COMMUNITY DETECTION USING ANT COLONY OPTIMIZATION TECHNIQUES

Sercan Sadi[1], Şima Etaner-Uyar[2], Şule Gündüz-Öğüdücü[2]

[1]Informatics Institute
Department of Computer Sciences
[2]Electrical and Electronics Faculty
Department of Computer Engineering
Istanbul Technical University
Maslak 34469 Istanbul, Turkey
{sadis, etaner, sgunduz}@itu.edu.tr

Abstract: *Parallel to the continuous growth of the Internet, which allows people to share and collaborate more, social networks have become more attractive as a research topic in many different disciplines. Community structures are established upon interactions between people. Detection of these communities has become a popular topic in computer science. Currently, community detection is commonly performed using Social Network Analysis (SNA) algorithms based on clustering. The main disadvantage of these methods is their high computational costs and non-scalability on large-scale social networks. Our main aim is to reduce these computational costs without loss on solution quality. In this study, we focus on Ant Colony Optimization techniques to find cliques in the network and assign these cliques as nodes in a reduced graph to use with SNA algorithms.*

Keywords: *Social Networks, Ant Colony Optimization, Clique Problem, Community Detection, Social Network Analysis, Clustering*

1. Introduction

Parallel to the continuous growth of the Internet, which allows people to share and collaborate more, social networks have become more attractive as a research topic in many different disciplines. Community structures are established upon interactions between people. Detection of these communities has become a popular topic in computer science. There are many definitions of communities in various disciplines. Having small differences in the terminology, communities can be explained by two major definitions: In biological terms, organisms that share an environment and have intersecting demands, intentions and risks are called communities. In sociology, a group of people having the same organizational structure, same cause or same intentions on a shared location is called a community. With the advent of the Internet, such terminology for communities may be updated to have geographical (physical) or virtual locations shared. Such social networks that form communities can also be found online with the significant increase on collaborating and sharing ideas through the popularity of the WWW. From the computer science point of view, members of a network are presented as nodes of a graph with edges in between, if there is relevance or interaction between the two members. Based on this, communities can be defined as a group of nodes that have higher density of edges in between, compared to edges between them and other node groups [1]. Community detection is an important topic for many disciplines,

Girvan and Newman proposed a term on community detection called *modularity* [2]. Many community detection algorithms based on this definition are proposed. Most of these techniques use clustering for community detection. Social Network Analysis (SNA) methods that are based on community detection through clustering, suffer from high computational costs and non-scalability on large-scale social networks. Our main aim in this study is to reduce these computational costs while maintaining an acceptable level of solution quality; thus decreasing the execution times and increasing scalability on such networks. In our approach, we used the concept of a *clique* in graph theory, to define community structures. In a graph, a clique is a subgraph which is complete, i.e. there is an edge between all possible node pairs. We then, used Ant Colony Optimization (ACO) techniques to search for cliques on a given network graph. We used a modified version of a maximum clique search algorithm to find cliques of all possible sizes in the given graph. Then, the graph is transformed into a smaller scale graph, where the cliques are taken as meta-nodes. We will refer to these meta-nodes as *clique-nodes*. New edges are formed between the clique-nodes, based on the connections between the individual nodes belonging to each clique. Finally, we use traditional SNA methods to find community memberships on the reduced graph.

2. Problem Definition and Related Work

Based on many definitions of community detection in literature, the problem can be formulated for social networks as follows: Assume that a social network is modeled as a graph *G=<V, E>*, where *V* represents the vertices corresponding to individuals in the network and *E* represents the edges which show the pairwise connections between the individuals (i.e. similarities or relevancies between two individuals). Referring to the previous definition of a community in the previous section, a community in a network graph is a subgraph that has a higher density of edges in between its members and a lower density of edges from its members to those outside the subgraph. As a common approach, network graphs can be represented by an *NxN* adjacency matrix where *N* is the number of nodes/vertices in the corresponding graph. For unweighted graphs, an adjacency matrix cell holds a value of 1, if there is an edge (i.e. an interaction) between the corresponding two vertices, and 0 if there is none. If the graph has weighted edges that represent the amount of similarity between two vertices, then the corresponding cell in the adjacency matrix takes on real values.

