A Density-based Algorithm for Computing Community Structure in Directed Social Networks

Yasmine Chaabani[1], Lotfi Ben Romdhane[2]
[1]MARS Research Group
Faculty of Sciences
University of Monastir
Tunisia
[2]MARS Research Group
ISIT'COM, University of Sousse
Tunisia
chaabanijasmin@gmail.com, lotfi.ben.romdhane@usherbrooke.ca

ABSTRACT: *Community detection plays a key role in such important fields as biology, sociology and computer science. For example, detecting the communities in proteinprotein interactions networks helps in understanding their functionalities. Most existing approaches were devoted to community mining in undirected social networks (either weighted or not). In fact, despite their ubiquity, few proposals were interested in community detection in oriented social networks. For example, in a friendship network, the influence between individuals could be asymmetric; in a networked environment, the flow of information could be unidirectional. In this paper, we propose an algorithm, called ACODIG, for community detection in oriented social networks. ACODIG uses an objective function based on measures of density and purity and incorporates the information about edge orientations in the social graph. ACODIG uses ant colony for its optimization. Simulation results on real-world as well as power law artificial benchmark networks reveal a good robustness of ACODIG and an efficiency in computing the real structure of the network.*

Keywords: Ant Colony Optimization, Community Detection, Social Networks, NP-Complete

Received: 10 May 2013, Revised 2 July 2013, Accepted 7 July 2013

1. Introduction

Many complex systems, including physical, biological and social systems as well as many man-made technical systems can be modeled by networks [2]. Networks can be represented by a graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges (or links) representing system units and relations between these units, respectively. When the edges have a direction, the network is called directed and; otherwise, it is called undirected.

Many networked systems are found to divide naturally into modules or communities; i.e., groups of vertices with relatively dense connections within groups but sparser connections between them. Detecting such communities could have a wide range of potential applications in real-world as detecting genes with the same functionality in a biological network, detecting the actors of influence in a political network, etc. From a theoretical perspective, community detection in a social network can be modeled as graph partitioning problem which is known to be NP-complete [10]. To overcome this inherit difficulty, researchers proposed several methods to obtain the best partitioning of the given network [10], [17], [21], [22]. At this stage, it is important to notice that most existing methods can only be applied to undirected networks. However, many complex networks in the real-world are directed. Among these networks, let us mention PPI (Protein-Protein Interaction) networks in biology; the World Wide Web; citation networks in the research community; telecommunication/phone call networks; and email networks to highlight just a few. In this kind of networks, the direction of a link contains important information such as asymmetric influence or information flow. A link between a pair of nodes may represent fundamentally different dynamics when its direction is reversed. Therefore, any kind of approach that disregards the direction of links may fail to understand the dynamics and the function of these directed networks. Also, any kind of community detection approach may fail to detect the communities correctly if the direction of the link is not considered properly. In this regard, several recent proposals [5], [15], [16], [17] have tried to resolve this problem. A common background between all of these methods is that each of them outlines its own definition of a community in a directed network. Actually, no definition is universally accepted. Hereafter, we will outline the most known methods for community detection in oriented social graphs. For a substantial review of the existing approaches as well as their basics, we refer the interested reader to the specialized literature– see for example [10].

Random Walks is a widely used technique for community detection in directed (as well as undirected graphs). The basic idea is to use a random walker from one vertex to another; and edges (resp. nodes) "*central*" to a community will form a trap in a random walk journey. Darong Lai [16] uses PageRank random walk induced network embedding to transform a directed network into an undirected one, where the information on edge directions is effectively incorporated into the edge weights. The purpose of network embedding is to represent each vertex of a network as a low dimensional vector that preserves these similarities between the vertex pairs, usually measured by the edge weights. Starting from this new undirected weighted network, previously developed methods for undirected social graphs can be used without any modification. In [15], Kim et al propose a similar model but based on the importance of links rather than nodes. In fact, the authors proposed a new generalization of modularity based on LinkRank, which is a quantity that indicates the importance of links in a directed social graph. The proposed generalized modularity is the fraction of time spent by a random walker moving within communities minus the expected value of this fraction. In a second phase, any existing model for community detection in undirected networks can be used to optimize this generalized modularity. Although, transforming a directed social network into a weighted undirected gives us the valuable advantage of directly using previously developed methods for undirected networks to find communities in directed ones; there is no real guarantee that this transformation is done without "*information loss*"; i.e., without altering the hidden real community structure of the original oriented network. Stated otherwise, ignoring edge orientations may discard potentially useful information. In addition, these "*random walk*"-based methods suffer from the high spatial complexity [15].

Graph mining is also another used technique for community detection in oriented social networks. In [5], the authors define a community as being a "*dense area*" in the original graph. Thereby, the problem of community detection is reduced to the problem of extracting the set of meaningful dense subgraphs from a given sparse graph. The proposed idea in the algorithm bears some similarities with the problem of reordering/blocking matrices in sparse matrix techniques and which utilizes the cosine similarity of matrix columns. Doing it this way, a partial clustering of the vertices in a graph is computed, where each cluster represents a dense subgraph. The proposed algorithm is parametric and requires a density threshold above which the output subgraphs are considered to be dense. Unfortunately, the proposed method is unable to detect communities with unbalanced sizes [5]. Stated otherwise, in the presence of large dense communities; smaller sparse ones will be ignored.

