Genetic Clustering Algorithms for Detecting Money-Laundering

Lluís Alsedà[1], Abhishek Awasthi[2⋆], and Jörg Lässig[2]

[1] Department of Mathematics, Autonomous University of Barcelona, Spain
alseda@mat.uab.cat
[2] Department of Electrical Engineering and Computer Science ,
University of Applied Sciences Zittau/Görlitz, Germany
{aawasthi,jlaessig}@hszg.de

Abstract. Genetic Algorithms are one of the most applied class of algorithms for solving global/multi-modal optimization problems and have been extensively studied for solving NP-hard optimization problems. This work presents the development of genetic algorithms for detecting money-laundering by finding the clusters in a graph, constructed using financial and customer data. The developed algorithm can be applied in other related areas as well. We present two algorithms based on Genetic Algorithms for (i) detecting all the clusters in a graph and (ii) detecting the cluster of any given node.

Keywords: Genetic Algorithms, Money-Laundering, Graph Clustering, Modularity, Island Model

1 Introduction

Money-laundering is the practice of engaging in financial transactions to conceal the identity, source or destination of illegal money. It is one major concern for many countries and international organizations like I.M.F., U.N. and the World Bank. This work presents one way to detect money-laundering by detecting and studying well connected communities in the network of financial and telecommunication data. Basic work-flow steps for this approach are (i) to collect the relevant financial data, (ii) to construct a graph of the data representing financial and telecommunication transfers between the entities and (iii) to find the clusters of entities which are suspicious. In money-laundering it is sometimes unnecessary to find all the clusters in the financial graph. In such a situation, finding all the clusters in a huge graph could be an unnecessary and time consuming task and one may be only interested in finding the cluster of one particular node only. Moreover, Financial Intelligence Units (FIUs) usually have knowledge of one or more suspicious entities which cause money-laundering or some financial fraud. Hence, it makes perfect sense to find the cluster of the

⋆ The work presented here is the MathMods Master Thesis of the second author, made under the supervision of the first author, at the Universitat Autònoma de Barcelona.

suspicious node (entity) instead of finding all the clusters in the network. All the standard and recent developments in clustering algorithms focus on finding all the clusters in the graph [1, 2, 5, 6]. Not much research has been carried out in finding the cluster of just one desired node. Almost all of the metrics which are available for evaluating the quality of the clusters work only if one is aware of all the clusters in the graph. Modularity for instance is one such measure for a graph which takes into account all the clusters present in a graph. Nonetheless, we show that modularity can be used in a different manner to find the cluster of any given node. This work presents genetic clustering algorithms which can be used for (i) detecting all the clusters in a graph and (ii) finding the cluster of any given node. Our approach for both clustering problems also incorporates the concept of an island model with migration, making the algorithm more efficient than a single population GA.

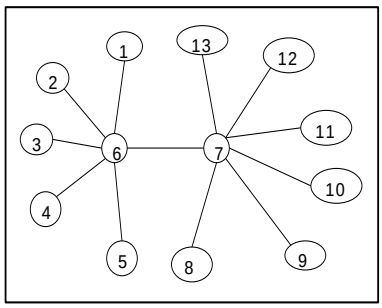
2 Related Work

Graph clustering is an efficient methodology to detect groups of entities in a network or graph, possessing certain similar characteristics. Many networks of interest in the sciences, including social networks, computer networks, and metabolic & regulatory networks are found to divide naturally into communities or modules. The problem of detecting and characterizing this community structure is one of the outstanding issues in the study of networked systems. Several techniques and algorithms have been proposed for community detection such as k-means [1], hierarchical clustering [2], evolutionary algorithms [9–11, 14] and other algorithms of different nature [3, 6, 7]. One highly effective approach for cluster detection is the optimization of the quality function known as modularity over the possible divisions of a network. Newman and Girvan highlighted the relevance of the community structure in complex networks and proposed a method to detect it [4]. This work opened a new scenario that has attracted a great deal of attention in recent years. Modularity is one of the most recent and important advancement in community detection that allows quantification of the modular structure of a network. Given a network represented as weighted graph and partitioned into communities or modules; C_i being the community to which node i is assigned, the mathematical definition of modularity is expressed in terms of w_{ij}, which represents the value of the weight in the link between the nodes i & j (0 if no link exists), the strengths $w_i = \sum_j w_{ij}$ and $w = \frac{1}{2}\sum_i w_i$, as:

$$Q = \frac{1}{2w}\sum_i \sum_j (w_{ij} - \frac{w_i w_j}{2w})\delta(C_i, C_j), \tag{1}$$

where the kronecker delta function $\delta(C_i, C_j)$ takes the value 1 if the nodes i and j are in the same module and 0 otherwise. For unweighted networks, w_i becomes the degree of node i, and w the total number of links of the network [8]. The modularity of a given partition is the probability of having edges falling within modules in the network minus the expected probability in an equivalent

(null case) network with the same number of nodes and edges placed at random preserving the nodes strength. The larger the modularity value the better is the partitioning of the network into communities. In fact, modularity gives a quantitative definition of the notion of clusters in a graph (modularity-clusters). In this paper "clusters" will mean the divisions of a graph which are obtained by maximizing modularity. As a small example, Figure 1 illustrates the modularity of a graph with different possible clusters. The modularity of the graph clustering is highest if the nodes are divided into two separate clusters: {1,2,3,4,5,6} and {7,8,9,10,11,12,13}.



(a)

Clusters	Modularity
{1,12,13,5,6,7}; {2,3,4,8,9,10,11}	-0.1701
{2,4,6,8}; {1,3,5,7,9,10,11,12,13}	0.0521
{1,2,3,4,5,6,7,8}; {9,10,11,12,13}	-0.0868
{1,2,3,4,5,6,7}; {8,9,10,11,12,13}	-0.1250
{1,2,3,4,5,6}; {7,8,9,10,11,12,13}	0.4132
{8,3,7,5,12,2}; {1,4,6,9,10,11,13}	-0.1667
{3,6,2,7,1,9,4,5}; {8,10,11,12,13}	-0.0868

(b)

Fig. 1. Modularity values for several divisions.

Modularity optimization is a very effective method to detect communities, both in real and in artificially generated networks. However, modularity itself has not yet been thoroughly investigated, and only a few general properties are known. Recently, Fortunato and Barthélemy presented critical analysis community detection [12]. They showed that modularity contains an intrinsic scale that depends on the total number of links in the network. Modules that are smaller than this scale may not be resolved, even in the extreme case where they consist of complete graphs as subgraphs connected by single bridges. The resolution limit of modularity actually depends on the degree of interconnectedness between pairs of communities and can reach values of the order of the size of the whole network. The problem of the resolution limit of modularity arises if in a huge graph there are communities of very small sizes. In that case modularity might not recognize all the smaller clusters and might club them with the clusters of bigger size. An examplary case could be a graph having 1000 nodes with 10 clusters of 90 nodes each and 2 clusters of 5 nodes each, as optimal clustering. In such a case modularity optimization could end up in clubbing the smaller clusters with the larger ones. However, the resolution limit of modularity is not accounted for in this work and needs further investigation.

3 Proposed Genetic Clustering Algorithm (GAC)

The genetic clustering algorithm which we propose takes the graph data, $G = (V, E)$; the population size, $PopSize$; and an upper bound for the number of clusters, NC_{UB}; as inputs and returns the best fit clusters present in the graph, as per the modularity optimization. In our algorithm, we have taken the upper bound value for the number of clusters as $\left\lceil \sqrt{N} \right\rceil$, where N is the number of nodes in the graph (see [15]) and $\lceil \cdot \rceil$ denotes the ceiling function. The algorithm is based on the evolution of the solutions. To improve the efficiency and accuracy of the algorithm the island model of the genetic algorithm has been used, with migration every specific number of generations [13]. In the next sections we describe the population initialization, fitness function, selection methodology, methodologies for all the genetic operations implemented and the implementation of the island model.

Algorithm 1: Genetic Clustering Algorithm (GAC).