The problem is to find *k* number of communities in a given network graph, such that each community satisfies Eq. (1):

$$\sum_{i \in K} \sum_{j \in K} a_{ij} > \sum_{k \in K} \sum_{l \notin K} b_{kl}, b_{kl} = b_{lk}, (i, j, k, l) \in [1, N] \qquad (1)$$

where a_{ij} is the similarity value of an edge in the community *K* and b_{kl} is the similarity value of an edge to the outside of that community *K*.

There are many proposed Social Network Analysis (SNA) methods for community detection based on bisection and clustering, where a community is defined as a cohesive group, such as cliques, clans or plexes. A hierarchical clustering method proposed by Donetti and Munoz [3], to find larger communities in a given graph, uses Laplacian eigenvectors as a similarity measurement; communities are created after a division operation and vectors are re-calculated to continue division. Although this method does not require the number of communities to be defined initially, the termination condition of the process can not be optimized for the best clustering on the network.

Network modularity is a term introduced by Girvan and Newman [2] [4] for a divisive approach which is based on eliminating edges from the graph depending on their betweenness values. The betweenness used here is based on *edge betweenness* where weights are assigned to edges which are stationed on the shortest path between pairs of nodes. As the number of shortest paths which go through an edge increases, the betweenness value of that edge increases. The network modularity Q is defined as the ratio of in-community edges to the edges of a randomly created subgraph on a given network. Q can take on values between 0 and 1; the value approaches 1 if the community has less connections to the outside compared to the edges in the group. While the Q value is optimized to find a better division on the given graph, the proposed algorithm shows performance loss on large-scale networks. Radicchi proposed a similar and better edge-clustering approach on the given method [5]. Clauset, Newman and Moore [6] have also proposed a fast greedy clustering algorithm that uses modularity maximization. Clustering continues by merging nodes that have the maximum [ΔQ] until the difference becomes negative. Although Wakita and Tsurumi [7] came up with an optimized version of this method, there are still concerns on performance and solution quality for large-scaled graphs.

Along with the above SNA methods, nature-inspired approaches are also used in community detection. A genetic algorithm which is based on a fitness function for diversification of node groups that match a community definition was proposed by Pizzuti [8]. Ant clustering is another evolutionary technique used on community detection. An example of this technique is used in the study of Liu et al. for Enron's mail network communities [9]. In our study, we also used the ACO technique as a nature-inspired approach. However, unlike in [9], ACO is not used for clustering. We use ACO to determine cliques which will then be used as vertices in a reduced graph. A regular clustering based SNA algorithm can then be applied on this reduced graph. By doing this, we aim to overcome the performance loss of SNA methods on large-scale networks.

3. Ant Colony Optimization

ACO, one of the most commonly used swarm intelligence techniques in literature, is based on the behavior of real ants. ACO was first introduced by Marco Dorigo in his PhD thesis [10]. In the real world, ants (initially) wander randomly, and upon finding food return to their colony, while laying down a special chemical called the *pheromone*. This is used to communicate with other ants. If other ants come across a path with pheromones on it, they are likely to follow the trail, returning and reinforcing it if they also find food along the same path. The basic ACO algorithm is given in Algorithm (1). An ACO iteration consists of the solution construction and pheromone update stages. In each iteration, each ant in the colony constructs a complete solution. Ants start from random nodes and move on the construction graph by visiting neighboring nodes at each step.

Algorithm 1: Basic ACO Outline

```
1:  set ACO parameters
2:  initialize pheromone levels
3:  while stopping criteria not met do
4:      for each ant k do
5:          select random initial node
6:          repeat
7:              select next node based on decision policy
8:          until complete solution achieved
9:      end for
10:     update pheromone levels
11: end while
```

An ant *k* chooses the best neighbor with a probability of q_0. Otherwise, the next visited node is determined using a stochastic local decision policy based on the current pheromone levels τ_{ij} and heuristic information η_{ij} between the current node and its neighbors with a probability p^k_{ij} as calculated in Eq. (2).