Hierarchical clustering is a widely used technique, among others, which puts together similar vertices into larger communities. Hierarchical clustering algorithms build the communities gradually in a hierarchical manner. Sometimes we use some terminating conditions to select the partition or the group of partitions that satisfy a given criteria such as the number of communities desired, the minimum (or maximum) number of objects in each community, the optimization of an objective function, etc. [10]. In [17], the well-known modularity function is generalized in a principled fashion to incorporate the information contained in edge directions. Then the community structure of the networks is computed by maximizing this generalized modularity function using an hierarchical clustering approach; i.e., over several possible divisions of the network. Unfortunately, as any modularitybased model, this approach will ignore small communities which will be merged with bigger ones. This is known in the literature as the "*resolution limit*" problem and is discussed in details in [11].

In this paper, we propose an algorithm, called *ACODIG*, for community detection in directed social networks. In *ACODIG*, we define an objective function that takes into account edge orientation, and we use ant colony for its optimization. A strong feature of our model is its robustness in detecting communities of unbalanced sizes; and thereby avoids the "*resolution limit*" problem. The rest of this paper is structured as follows. In Section II, we outline preliminary material. Section III details our proposals; illustrates it on a sample network; and analyzes its time and space complexity. In Section IV, we conduct an experimental analysis and compare our model to other recent proposals using large scale real-world and synthetic social networks; while the last section offers concluding remarks. Many complex systems, including physical, biological and social systems as well as many manmade technical systems can be modeled by networks [2]. Networks can be represented by a graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges (or links) representing system units and relations between these units, respectively. When the edges have a direction, the network is called directed and; otherwise, it is called undirected.

Many networked systems are found to divide naturally into modules or communities; i.e., groups of vertices with relatively dense connections within groups but sparser connections between them. Detecting such communities could have a wide range of potential applications in real-world as detecting genes with the same functionality in a biological network, detecting the actors of influence in a political network, etc. From a theoretical perspective, community detection in a social network can be modeled as graph partitioning problem which is known to be NP-complete [10]. To overcome this inherit difficulty, researchers proposed several methods to obtain the best partitioning of the given network [10], [17], [21], [22]. At this stage, it is important to notice that most existing methods can only be applied to undirected networks. However, many complex networks in the real-world are directed. Among these networks, let us mention PPI (Protein-Protein Interaction) networks in biology; the World Wide Web; citation networks in the research community; telecommunication/phone call networks; and email networks to highlight just a few. In this kind of networks, the direction of a link contains important information such as asymmetric influence or information flow. A link between a pair of nodes may represent fundamentally different dynamics when its direction is reversed. Therefore, any kind of approach that disregards the direction of links may fail to understand the dynamics and the function of these directed networks. Also, any kind of community detection approach may fail to detect the communities correctly if the direction of the link is not considered properly. In this regard, several recent proposals [5], [15], [16], [17] have tried to resolve this problem. A common background between all of these methods is that each of them outlines its own definition of a community in a directed network. Actually, no definition is universally accepted. Hereafter, we will outline the most known methods for community detection in oriented social graphs. For a substantial review of the existing approaches as well as their basics, we refer the interested reader to the specialized literature– see for example [10].

Random Walks is a widely used technique for community detection in directed (as well as undirected graphs). The basic idea is to use a random walker from one vertex to another; and edges (resp. nodes) "*central*" to a community will form a trap in a random walk journey. Darong Lai [16] uses PageRank random walk induced network embedding to transform a directed network into an undirected one, where the information on edge directions is effectively incorporated into the edge weights. The purpose of network embedding is to represent each vertex of a network as a low dimensional vector that preserves these similarities between the vertex pairs, usually measured by the edge weights. Starting from this new undirected weighted network, previously developed methods for undirected social graphs can be used without any modification. In [15], Kim et al propose a similar model but based on the importance of links rather than nodes. In fact, the authors proposed a new generalization of modularity based on LinkRank, which is a quantity that indicates the importance of links in a directed social graph. The proposed generalized modularity is the fraction of time spent by a random walker moving within communities minus the expected value of this fraction. In a second phase, any existing model for community detection in undirected networks can be used to optimize this generalized modularity. Although, transforming a directed social network into a weighted undirected gives us the valuable advantage of directly using previously developed methods for undirected networks to find communities in directed ones; there is no real guarantee that this transformation is done without "*information loss*"; i.e., without altering the hidden real community structure of the original oriented network. Stated otherwise, ignoring edge orientations may discard potentially useful information. In addition, these "*random walk*"-based methods suffer from the high spatial complexity [15].

Graph mining is also another used technique for community detection in oriented social networks. In [5], the authors define a community as being a "*dense area*" in the original graph. Thereby, the problem of community detection is reduced to the problem of extracting the set of meaningful dense subgraphs from a given sparse graph. The proposed idea in the algorithm bears some similarities with the problem of reordering/blocking matrices in sparse matrix techniques and which utilizes the cosine similarity of matrix columns. Doing it this way, a partial clustering of the vertices in a graph is computed, where each cluster represents a dense subgraph. The proposed algorithm is parametric and requires a density threshold above which the output subgraphs are considered to be dense. Unfortunately, the proposed method is unable to detect communities with unbalanced sizes [5]. Stated otherwise, in the presence of large dense communities; smaller sparse ones will be ignored.