Data: $G = (V, E)$, $PopSize$ and NC_{UB}
Result: Clusters in the graph
while *modularity value not stabilized* **do**
  crossover & mutation in each island ;
  calculate modularity ;
  elitist selection in each island ;
  migrate best individual to all other islands, after every τ iterations ;
end

3.1 Population Initialization

The clustering problem can be solved using a GA by either the binary or decimal representation of the population. But considering the fact that the size of the graph could be really huge, and from our own experiments with both kinds of population representation, the decimal representation of the population seems to yield the results more efficiently. A decimal number for a node indicates to which cluster the node belongs. Let us say, we have a graph with 15 nodes, then one may represent an individual of the population as shown in Table 1. Hence, each individual in the population would be a vector of size n where n is the total number of nodes in the graph and each entry of the vector indicates which cluster the nodes belong to. The population size in both algorithms has been taken as approximately 1.5 times the number nodes in the graph.

Table 1. Chromosomes of a single individual in the population.

Nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Chromosomes	1	2	3	1	1	1	1	2	2	2	2	3	3	3	3

3.2 Fitness Function

The fitness function is one of the corner stones of a genetic algorithm. It is used to identify and select the best fit solutions (individuals) in the entire population. For the clustering problem, we use modularity as fitness function. As stated in the earlier section, modularity is a well accepted criterion and measure to evaluate the quality of graph clustering. Moreover, as it has been said before its maximization defines the clusters of a graph in this paper.

3.3 Selection

The next step is to select the individuals in the population for mating and producing new offspring. There are several methods available for selection; however for our clustering problem we have adopted the elitist selection methodology, keeping in mind that the best fit individuals are more prone to produce better offspring. Let us say, the population size is $PopSize$, then the best individual (highest modularity value) moves to the next generation without any genetic operations (to avoid the loss of better fit individual of the population) and the remaining $PopSize - 1$ individuals move to the mating pool and undergo crossover and mutation. Likewise, each island undergoes this elitist selection procedure. The mating pool is created by coupling each individual with its adjacent individual, as per the ranking (e.g. individuals (2 and 3), (4 and 5) and so on). $PopSize$ has to be an odd number so as to pair up the individuals in the mating pool.

3.4 Crossover

Crossover is one of the two standard genetic operations during which certain genes of the two mating parent-individuals get swapped forming two new individuals with mixed characteristics. Crossover is an essential operation if one wants to combine characteristics of different individuals. Two types of crossover techniques that have been applied here are explained below. Each island implements only of these crossover operations as shown in Table 2.

- 2-Point Sectional Crossover (CdecS): The two mating individuals are divided in almost three equal sections (using the *ceil* or *floor* functions) and the first and last sections of each of the two mating individuals are swapped with each other forming two new individuals with mixed characteristics of the parent genes. Figure 2 shows two selected sections of the mating individuals and their flipping.

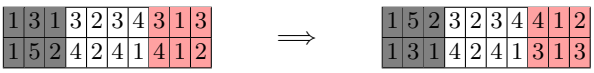


Fig. 2. 2-Point sectional crossover (CdecS).

- Uniform Crossover (CdecU): In this operation, each gene of the mating individuals is swapped or retained with an equal probability of 0.5. A set of N (number of nodes in the graph) random numbers are generated corresponding to each gene of the pair of chromosomes. If the random number for a gene is less than 0.5 then they are swapped otherwise they are retained as it is. Figure 3 illustrates this operation with 3 genes flipped.

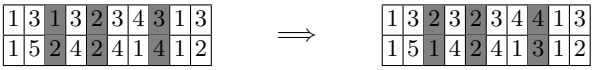


Fig. 3. Uniform crossover (CdecU).

3.5 Mutation

Mutation is an essential genetic operation particularly to gain more diversity and be explorative. As the algorithm by itself decides and searches for the optimal number of clusters, it should be able to incorporate certain changes crucial for graph clustering. It might happen that at a certain point in time the solution may contain less than the optimal number of clusters. Hence, the algorithm should be able to add a new cluster as shown by the green box in Figure 4. Likewise, another possibility is that the solution may contain more than the optimal number of clusters, so the algorithm should be able to delete clusters as shown by the red box in Figure 4. Another likely possibility is the wrong assignment of nodes to clusters. One may need to switch one (or more) node(s) from one cluster to another to optimize the solution, as shown by grey boxes in Figure 4. To overcome the above three issues, two kinds of mutation operations