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, j \in N_i^k \tag{2}$$

Pheromone trails are modified when all ants have constructed a solution. First, the pheromone values are evaporated by a constant factor on all edges. Then, the pheromone values are increased on the edges the ants have visited during their solution construction. Pheromone evaporation and update are implemented as given in Eq. (3) and Eq. (4), where $0 < \rho \leq 1$ and $\Delta\tau^k_{ij}$ is the amount of pheromone deposited by ant *k*.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \tag{3}$$

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{4}$$

Ant System (AS) is the first implementation of ACO algorithms, and has been the basis for many ACO variants. There are many successful AS variants in literature. Among the most commonly used variants, the elitist AS, rank-based AS, the MAX-MIN AS (MMAS), the ant colony system (ACS), the best-worst AS, the approximate nondeterministic tree search, and the hyper-cube framework can be mentioned [11]. MMAS and ACS are shown to be good both in solution quality and also in solution speed for the example cases in [11]. Therefore, we also use them in this study. MMAS and ACS are among the approaches which can be considered as direct variants of AS, since they both use the basic AS framework. The main differences between AS and these variants are in the pheromone update and pheromone management details. The AS algorithm implements the basic ACO procedure detailed above. The following paragraphs explain the differences between the selected ACO variants and AS. For further details see [11].

The MAX-MIN Ant System (MMAS) has four major differences from AS: first, the pheromone update is allowed for the iteration-best, i.e. the ant with the best solution for that iteration, or best-so-far ant, i.e. the ant with the best solution over all iterations. Secondly, pheromone limits in an interval $[\tau_{\min}, \tau_{\max}]$ is used to prevent stagnation on local optima. Thirdly, edges are initialized with upper pheromone limits to favor exploration over exploitation in the beginning of the run. Finally, the pheromone trails are reinitialized when the solution does not improve for a number of iterations or stagnation occurs. Pheromones are deposited on the edges according to the equations as given for AS above. The difference is that the ant which is allowed to add pheromone may be either the best-so-far or the iteration-best ant. Commonly in MMAS implementations, both the iteration-best and the best-so-far update rules are used alternatively.

The ACS differs from AS in three main points: first, a pseudo-random proportional action choice rule is used, which allows the exploitation of the ants' search experience. Secondly, pheromone evaporation and deposit is applied to the edges of the best-so-far solutions. Finally, a local pheromone update, which includes evaporation, is applied each time an ant passes through the corresponding edge. This favors exploration over exploitation. At the end of each iteration in ACS, the pheromone trails are again updated similar to in AS, but the pheromone trail updates, both evaporation and new pheromone deposit, are implemented only for the edges belonging to the best-so-far solution.

For the maximum clique version of ACO, each ant is placed on a random node of the given graph *G=<V, E>* where *V* is the set of nodes and *E* is the set of edges. Ants lay pheromones on the edges of the cliques they find through their walk. A tabu list is maintained for each ant, which forces them to visit a node only once. This list contains all the nodes in the ant's trajectory until it gets stuck or it finds a feasible solution. In such a case, the ant restarts its journey and its tabu list is reset. Each ant chooses its next node based on the probabilistic state transition rule, explained above. Also note that nodes are chosen by the ant if they establish a clique with the nodes the ant has already visited, i.e. the next node to be selected should have connections with all nodes in the clique the ant has created. After each ant applies the same rules and creates a solution, pheromone update is performed, based on the used ACO variant.

4. Proposed Solution

Our proposed solution approach for community detection using an ACO technique consists of the following four steps:

1. An ACO technique is used to find the cliques on the given graph.
2. Overlapping cliques are fixed.
3. The graph is transformed into a smaller one where the new edges are created through a scheme based on the concept of *betweenness*.
4. An SNA algorithm is applied on the transformed graph to find the communities.