Hierarchical clustering is a widely used technique, among others, which puts together similar vertices into larger communities. Hierarchical clustering algorithms build the communities gradually in a hierarchical manner. Sometimes we use some terminating conditions to select the partition or the group of partitions that satisfy a given criteria such as the number of communities desired, the minimum (or maximum) number of objects in each community, the optimization of an objective function, etc. [10]. In [17], the well-known modularity function is generalized in a principled fashion to incorporate the information contained in edge directions. Then the community structure of the networks is computed by maximizing this generalized modularity function using an hierarchical clustering approach; i.e., over several possible divisions of the network. Unfortunately, as any modularitybased model, this approach will ignore small communities which will be merged with bigger ones. This is known in the literature as the "*resolution limit*" problem and is discussed in details in [11].

In this paper, we propose an algorithm, called *ACODIG*, for community detection in directed social networks. In *ACODIG*, we define an objective function that takes into account edge orientation, and we use ant colony for its optimization. A strong feature of our model is its robustness in detecting communities of unbalanced sizes; and thereby avoids the "*resolution limit*" problem. The rest of this paper is structured as follows. In Section II, we outline preliminary material. Section III details our proposals; illustrates it on a sample network; and analyzes its time and space complexity. In Section IV, we conduct an experimental analysis and compare our model to other recent proposals using large scale real-world and synthetic social networks; while the last section offers concluding remarks.

2. Basic Concepts

2.1 Problem formulation

A social network can be modeled by a graph $G = (V, E)$ where V denotes the set of vertices and $E \subseteq (V, V)$ denotes the set of edges. In a directed network an edge is an ordered vertex pair (i, j), while in an undirected one both the vertex pair (i, j) and (j, i) represent the same edge. A network can be represented by an adjacency matrix A whose elements are non negative; i.e., A_{ij} is positive if there is an edge between vertex i and vertex j, and 0 otherwise. We should note that A is asymmetric for directed graphs. The problem of community detection could be defined as follows.

[Community detection] Let $G = (V, E)$ be a directed graph modeling the directed social network at hand. Let f be an objective function measuring the quality of a partitioning of G. The problem of community detection consists in finding a partitioning $P = \{C_1,...,C_k\}$ of G_k such that: (i) $C_i \subset V$; (ii) $C_i \cap C_j = \emptyset \forall C_i, C_j \in P$; (iii) $\cup_{i=1}^{k} C_i = V$; and (iv) $f(P)$ is optimal.

From Definition II-A, we can state that a good partitioning is that optimizing a given objective function measuring the overall quality of the detected communities. In a computed partitioning, we assume that communities do not overlap. We should notice that we do not know a priori neither the size nor the number of communities.

2.2 Notations and basic definitions

The main goal of this section is to introduce preliminary concepts and the basic definitions fundamental to our model. Given a directed graph $G = (V, E)$, we denote by $|V|$ its number of vertices and $|E|$ its number of edges.

[Degree of vertex] Given a graph $G = (V, E)$, then the degree of a vertex s is the number of incoming and outgoing edges from s given by:

$$Degree(s) = \sum_{i \in G} E(s_i, s) + E(s, s_i) \qquad (1)$$

[Importance of a vertex] Given a graph $G = (V, E)$, we define the importance of a vertex s as the fraction of the sum of the outgoing edges from s divided by the sum of the maximum number of incoming edges and the maximum outgoing edges existing in G. It is given by:

$$Importance(s) = \frac{D_{out}(s)}{D_{max\_in} + D_{max\_out}} \qquad (2)$$

From Definition II-B, we can see that the importance of a node increases as the number of its outgoing edges increases which means that it flows more information to the rest of he nodes. Intuitively speaking, the importance of a node in a directed graph quantifies how "*influential*" is this node in forwarding information to the rest of the graph [22].

[Direct neighbor] In the graph $G = (V, E)$, vertex v is said to be a direct neighbor of vertex s if s have a directed edge to v. This relationship is represented by the edge $(s, v) \in E$. We should notice from Definition II-B that, unlike in undirected graphs, if node s is a direct neighbor of node v; then the opposite is not necessarily true.

[Cardinality of a community] The cardinality of a community C, denoted by $| C |$, is its number of vertices.

[Free vertex] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, a vertex $s \in C_i$ is said to be free if C_i contains only s. In other terms, a free vertex composes a community on its own.

Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$ and a vertex s, we denote by $NFDN(s)$ the number of neighbors of s that are at the same time free and direct.

[Compatibility of a vertex to a community] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, a vertex s and a community $C \in P$; we define the compatibility of s to C as the number of incoming edges to s from C and is given by:

$$comp\,(s, C) = \sum_{s_i \in C} E\,(s_i, s) \tag{3}$$

In Definition II-B, the compatibility of a vertex to a community quantifies its "*attachment*" to it.

[Internal degree of a vertex] Given a partition $P = \{C_1,...,C_k\}$ of $G = (V, E)$, a vertex s and a community $C \in P$; we define the internal degree of s in C as the number of outgoing and ingoing edges to s from the members of C; and is given by:

$$Degree_{in}\,(s, C) = \sum_{s_i \in G} E\,(s_i, s) + E\,(s, s_i) \tag{4}$$

[Compatibility of a vertex] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$ and a vertex s; we define the compatibility of s as the the maximum between the number of its free and direct neighbors and the values of its compatibility to all existing communities in P; and is given by:

$$Comp_{max}\,(s) = max\,\{Comp\,(s, C_i), \forall C_i \in P; NFDN\,(s)\} \tag{5}$$

Intuitively, the compatibility of a vertex (in definition II-B) answers the following question: is a free vertex, say s, closer to an already computed community; or closer to its free direct neighbors (and with whom s may compose a new community)?