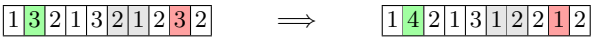


Fig. 4. Expected mutation operations.

have been implemented in this algorithm. One operation is MdecA which is implemented be replacing each gene by randomly choosing an integer between $MaxCN+1$ and NC_{UB}, where $MaxCN$ is the maximum cluster number present in the population and NC_{UB} is the upper bound for the number of clusters. Operation MdecA ensures the possibility of addition of a cluster as it adds an extra cluster to the gene(s). Another mutation operation which is considered is MdecB where the gene(s) are replaced by a random integer between 1 and $MaxCN$ but not the gene value itself. As MdecB replaces the gene(s) with a new integer between 1 and $MaxCN$, it takes into account the rearrangement of clusters as well as deletion of the clusters. Each gene of the whole population in an island undergoes one of the operations MdecA or MdecB with probabilities of 0.007 and 0.03, respectively, since those probabilities lead to competitive results.

The reason for implementing MdecA with a low probability is because of the fact that the initial genes in each island consists of clusters are chosen uniformly at random between 1 and $MaxCN$, where $MaxCN$ is the upper bound for the optimal number of clusters. Hence, the addition of a new cluster should come with a low probability. Table 2 shows the specific combinations of crossover and mutation operations implemented in each island.

Table 2. Crossover and mutation operations in each island.

# Island	1	2	3	4	5	6
Crossover	CdecU	CdecS	CdecU	CdecS	CdecU	CdecS
Mutation	MdecA	MdecB	MdecA	MdecB	MdecA	MdecB

3.6 Island Model & Migration

The island model is one class of parallel genetic algorithms and it typically runs serial GA's on each of several connected processors. The algorithms at the different islands are usually identical, but run independently to the other islands. A difference could be in the genetic operations or the selection procedures carried out at each island. Each island usually starts with a different population chosen uniformly at random and some individuals are transmitted between the islands periodically (after certain number of τ generations) in a process called migration. In our algorithm we propose 6 equal sized islands with different combinations of crossover and mutation, shown in Table 2. After τ iterations, the best individual of each island is migrated (retaining in its own island) to all other islands to replace the worst individual in every other island, according to its fitness ranking. We have implemented migration with τ as 10. The concept of the island model and migration is a very efficient and accurate enhancement of the genetic algorithm. With the advancement of multi-core CPUs this methodology can considerably increase the speed of the algorithm by using different cores of the processor for different islands. This way we try to make the algorithm a lot more efficient to attain a good solution. The number of islands could vary as per the requirement and the presence of multi-core CPUs. However, we implemented the algorithms one a single-core CPU and it seems to fetch good and fast results with 6 islands. Another variation is the migration strategy. One may device a number of migration strategies and experiment for the best results. We tried with two different kinds of migration strategies in our experiments. One as described above (Figure 5), and the second strategy was to migrate only the best individual among all the islands to all the other islands, after every τ iterations. But we found that the second strategy does not converge to the optimum or better solution as fast as the first one.

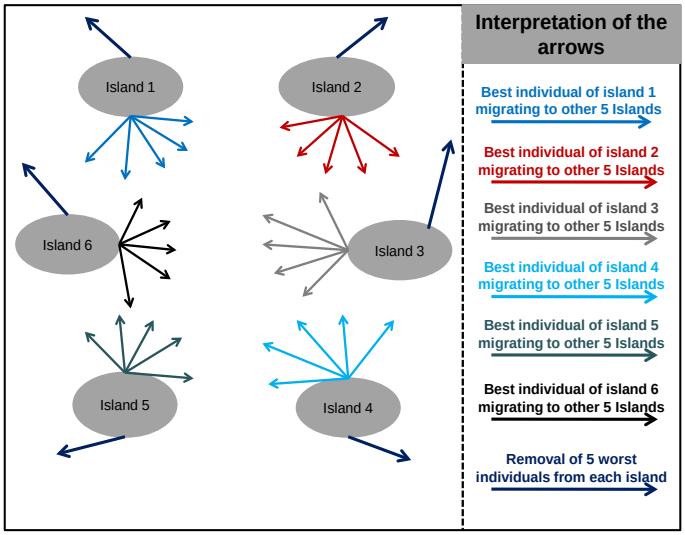


Fig. 5. Graphical explanation: island Model & migration.