Ant Colony Optimization for Finding Cliques: In this study, we modified the ACO model proposed for solving the Maximum Clique Problem, where the clique having the highest number of nodes is searched for. In the original version, proposed by Fenet and Solnon [12], the MMAS algorithm was used for the maximum clique search. In our version, ants still try to find the maximum possible clique at each step but once such a clique is found, they continue on their travel on the graph to find other cliques. An ant starts a new clique when there is no eligible neighbor node is left to add to the current clique. This means that no neighbor node has a connection to all the other nodes of the current clique the ant has created so far. To achieve this, each ant keeps a tabu list for the visited nodes as well as the constructed cliques. For the pseudo-random proportional action choice rule, ants use the degrees of the nodes in the neighborhood of the current node as the heuristic information and choose the node with the maximum degree.

In the MMAS model, the best-so-far ant is allowed to deposit its pheromones on its solution path. The amount of pheromone laid on the paths is calculated, based on the *points* gathered by the best ant. A scoring system evaluates the points achieved by an ant, based on the cliques it has found so far. As shown in Eq. (5), the total points gathered by an ant are equal to the sum of the squares of the number of nodes in the discovered cliques. The best-so-far ant has the highest points in the iterations up to the current one.

$$\mathrm{points}(ant_k) = \sum_i (\mathrm{vertices}(C_i))^2, i = 1..nbCliques \tag{5}$$

The pheromone update equations are given in Eq. (6) and Eq. (7). These equations show how the pheromones are distributed over the edges of the cliques found by the best-so-far ant, based on the total point achieved by the ant and the node counts of the cliques. It can be seen from the equations that the edges of the cliques with a higher number of vertices get more pheromone deposited, than the ones with a lower number of vertices. *nbCliques* is the number of cliques found by the current ant and *max_vertices()* function gives the number of vertices in the maximum clique found so far in Eq. (7).

$$\Delta\tau(ant_k) = 1 - (\mathrm{points}(ant_k))^{-1} \tag{6}$$

$$\tau_{ij} \leftarrow \tau_{ij} + (\Delta\tau(ant_k) \cdot \frac{\mathrm{vertices}(C_l)}{\mathrm{max\_vertices}(ant_k)}), l = 1..nbCliques \tag{7}$$

Pheromone initialization depends on the “popular neighborhood” tour executed just before the main algorithm iterations begin. The popular neighborhood is a list of connected nodes to the corresponding node, sorted in decreasing order of

degrees. Each node in the graph has its own popular neighborhood list to be used in the heuristic approach explained above. Pheromone limits are directly related to the number of ants used in the solution for the τ_{min} value and the best-so-far ant's points gathered at the "popular neighborhood" tour for the τ_{max} value. The pheromone levels on all edges are set to the τ_{max} value to favor exploration in the beginning of the run. In Eq. (8) and (9), the function *pn_tour()* gives the total points gained by the scout ant on the above mentioned tour calculate the initial pheromone levels and ρ is the evaporation rate.
.

$$\tau_{\max} = \rho.\mathrm{pn\_tour}() \tag{8}$$

$$\tau_{\min} = \tau_{\max}.(2n)^{-1} \tag{9}$$

The MMAS model used in our work differs from the original one at one major point. Due to the fact that ants traverse all nodes in the graph to find all possible cliques, there is no possibility for them to get stuck at any point in the journey, like in other problems, e.g. for the TSP. In our implementation, when there is no possible move from the current node, the ant goes back to another node in its tabu list and chooses one of its unvisited neighbor nodes to continue. The ant stops only when all the nodes have been visited. Hence, restarting for ants depending on a branching factor is not relevant.

Fixing Overlapping Cliques: Cliques created by the ants in our ACO model have at least one node shared by another clique. The reason behind this is that, to start a new clique, the ant goes back to another node in its tabu list, which has already been included in the current clique. It then continues on from that node to form the new clique. This leads to sharing of a node between the two cliques. To remove these overlaps, we simply detect the shared nodes between the cliques and rearrange the two cliques by assigning the shared node to the clique with the highest number of vertices and removing it from the other. At the end of this step, there are no shared nodes between the cliques.