[Importance of a community] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, we define the importance of a community C in P as the the average importance of its nodes; i.e.,

$$Importance\,(C) = \frac{\sum_{s_i \in C} Importance\,(s_i)}{| C |} \tag{6}$$

[Local density of a vertex] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, a vertex s and a community $C \in P$; we define the local density of node s in community C as the the fraction of the internal degree of s in C divided its degree in G; and is given by:

$$Density\,(s, C) = \frac{degree_{in}\,(s, C)}{degree\,(s)} \tag{7}$$

where the $degree_{in}(s, C)$ is given by equation (4) and *degree* (s) by equation (1). The local density of a node s w.r.t. a given community C quantifies how "*typical*" is s w.r.t. C.

[Density of a community] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$ and a community $C \in P$, we define the density of C as the average density of its vertices; and is given by:

$$Density\,(C\,) = \frac{1}{|\,C\,|}\sum_{s_i \in C} Density\,(s_i\,, C\,) \tag{8}$$

Now, we are ready to define the density of a partitioning as follows:

[Density of a partitioning] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, we define the density of P as the average density of its communities; and is given by:

$$Density\,(P\,) = \frac{1}{|P|}\sum_{c_i \in P} Density\,(C_i\,) \tag{9}$$

The density of a node, community and a partitioning has the following natural properties, respectively.

For each vertex $s \in G$, we have: $0 \le Density\,(s) \le 1$. For each Community $C \in P$, we have: $0 \le Density\,(C\,) \le 1$. For each partitioning P, we have: $0 \le Density\,(P) \le 1$. After having defined the concept of density, now we will introduce the concept of purity. [Purity of a vertex] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, a vertex s and a community $C \in P$; s is said pure w.r.t. C if the following holds:

$$Comp\,(s, C) = Comp_{max}(s) \tag{10}$$

where $Comp\,(s, C)$ is the compatibility of s to C given by equation (3); and $Comp_{max}\,(s)$ is the maximal compatibility of node s given by equation (5). From Definition II-B, one can state that a vertex s is pure to C if it is attached to C more than it is attached to its direct free neighbors (and with whom it may compose a new community) and to the rest of already computed communities. [Purity of a community] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$ and acommunity $C \in P$, we define the purity of C as the average of the importance of its pure vertices. It is computed by:

$$Purity\,(C) = \frac{1}{|C|}\sum_{s_i \in\ C\,/\,s_i\ is\ pure} Importance\,(s_i\,) \tag{11}$$

[Purity of a partitioning] Given a partitioning $P = \{C_1,...,C_k\}$ of $G = (V, E)$, we define the purity of P as the average of the weighted sum of the purity of its communities; and is given by:

$$Purity\,(P) = \frac{1}{|P|}\sum_{C_i \in P} w_i * Purity\,(C_i\,) \tag{12}$$

where w_i is a weighting factor measuring the importance of community C_i w.r.t. the rest and is given by:

$$w_i = \frac{|\,C_i\,|}{max\,\{|C_j|;\ \forall C_j \in P\}} \tag{13}$$

The best partitioning maximizes purity. In fact, this case corresponds to a partitioning in which each vertex is more attached to its community than to the rest. The purity measure has the following straightforward properties. Each free vertex is an impure vertex. For each Community $C \in P$, we have: $0 \le Purity\,(C) \le 1$. For each partitioning P, we have: $0 \le Purity\,(P) \le 1$.

Given all these definitions, now we are ready to outline our proposal subsequently.

3. Our Proposal

Our model, called *ACODIG*[1], uses an objective function that characterizes the overall quality of a partitioning and uses ant colony for its optimization.

[1]ACODIG is an acronym for Ant colony optimization for Community detection in DIrected Graphs

3.1 The objective function of our model

Our objective function is based on the concepts of purity and density outlined in the previous section. Given a partitioning $P = \{C_1,...,C_k\}$ of graph $G = (V, E)$, it is qualified by:

$$DP\,(P) = \frac{Density\,(P) + Purity\,(P)}{2} \qquad (14)$$

where *Density* (*P*) and *Purity* (*P*) are given by equations (9) and (12), respectively. Maximizing our objective function tends to produce a partitioning in which communities are dense and pure; i.e., in which vertices of the same community are well connected together and well separated from the rest of communities. Hence, maximizing our objective function will maximize the connectivity intra community and minimize the connectivity inter community. Our objective function in equation (14) has the following property. For any partitioning *P*, we have $0 \leq DP\,(P) \leq 1$.

In order to maximize our objective function, we use ant colony optimization (ACO, for short), a class of meta heuristics which turned out to be efficient in solving a variety of NPhard problems to our satisfaction. ACO was first introduced by Marco Dorigo in his PhD thesis [8]. ACO is based on a collective behavior for filling and tracking of colonies observed in ants. A colony of simple ants communicate indirectly via dynamic changes in their environment (pheromone trails) and built as a solution to a problem, based on their collective experience. ACO has shown promise in solving hard problems including project scheduling [18], travelling salesman [13], vehicle routing [12] and routing in communications networks [7] to highlight just a few. For a survey of ACO in solving hard problems, we refer the interested reader to [4], [19], [20]. Subsequently, we detail the distinct steps of our approach.