4 Genetic Algorithm: Cluster of a Node (GACN)

Not much research has been carried out on finding the cluster of one desired node. However, we show that modularity can again be exploited to break up the graph and search for the desired cluster. The main idea behind the proposed algorithm is to iteratively divide the graph in two best fit union of clusters at every step, discard the group of clusters which does not contain the desired node and repeat this process in the remaining subgraph. This process is continued until the cluster which contains the desired node cannot be bisected further to yield a better modularity value. A small example in Figure 6 explains the methodology behind the algorithm. The graph in Figure 6(a) contains 3 dense subgraphs and we wish to find the cluster of the node marked with a star. At the first iteration, a variant of the genetic clustering algorithm (Algorithm 1, adapted to deal with only two groups of nodes) is applied to divide the graph in two best fit clusters, Figure 6(b). The cluster which does not contain the star node is discarded and in the next iteration the remaining graph is divided again in two best fit clusters, Figure 6(c). This procedure is iterated until the desired cluster cannot be divided any more, i.e., the modularity of the graph does not improve and finally we are left with the desired cluster as shown in Figure 6(d). The crucial aspect of this algorithm is the stopping criterion. Following the above procedure, we definitely know that the algorithm divides the graph in two groups of clusters since modularity defines the clusters for us. The fact that we maximize modularity when dividing the graph in two groups, tells us that each group is a union of modularity-cluster(s). Let us say, the proposed GACN algorithm is run for some number of iterations and ultimately we are left

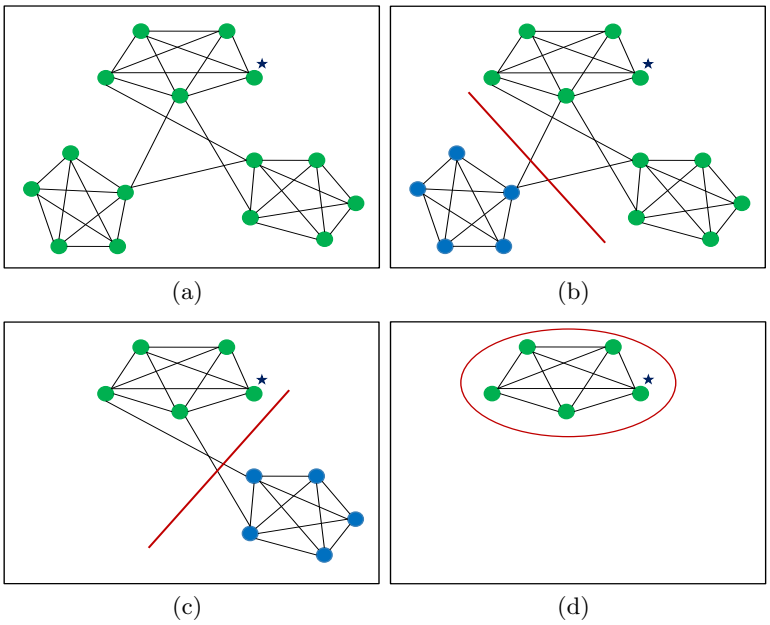


Fig. 6. Graphical explanation of GACN.

with a graph having just one cluster with the desired node inside it. As per the algorithm, when this graph (with just one cluster i.e., the whole graph) is divided in two clusters, the modularity value of such a division would be less than the modularity value of the whole graph as one single cluster. In such a scenario the genetic algorithm will search for the best fit solution and this solution would be nothing but the whole graph as a single cluster. Moreover, modularity for the whole graph is always zero; hence, this is a well defined stopping criterion for the proposed genetic algorithm for detection of the cluster of one node. The pseudo code for finding the cluster of a desired node is given in Algorithm 2. The algorithm takes the data for the graph & the desired node of which the cluster is sought as input and returns the cluster in which the desired node lies. The Algorithm 2 uses a variant of Algorithm 1 as subroutine for detecting clusters in the network. The only difference is that this time we implement GAC to yield the maximum modularity while bisecting the graph in just 2 clusters, hence some genetic operations have been eliminated or changed to preserve the "two groups of clusters" structure. When the size of the graph increases the complexity of GAC increases much faster than GACN which performs some number of iterations of a less complex version of GAC (since it only has to divide the graph into two groups). Therefore GACN is expected to perform better than GAC in determining the cluster of a node.