Transforming the Graph: In this step, new meta-nodes are formed using the cliques, thus reducing the size of the original graph. Each clique becomes a meta-node, which we refer to as a *clique-node*. As a result of this, nodes in a clique will be in the same community when the SNA algorithm runs in the next step. For the reduced graph, new edges will be formed between the clique-nodes, based on the edges between the nodes in the different clique-nodes and the edges between the nodes in the same clique-node. The equation used in calculating the edge weights between the clique-nodes is given in Eq. (10).

$$e_{kl} = \frac{\sum_{i \in C_k} \sum_{j \in C_l} a_{ij}}{\min\left(\sum_{m \in C_k} \sum_{n \in C_k} b_{mn}, \sum_{p \in C_l} \sum_{r \in C_l} b_{pr}\right)} \tag{10}$$

In Eq. (10), a_{ij} is the relevance value of the edges between the clique-nodes while b_{mn} and b_{pr} represent the relevance value of inward edges of clique-nodes. The sum of the edge weights between the nodes of the two clique-nodes is divided by the minimum of the sum of the weights of the edges between the nodes in each clique-node. The resulting value gives the weight value of the new edge e_{kl} between the two clique-nodes. A higher weight means that the two clique-nodes, hence the individual nodes in each, are more likely to be in the same community.

Using an SNA Algorithm for Finding the Communities: In this last step, a clustering-based SNA algorithm, commonly used in literature, is applied on the reduced graph to find communities. While there are several methods in literature to detect community memberships, we have to choose one that works with edge weights rather than just a "0" or a "1", which only shows that there is a relationship between the two nodes or not. This is because our reduced graph has weights assigned to each edge between the clique-nodes, showing the strength of the relationship between them. This chosen approach, proposed by Clauset, et. al. [6] uses a greedy method to calculate modularity differences to build a graph from scratch.

5. Experimental Results

In this section, we present the experimental results of our approach. The algorithm is coded in the C language and we used the iGraph [13] C library for the last step. For all the experiments, we used a single PC (1.6GHz processor with 1GBytes of main memory). We ran iGraph both on the whole graph and on the graph reduced using our approach. We compare our results based on the number of communities detected, the modularity value Q and the execution times of iGraph on the original and the reduced graphs. We used 4 network graphs for our experiments. First 3 are taken from Newman's personal website [15].

- Zachary's Karate Club [14]: A social network of a karate club at a US university. It consists of 34 members and 78 connections in between them.
- American College Football [16]: A set of American football game matches with 115 teams and 616 games.
- Les Miserables [17]: A network data retrieved from the novel *Les Miserables*.
- Chesapeake Bay Food Web [18]: A food web for predator-prey relationship on Chesapeake Bay in US. Data taken from [19].

Parameters which are specific to the ACO techniques used in our implementation are shown on Table 1, where *m* is the number of ants. We use 3 different *q*0 values to calculate the heuristic information. In ACS we used $\xi = 0.1$ for the local pheromone update. For each of the datasets, we executed the algorithms 10 times with each run scheduled to complete in 10 seconds.

Table 1 – ACO parameters

α	β	ρ	m	*q0*
1	2	0.5	25	0 / 0.4 / 0.8

Table 2 shows the confidence intervals for the average number of cliques generated for each dataset, where ACS and MMAS with 3 different *q*0 values are applied. In Table 2, we can see that both ACO techniques achieve similar results on clique creation except for the football games data. For Karate and Chesapeake Bay data, it can be seen that changing the decision rule probability does not affect the number of cliques. However, as the number of vertices and edges increase, differences between ACS and MMAS and the effect of *q*0 increase. For the football games data, ACS finds more cliques than MMAS, which means that MMAS finds better *kernels* upon its execution. This is meaningful as for a given time period of 10 seconds MMAS can find a solution more quickly than ACS, but in the longer runs, ACS is expected to get better results [11].