3.2 Algorithm

In *ACODIG*, outlined in Algorithm 1, we start by an initial partitioning in which each vertex is considered as being free; i.e., not included yet in any community. For this, the vector *FREE* is composed of all vertices of the graph. By the same way, the set of computed partitions *P* is empty. In computing the set of communities, the nodes are considered with respect to their "*influence*" in the social graph. This influence is measured by their importance degree given by equation (2). For this, we compute thereafter the importance of each vertex and sort them (in descending order) w.r.t. that importance. Indeed, the most important influential vertex (and which is still free) will be the starting point (center) for he next community (line 6 in the algorithm). Thereafter, we call *Mark* outlined in Algorithm 2 which accepts as parameters the graph *G*, the center *v*, and outputs the computed community C_j.

In Algorithm 2, we initialize community C_j with its its central node *S* on which we place an ant with a given label color. Therefore, *S* is labelled with the color of this ant. Then, we compute the set of direct free neighbors of *S*. The ant on *S* will probabilistically put other ants (with the same color label) on this set of free direct neighbors. Generally speaking, the probability of transition from vertex *i* to vertex *j* is subject to the following probability function:

$$P_{i,j} = Importance\,(j) * Purity\,(j, C_i) \qquad (15)$$

where *Importance* (*j*) is the importance of vertex *j* and *Purity* (j, C_i) is the purity of vertex *j* with respect to community C_i and computed by equations (2) and (10), respectively. We should notice that this probability is high as long as vertex *j* is important in the graph and is "*close*" (pure) to community C_i (to which the labelled vertex *i* already belongs). For this, free and direct neighbors of *S* are considered with respect to this probability. Being on a given vertex, an ant has to decide to add it to the actual community (by labelling it with its color label) or not. This decision is based upon the value of our objective function *DP* given by equation (14).

If the actual node will increase the value of that function, then it is added; otherwise, it is ignored (and thereby remains unlabelled). After adding a vertex to the actual community, the latter is recursively extended from that vertex (line 8 in Algorithm 2). In the next subsection, we will illustrate the distinct steps of *ACODIG* on a sample direct graph.

3.2.1 Illustration

The main purpose of this section is to illustrate the distinct steps of our algorithm *ACODIG* on a sample oriented social graph composed of 12 nodes an 20 edges (see Figure 1).

Algorithm 1: ACODIG

```
Input: A graph $G = (V, E)$
Output: A partitioning $P = \{C1, ..., C_k\}$
begin
1     $FREE \leftarrow V$
2     $P \leftarrow \emptyset$
3     Sort vertices in $FREE$ with respect to their
      importance (in descending order)
4     $j \leftarrow 1$
5     while ($FREE \neq \emptyset$) do
6         $v \leftarrow$ (the vertex in $FREE$ with the highest
          importance)
7         $C_j \leftarrow \emptyset$
8         Mark ($G$, $v$, $C_j$)
9         delete labelled nodes from $FREE$
10        $P \leftarrow P \cup C_j$
11        $j \leftarrow j + 1$
      end
12    return $P$
end
```

Algorithm 2: Mark (G, S, C)

```
Input: A graph $G = (V, E)$, vertex $S$
Output: $C$, a set of labelled (colored) vertices
begin
1     $C \leftarrow C \cup S$
2     Place an ant on vertex $S$ and label it with the label
      color of that ant
2     $PROB(S) \leftarrow$ (the probabilities of transitions from 3
      node $S$ to its direct and free neighbors computed by
      equation (15))
4     Sort $PROB$ ($S$) (in descending order)
      for $k = 1 \rightarrow PROB(S).size$ do
5         $v_k \leftarrow$ (the vertex with the highest probability in
          $PROB$ ($S$ ))
6         if ($DP(C) < DP(C \cup v_k)$) then
7             place an ant on $v_k$ with the same color label
              as $S$
8             Mark ($G$, $v_k$, $C$)
          end
      end
end
```

Initially all vertices are in *FREE*. After computing the importance of the vertices, it appears that vertex n_1 is the most important and therefore will be the central node of the first community, say C_1. Then, we call algorithm *Mark* (G, n_1, C_1) which will consider the direct and free neighbors of n_1 w.r.t. their transition probabilities. The direct free neighbors of n_1 is the set $\{n_3, n_4, n_5, n_6, n_9\}$; and vertex n_9 has the maximal probability of transition. Unfortunately, adding n_9 will not increase our objective function, and therefore it is not added. The next explored neighbor of n_1 is n_3 which increases our objective function; and thereby is labelled (colored) with same color as n_1. This process is recursively repeated until no extensions of C_1 are possible–see figures 2(a)–(h). Then the next most important free vertex is n_9 which will compose next community C_2– see figures 2(i)– (l). Consequently, *ACODIG* divides our sample graph into two communities $C_1 = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}$ and $C_2 = \{n_9, n_{10}, n_{11}, n_{12}\}$– see figure 3.

In the next section, we will analyze the complexity (temporal and spatial) of *ACODIG*.

3.2.2 Complexity of our algorithm

Due to the large scale of real world social networks, the complexity (temporal and spatial) becomes crucial. In this section, we will theoretically analyze the time and space complexities of our algorithm *ACODIG*. The following theorem is about the time complexity. The time complexity of *ACODIG* is $O(n^2)$ where n is the number of vertices in the social directed network.