Algorithm 2: Genetic Algorithm: Cluster of a node (GACN).

```
Data: G = (V, E), PopSize & N_d
Result: The Cluster of Node N_d
while Modularity ≠ 0 do
    run GAC on G for 2 clusters ;
    calculate modularity (Q) ;
    if (Q==0) then
        return G ;
    else
        G = G-{Cluster without N_d} ;
    end
end
```

4.1 Algorithm Specifications

The initial population representation is slightly different than in Algorithm 1 as here each node is assigned either 1 or 2, with an equal probability of 0.5 each. This way, at every new generation the graph is divided in only two clusters and ultimately yields two communities in the graph with the highest modularity value. Modularity is again the fitness function. The modularity divides the graph into two clusters (union of modularity-clusters). The selection procedure is the same as in the previous algorithm (GAC) implementing the elitist selection methodology in each island, explained in Section 3.3. However, due to the less complex nature of GACN, we consider only one crossover and one mutation operation. All the individuals in the mating pool of the islands undergo these two genetic operations. In this algorithm a one point crossover (Cdec1) is implemented wherein a point is randomly selected from the entire length of the chromosome and the two sections of the genes are swapped between the mating individuals (Figure 7). The mutation operation (Mdec1) in this algorithm flips each gene of the population from 1 to 2 (or vice-versa) with a probability of 0.02. Each individual in the mating pool of all the islands undergoes these two operations (Cdec1 & Mdec1). Apart from these variations, we consider 4 islands with the same population sizes and the same number (τ) of generations between migration events (for $\tau = 10$).

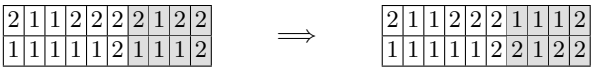


Fig. 7. 1-Point sectional crossover (Cdec1).

5 Results & Conclusion

This paper presents two algorithms to find (i) all the clusters in a graph and (ii) the cluster of any desired node in a graph using genetic algorithms. The first

Table 3. Results of GAC and GACN (runtime ± standard deviation).

#	Number of Nodes	Number of Clusters	Run Time (GAC) (seconds)	Run Time (GACN) (seconds)
1	8	2	0.06±0.033	0.55±0.035
2	11	2	0.15±0.056	0.54±0.029
3	15	3	0.24±0.066	0.79±0.086
4	21	3	0.74±0.223	2.07±0.409
5	25	4	1.27±0.305	2.40±0.025
6	29	4	1.77±0.419	2.40±0.126
7	42	5	5.62±1.538	3.53±0.307
8	61	4	12.53±3.161	4.37±0.104
9	81	4	24.08±5.737	5.50±0.279
10	101	5	34.97±9.487	6.96±0.541
11	201	10	168.23±32.315	18.98±0.565
12	301	15	430.90±110.5	25.53±0.508
13	401	10	643.23±170.032	37.61±0.625
14	619	10	967.43±270.735	82.77±1.685

algorithm (GAC) is an enhanced genetic clustering algorithm, which implements the island model for genetic algorithms with migration and provides all the clusters in the graph, supposedly better than the previously proposed GA based clustering algorithms [9–11, 14]. The second manifestation of this work is a novel algorithm (GACN) to find the cluster of any specific node in a network. The implementation of the algorithms was carried out in MATLAB on a Windows machine with 1.5GB RAM, 1.73GHz dual core processor. We ran both, the algorithms, on several graphs of various size and number of clusters, Table 3. Figure 8 shows the error plot of the logarithmic runtimes for both the algorithms. We determined that for graphs of over 30 nodes, the Algorithm 2 for finding the desired cluster runs much faster than the Algorithm 1, which is shown in Table 3. The Algorithm 2 is extremely useful especially in money-laundering, wherein one is usually aware of some suspicious entities. This algorithm can be efficiently used for money-laundering detection, where we are interested in finding the community of a suspicious entity. The algorithms have been applied to a range of examples and have proven to be robust and efficient.