Table 2 – Lower and upper bounds for cliques created with %95 confidence interval

		Karate	Chesapeake	Les Miserables	Football
	*q*0	lower - upper	lower - upper	lower - upper	lower - upper
ACS	0.0	21.63 - 22.77	18.32 - 20.68	37.94 - 38.06	27.66 - 29.34
	0.4	21.41 - 22.19	18.42 - 20.98	37.81 - 38.19	27.29 - 29.31
	0.8	21.54 - 22.06	18.2 - 20.4	38.19 - 39.21	27.48 - 28.72
MMAS	0.0	21.55 - 22.25	18.76 - 20.44	37.4 - 38	23.94 - 24.46
	0.4	21.55 – 22.25	17.75 - 20.45	37.06 - 37.94	23.54 - 24.06
	0.8	21.7 - 22.1	19.75 - 20.45	37.17 - 37.83	23.16 - 23.44

Table 3, shows the confidence intervals of the average modularity for each dataset. The modularity for each set is calculated via iGraph modularity calculating implementation. Table 4, shows the execution times of iGraph for each set on the reduced graphs obtained using the ACO techniques. On the last rows of both tables, the results of iGraph on the original graph (without any graph reduction) are shown.

Table 3 – Lower and upper bounds for modularity with %95 confidence interval

		Karate	Chesapeake	Les Miserables	Football
	*q*0	lower - upper	lower - upper	lower - upper	lower - upper
ACS	0.0	0.38 - 0.4	0.3 - 0.38	0.48 - 0.58	0.41 - 0.47
	0.4	0.37 - 0.39	0.33 - 0.39	0.4 - 0.56	0.44 - 0.48
	0.8	0.39 – 0.41	0.35 - 0.37	0.52 - 0.6	0.42 - 0.5
MMAS	0.0	0.37 – 0.39	0.32 - 0.36	0.44 - 0.56	0.37 - 0.41
	0.4	0.39 – 0.41	0.33 - 0.39	0.47 - 0.55	0.35 - 0.43
	0.8	0.39 – 0.41	0.34 - 0.38	0.53 - 0.61	0.4 - 0.42
Orig. Graph		0.380671	0.410783	0.547220	0.549741

Table 4 – Execution times

		Karate	Chesapeake	Les Miserables	Football
	*q*0	time spent	time spent	time spent	time spent
ACS	0.0	0.013	0.014	0.016	0.017
	0.4	0.014	0.014	0.016	0.017
	0.8	0.014	0.015	0.016	0.02
MMAS	0.0	0.013	0.013	0.016	0.016
	0.4	0.013	0.014	0.015	0.017
	0.8	0.012	0.013	0.015	0.016
Orig. Graph		0.015	0.015	0.023	0.029

In Table 5, the average number of clusters (communities) is given. The top part of the table shows the number of clusters found by iGraph on the reduced graphs obtained by the corresponding ACS variations. In the "Orig. Graph" row, the number of clusters found by iGraph when it is run on the original graph, with no size reductions, can be seen. The last row gives the actual number of clusters given in literature for the corresponding dataset.

Table 5 – Mean Values of Number of Communities

		Karate	Chesapeake	Les Miserables	Football
	*q*0	communities	communities	communities	communities
ACS	0.0	4.2	3.2	5.6	5.2
	0.4	4	3.1	5.2	5.3
	0.8	4	3	5.8	4.9
MMAS	0.0	3.8	3	6.2	4
	0.4	4	3	5.4	4.2
	0.8	4	3.2	6	4.4
Orig. Graph		3	5	5	6
Actual no. of clusters		2	3	Not given	12

Modularity values in Table 3 show that, values are almost preserved after the transformation which means that we have minimum loss on solution quality. Keeping in mind that higher modularity values show better clustering, best solution for similarity of the results with iGraph is achieved on Les Miserables data with the help of initially provided relevance/similarity values on the edges, while the best solution with respect to the actual number of communities in the dataset is achieved on the Chesapeake data as seen on Table 5. For the football game data, modularity values are higher which may show that the fast greedy community detection algorithm gets better clustering results on lower packing (cliques with less or balanced number of vertices). For all datasets, we obtained faster execution times on the reduced graph as shown on Table 4. Also, except for the Football games data, the numbers of communities found are close to their actual values. However, on Football games data, as seen in Table 5, iGraph got similar results on the original and the reduced graph but the number of clusters found by both is too different from the actual number of the communities in the dataset. For verification of our results, we also applied a modified silhouette index to the clustering solutions obtained of this dataset to measure clustering quality and got results close to 0.75 which means that the clustering on both the original and the reduced graph is adequate. We will try to examine and optimize our approach to get better results on this dataset in our further studies.