Proof: To evaluate the time complexity of our algorithm, we start by calculating complexity of each step. First, we consider the initialization step of the degree vector and the partition initial time complexity of this step is $O(n)$. The vector calculation step open containing the values of the importance of vertices is done with a time complexity $O(n^2)$. Then, we sort the vertices according to the value of their importance in the free vector the time complexity of sorting (bubble sort) and the distribution is $O(n^2)$. The time complexity of the initialization phase is:

$$T_{initial} = O(max(n, n^2))$$
$$= O(n^2)$$

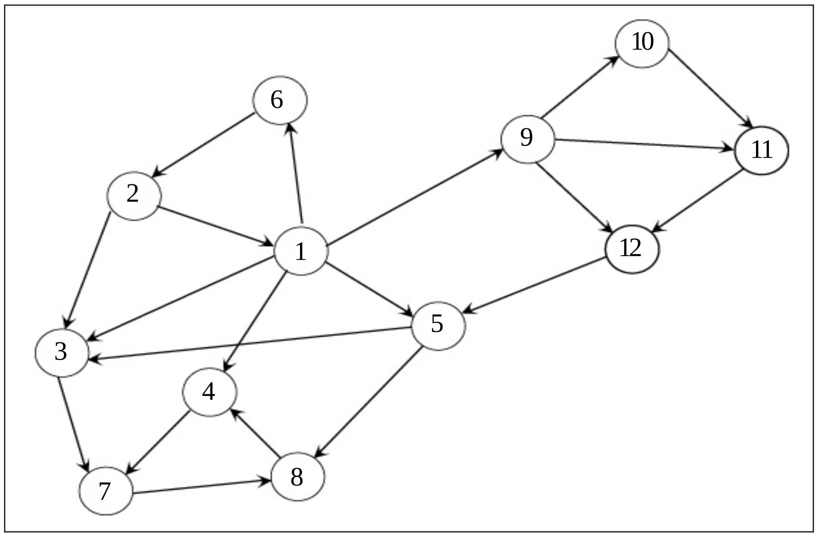

Figure 1. Sample directed graph

After the distribution of ants, we apply the functions *mark* on each element of free vector. Hence the complexity of recursive function *Mark* is $O(n^2)$. The complexity of removing labelled nodes is $O(n)$. Hence, The time complexity of this phase is $O(n^2)$.

$$T_{main} = O(max(n, n^2))$$
$$= O(n^2)$$

So the total time complexity of the worst case is:

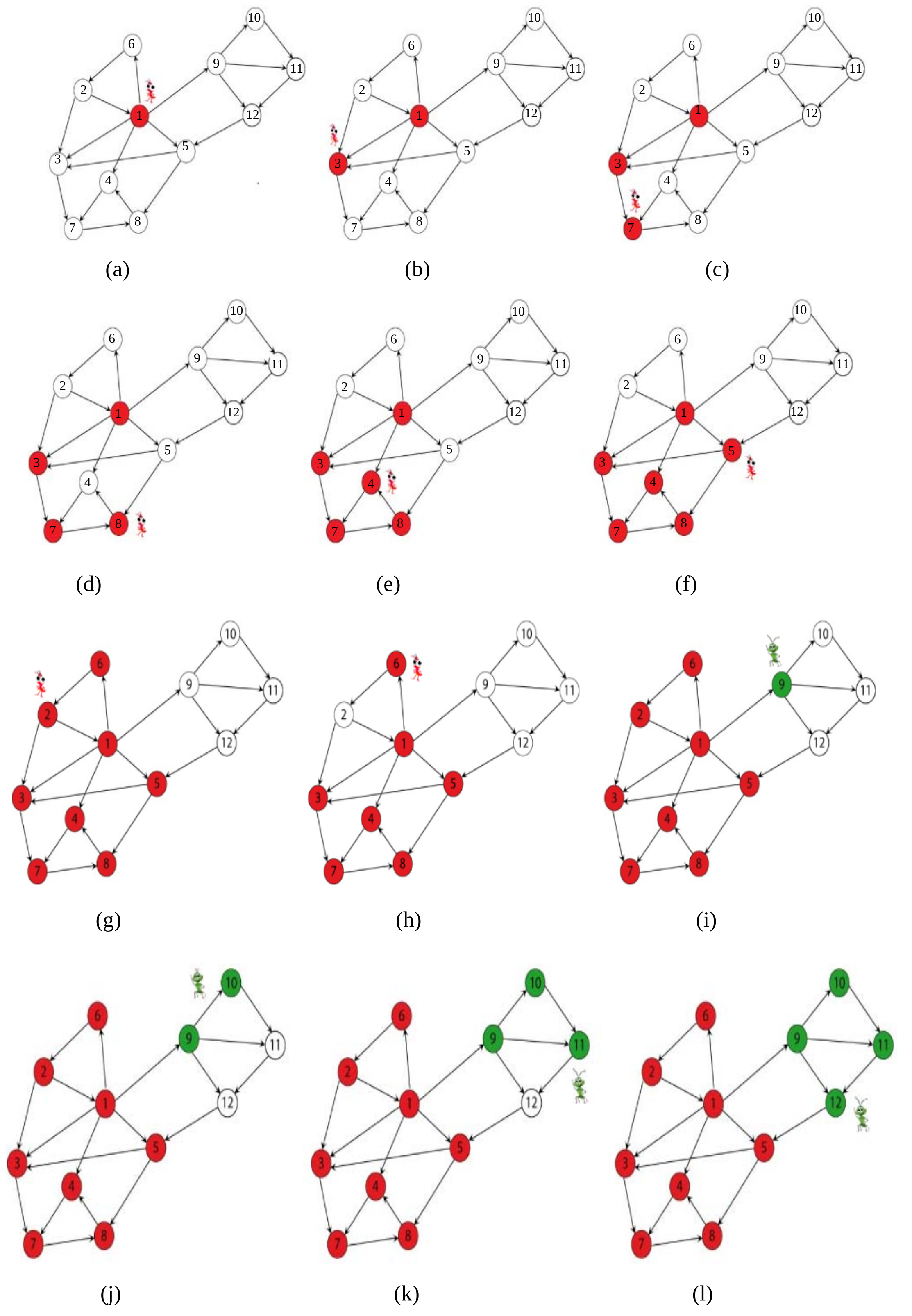

Figure 2. Visualization of the main steps of our algorithm

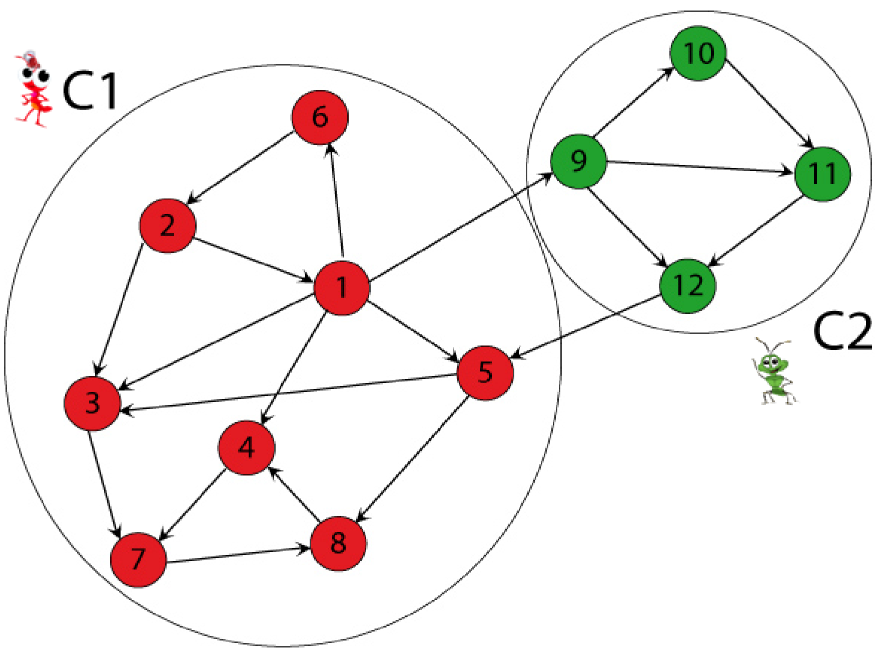

Figure 3. Computed communities

$$complexity\_temporal = O(n^2) \quad (16)$$

The following theorem is about the spatial complexity of *ACODIG*. The space complexity of *ACODIG* is $O(m)$ where m is the number of edges (relations) in the social directed network.

Proof: In our approach, we use two vectors: the first vector stores the importance of nodes; and the second stores the free nodes. We also build a graph with m edges. Considering the worst case where the input graph is very dense (w.r.t. the number of edges), then we have:

$$complexity\_spatial = O(max\{m, n\}= O(m) \quad (17)$$

After having theoretically analyzed the complexity of our algorithm *ACODIG*, we will analyze it experimentally and compare it to other proposals in the next section.

4. Benchmarks

The main purpose of this section is to analyze experimentally our model *ACODIG* and to compare it to other recent proposals for community detection in directed graphs. We will compare *ACODIG* to two well-known proposals; namely, the model in [15] (noted *LR*) based on link ranking, and the model in [17] (noted *M*). We should remind that both proposals are based on an objective function derived from the standard well-known modularity. As a test bed, we considered large scale real-world networks as well as synthetic networks generated using the LFR benchmarks [9]. As evaluation criteria for a computed partitioning, we considered these standard measures: density, coverage and performance. When the real partitioning is available (which is the case for LFR benchmarks), we considered in addition the normalized mutual information. Now, we will define these criteria. For this, let $P = \{C_1, C_2,..., C_k\}$ a computed partitioning by any model.