Acknowledgements We would like to thank AIA (Aplicaciones en Informática Avanzada, S.A.) for providing us the environment and the financial data to implement the algorithms for money-laundering detection. We are also thankful to CRM (Centre de Recerca Matemàtica) for sponsoring this work.

References

1. MacQueen, J.B.: Some Methods for Classification and Analysis of Multivariate Observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics

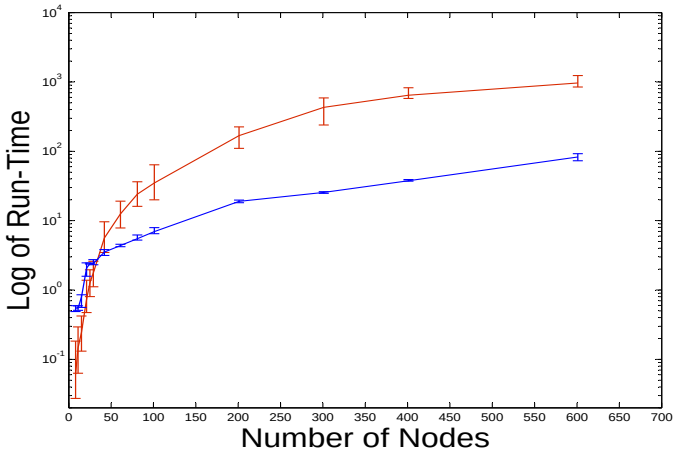


Fig. 8. Error-plot of the logarithmic run-time; GAC (red), GACN (blue).

and Probability, University of California Press, pp. 281-297, (1967).
2. Johnson, S.C.: Hierarchical Clustering Schemes. Psychometrika 2, 241254 (1967).
3. Newman, M.E.J., Girvan, M.: Community Structure in Social and Biological Networks. In: Proceedings of the National Academy of Sciences 99(12), pp. 7821–7826 (2002).
4. Newman, M.E.J., Girvan, M.: Finding and Evaluating Community Structure in Networks. Physical Review E 69:026113 (2004).
5. Newman, M.E.J.: Modularity and Community structure in Networks. In: Proceedings of the National Academy of Sciences 103(23), pp. 8577-8582 (2006).
6. Huang. J., Sun, H., Han, J., Deng, H., Sun, Y., Liu, Y.: SHRINK: A Structural Algorithm for Detecting Hierarchical Communities in Networks. ACM 978-1-4503-0099, (2010).
7. Schaeffer, S.E.: Survey: Graph Clustering, Computer Science Review 1, 27-64 (2007).
8. Arenas, A., Fernandez, A., Gomez, S.: Analysis of the Structure of Complex Networks at Different Resolution Levels. New Journal of Physics 10:053039 (2008).
9. Lipczak, M., Milios, E.: Agglomerative Genetic Algorithm for Clustering in Social Networks. ACM 978-1-60558-325, (2009).
10. Lorena, L.A.N., Furtado, J. C.: Constructive Genetic Algorithm for Clustering Problems. Evolutionary Computation 9(3), 309-328 (2001).
11. Lin, H.J., Yan, F.W., Kao, Y.T.; An Efficient GA-based Clustering Technique. Tamkang Journal of Science and Engineering 8(2), 113–122 (2005).
12. Fortunato, S., Barthelemy, M.: Resolution Limit in Community Detection. In: Proceedings of the National Academy of Science 104, pp. 36–41, (2007).
13. Gordon, V. S., Whitley, D.: Serial and Parallel Genetic Algorithms as Function Optimizers. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 177–183, (1993).
14. Chiou, Y.C., Lan, L.W.: Theory and Methodology - Genetic Clustering Algorithms. European Journal of Operational Research 135, 413–427 (2001).
15. Yu, J.,Cheng,Q.: The Upper Bound of the Optimal Number of Clusters in Fuzzy Clustering. Science in China Series: Information Sciences 44, 119–125 (2001).