6. Conclusions and Future Work

Since many SNA methods suffer from performance loss on large-scale networks, we aimed to find a method to overcome this handicap by reducing the whole graph via clique-nodes found using the ACO techniques. It is shown that the original graph can be reduced to a manageable size keeping in mind that communities can be based upon cliques. The experimental results show that while execution times for the SNA algorithm is lower when a reduced graph is used, solution quality is preserved. For further optimization on performance of our approach, there are some future directions which need to be pursued. Due to the nature of the current algorithm, all ants are traversing all the nodes in the graph to find all cliques. For larger graphs, this will be too time consuming. The algorithm will be modified so that each ant will traverse a part of the graph for a given time frame and the cliques found by the ants will be merged before the graph transformation step. For this modification, ants will be forced to have a common tabu list to avoid traversing the same nodes on the graph. We will also do further experimentation with larger graphs and also on real-world data.

References:

[1] J. Scott, *Social Network Analysis: A Handbook,* Sage Publications, 2002.
[2] M. Girvan and M. Newman, Community structure in social and biological networks, *PNAS*, 99(12):7821-7826, 2002.
[3] L. Donetti and M. Miguel, Detecting network communities: a new systematic and efficient algorithm, *Journal of Statistical Mechanics*, pp. 100-102, 2004.
[4] M. Girvan and M. Newman, Finding and evaluating community structure in networks, *Physical Review E*, 69(026113), 2004.
[5] F. Radicchi, C. Castellano, F. Ceconi, V. Loreto and D. Parisi, Defining and identifying communities in networks, *PNAS*, 101(9):2658-2663, 2004.
[6] A. Clauset, M. Newman and C. Moore, Finding community structure in very large networks, *Physical Review E*, 70(066111), 2004.
[7] K. Wakita and T. Tsurumi, Finding community structure in mega-scale social networks, *16th International World Wide Web Conference*, 2007.
[8] C. Pizzuti, Community detection in social networks with genetic algorithms, *GECCO: Genetic and Evolutionary Computation Conference,* 2008.
[9] Yan Liu, QingXian Wang, Qiang Wang, Qing Yao, and Yao Liu, Email community detection using artificial ant colony clustering, *LNCS 4537*, pp. 287–298, Springer, 2007.
[10] M. Dorigo, *Optimization, learning and natural algorithms*, PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
[11] M. Dorigo and T. Stutzle, *Ant Colony Optimization*, MIT Press, 2004.
[12] S. Fenet and C. Solnon, Searching for maximum cliques with ant colony optimization, *LNCS 2611*, pp. 236–245, Springer, 2003.
[13] G. Csárdi and T. Nepusz, The igraph software package for complex network research, *InterJournal Complex Systems*, 1695, 2006.
[14] W.W Zachary, An information flow model for conflict and fisson in small groups. *Journal of Anthropological Research* 33, pp. 452-473, 1977.
[15] Personal website developed by Mark Newman, Department of Physics, University of Michigan, http://www-personal.umich.edu/~mejn/, accessed May 2009.
[16] M. Girvan and M. E. J. Newman, *Proc. Natl. Acad. Sci. USA* 99, pp. 7821-7826, 2002.
[17] D. E. Knuth, *The Stanford GraphBase: A Platform for Combinatorial Computing*, Addison-Wesley, 1993.
[18] D. Baird and R. E. Ulanowicz, The seasonal dynamics of the Chesapeake Bay ecosystem, *Ecological Monographs* 59, pp. 329–364, 1989.
[19] Courses webpage, developed by Mark Newman, Department of Physics, University of Michigan, http://www-personal.umich.edu/~mejn/courses/2005/cscs535/index.html, accessed May 2009.