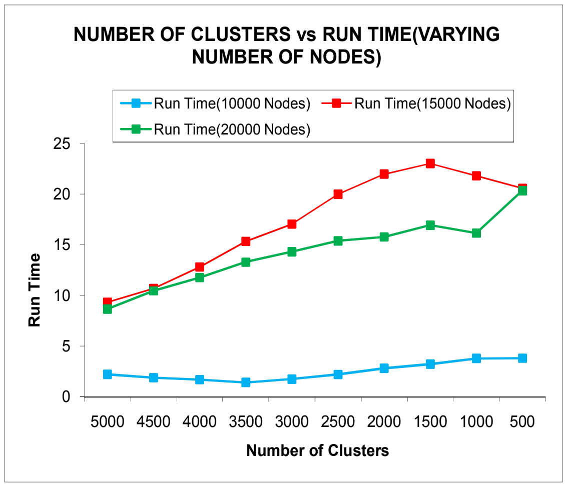

**Fig. 4: A line graph representing Number of clusters vs Run time.**

**Discussion:** Fig 4 shows a line graph representing Number of clusters vs Run time for a graph of more than 10,000 nodes. The RNSC algorithm support for predefinition of the maximum number of clusters we want in output. So we compare the performance of the algorithm by varying the number of nodes as input to the algorithm. If the number of nodes in the graph is small then the run-time will be corresponding smaller. As we decrease the maximum number of cluster the run-time increases. This is attributed due to the fact that as the maximum number of clusters decreases then there are few choices for the nodes to change cluster. The change in cost due to a move is very small and the cost keeps on oscillating in the small-range for long time. This increases the run-time of the algorithm as the max_cluster decreases. But as we increase the number of nodes in the input graph then we observe that for the graph of 2000 nodes, the run-time comes out to be slightly smaller than the run-time of graph of 1500 nodes. This can be due to the reason that the number of nodes is appropriate for the tabu list to control the number of moves to come out of some local minima, thus the run-time is decreased in that particular case.

## 5. CONCLUSIONS AND FUTURE WORK

In our approach we successfully implemented RNSC graph clustering algorithm in C++ for graphs having more than 10,000 nodes. With the help of these graphs we are able to analyze behavior of certain parameters of this algorithm. The RNSC algorithm implemented is only applicable for un-weighted and un-directed graphs. It is a local-search technique so only the neighborhood moves are considered at every step, i.e. those clusters which can be reached from current cluster by moving a single vertex. Since it is a local search technique, it doesn't have a worst-case bound on running time. The worst-case time complexity of naïve move is O (n) and that of scaled one is O ($n^2$). The number of moves is not fixed and there is no upper limit but on observing we can see that the number of moves has average time-complexity of O (n). So the total average time-complexity of the algorithm can be said to be O ($n^3$). The performance of this is also aided by the local search techniques like tabu list, diversification and multiple experiments. From the results we can conclude that the performance is not much affected by the choice of the diversification scheme selected. Setting $L_D=F_D/3$ is a costly choice, we spent twice as much time clustering as diversifying. In general, for large diversification frequency, destructive diversification will run faster than shuffling diversification. Future work will focus in optimizing the run time of RNSC algorithm by integrating it with genetic algorithm. This algorithm depicts heuristic approach which increases its run time so by integrating some portions of it by genetic algorithm we may succeed in optimizing its run time to a certain effect. This algorithm can be further extended to the weighted and directed graphs [5]. In the weighted graphs the weight can be added to the cost function and the naïve and scaled function will be slightly modified to inherit the change. The performance of the algorithm can also be improved by parallelizing the task of clustering. Single run of the experiment is difficult to divide in thread but different runs of the experiment can be threaded to produce result in lesser time. We can also apply some other higher-level parallelism to improve the run-time of experiment.

## 6. ACKNOWLEDGMENTS

The authors are very thankful to their respected Mr. Aseem Chauhan, Additional President, Amity University, Lucknow, Maj. Gen. K.K. Ohri, AVSM (Retd.), Director General, Amity University, Lucknow, India, for providing excellent computation facilities in the University campus. Authors also pay their regards to Prof. S.T.H. Abidi, Director and Brig. U.K. Chopra, Deputy Director, Amity School of Engineering, Amity University, Lucknow for giving their moral support and help to carry out this research work. I am also thankful to Mr. Andrew Douglas King, author of RNSC algorithm, for his support for understanding the concepts used in the design and implementation of our project. His report was very helpful in understanding the algorithm. Lastly we thank all of our colleagues who directly or indirectly supported us.

## 7. REFERENCES

[1] Sauta Elisa Schaeffer, “Survey Graph clustering,” Elsevier Computer Science Review, vol. I, pp. 27-64, 2007.

[2] P. Erdos and A. Renyi. On the evolution of random graphs. Publ. Math. Inst. Hungar. Acad. Sci., 5:17-61, 1960.

[3] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. Science 286(5439) (1999) 509-512.

[4] A.-L. Barabasi and Z. N. Oltvai. Network biology: Understanding the cell’s functional organization. Nature Reviews Genetics, 5:101-113, 2004.

[5] A.D. King, Graph clustering with restricted neighbourhood search. Master’s Thesis, University of Toronto, 2004.

[6] X. Hu and J. Han. Discovering clusters from large scale-free network graph. In ACM SIG KDD Second Workshop on Fractals, Power Laws and Other Next Generation Data Mining Tools, August 2003.

[7] S. Enright, A.j.van Dongen, C.A. Ouzounis, An efficient algorithm for large-scale detection of protein families, Nucleic Acids Res. 30(7) (2002) 1575-1584.

[8] Scaled-Free graph generator code. [Online].Available: http://www-rp.lip6.fr/~latapy/FV/

[9] King, A. D., Przulj, N., and Jurisica, I. (2004) Bioinformatics 20, 3013-20.

[10] King, A. D. (2005), McGill University, Montreal.

[11] C. Avanthay, A. Hertz and N. Zuerey. A variable neighbourhood search for graph coloring. European Journal of Operational Research, 151:379 388, 2003.

[12] J. U. Brandes, M. Gaertler and D. Wagner. Experiments on graph clustering algorithms. In Proc. 11th Europ. Symp. Algorithms (ESA'03), Lecture Notes in Computer Science, volume2832, pages568-579. Springer-Verlag, 2003.

[13] F. Glover. Tabu search, part I. ORSA Journal on Computing, 1(3):190-206, summer 1989. “ORSA" is called Informs today.

[14] F. Glover. Tabu search, part II. ORSA Journal on Computing, 2(1): 4-32, Winter 1990. “ORSA" is called Informs today.

[15] A. Hertz and D. de Werra. Using tabu search techniques for graph colouring. Computing, 39(4), 1987.

[16] Unix Make utility, online documentation: http://www.gnu.org/software/make/manual/.