The density of a partitioning is defined as follows [6]:

$$D(P) = \sum_{C_i \in P} \frac{D(C_i)}{|P|} \quad (18)$$

where the density of a community *C* is computed by:

$$D(C) = \frac{|E|}{|V|(|V|-1)} \tag{19}$$

We should notice that a good partitioning should maximize density; i.e., produce dense communities. As for the coverage criteria, it measures the fraction of edges inside a community w.r.t. the total number of edges in the graph and is given by [3]:

$$Coverage\,(P) = \sum_{i=1}^{k} \frac{|E_i|}{|E|} \tag{20}$$

Regarding the performance, it measures the rate of edges well positioned according to the partitioning *P* and is defined by [3]:

$$Perf(P) = 2\left(\frac{|False+|+|False-|}{N(N-1)}\right) \tag{21}$$

where $|False+|$ is the number of inter-community edges; $|False-|$ is the number of pairs (v_i, v_j) of a non-adjacent community. However, when the real community structure is available, we can compare it to the computed one using the normalized mutual information (NMI) given by [15]:

$$NMI(A,P) = \frac{-2\sum_{a\in A}\sum_{b\in P}|a\cap b|\,log\,(\frac{|a\cap b|\,n}{|a||b|})}{\sum_{a\in A}|a|\,log\,(\frac{|a|}{n}) + \sum_{b\in P}|b|\,log\,(\frac{|b|}{n})} \tag{22}$$

where *A* is the real structure of the network. We should notice that $NMI(A, P) = 1$ when both partitions *A* and *P* coincide. Given all these performance criteria, now we are ready to outline experimental results.

4.1 Evaluation on artificial networks

In this first set of runs, we used randomly generated social networks using the LFR benchmarks [9] and whose structure is imposed during the generation phase. Having the latter as a reference point, we can use the normalized mutual information *NMI* in equation (22) to evaluate the quality of the computed partitioning. The LFR generator has several parameters among them let us mention the number of nodes (*N*); the average degree of incoming edges (*k*); the maximum degree of the incoming edges (*maxk*); the fraction between incoming and outgoing edges inside a community (μ); the minimal community size (*minc* and the maximal community size (*maxc*). We started our simulation with reference graphs and by varying one or several parameters of the generator, we obtained several graphs of different complexities. For each simulation, we considered two graphs and computed the CPU time (in seconds) taken by the model to output a partitioning[2].

In addition, we take into account the computed number of communities (*#comm*) compared to the real exact one (*EP*).

4.1.1 First case: references graphs

We considered reference graphs composed of $N = 5000$, $k = 15$, $maxk = 50$, $\mu = 0.1$; $minc = 20$ and $maxc = 50$. Simulation results are reported in Tables 1 – 3.

First, we should remark that the model *LR* [15] was unable to compute a partitioning of the considered graphs mainly due to their size. In fact reporting to [15], all used graphs are of small size mainly due to the high spatial complexity of the algorithm. We notice from Tables 1 and 3 that our algorithm *ACODIG* computed better partitioning than *M* [17] with respect to *NMI* and the number of communities. Unfortunately, this resulted in much more CPU time (see Table 2).

[2] Due to systems' overload, the CPU time is averaged over five trials for each case

	M	LR	ACODIG
NMI (*g*1)	0.512	-	0.892
NMI (*g*2)	0.558	-	0.911

Table 1. NMI for Reference Graphs

	M	LR	ACODIG
CPU (*g*1)	154.123	-	161.434
CPU (*g*2)	162.156	-	165.381

Table 2. Average CPU Time (in Seconds) for Reference Graphs

	M	LR	ACODIG	PE
# *comm* (*g*1)	102	-	154	156
# *comm* (*g*2)	105	-	155	156

Table 3. Number of Computed Communities Compared to the Exact one (EP) for Reference Graphs

4.1.2 Second case: large communities

In this second set of runs, we consider graphs with large communities with *minc* = 30; *maxc* = 80. Simulation results are summarized in Tables 4 – 6.

	M	LR	ACODIG
NMI (*g*1)	0.745	0.521	0.956
NMI (*g*2)	0.785	0.498	0.914

Table 4. NMI for Graphs with Large Communities

	M	LR	ACODIG
CPU (*g*1)	64.562	842.465	66.241
CPU (*g*2)	66.254	975.397	67.834

Table 5. CPU Time (in Seconds) for graphs with large communities

	M	LR	ACODIG	PE
# *comm* (*g*1)	28	20	29	30
# *comm* (*g*2)	28	19	30	31

Table 6. Number of Computed Communities Compared to the Exact one (EP) for Graph with Large Communities

From Table 4, we can see that our algorithm *ACODIG* has the best partitioning with respect to *NMI*. This means that it computes partitions that are very close to the exact real ones. Regarding the same criteria, *M* performs better than *LR*. In addition, we notice in Table 6 that the computed number of communities by *ACODIG* are very close to the real ones than *M* or *LR*. Regarding the CPU time in Table 5, our algorithm performs better than *LR* but has equivalent performance to *M*.

4.1.3 Third case: small communities:

In this third set of runs, we consider small communities with *minc* = 10, *maxc* = 30. Simulation results are summarized in Tables 7 – 9.

	M	**LR**	**ACODIG**
NMI (*g*1)	0.356	0.578	0.899
NMI (*g*2)	0.389	0.599	0.923

Table 7. NMI for Graphs with Small Communities

	M	**LR**	**ACODIG**
CPU (*g*1)	71.256	855.164	70.445
CPU (*g*2)	70.895	999.224	71.369

Table 8. CPU Time (in Seconds) for graphs with Small Communities

	M	**LR**	**ACODIG**	**PE**
# comm (*g*1)	21	45	69	70
# comm (*g*2)	22	44	66	69

Table 9. Number of Computed Communities Compared to the Exact one (EP) for Graph with Small Communities

Here again, our model gives the best partitioning with respect to the normalized mutual information as well as the number of computed communities. As for the CPU time in Table 8, our model performs as well as *M*; and much better than *LR*.

4.1.4 Fourth case: sparse graphs

In this set of runs, we consider sparse graphs with k = 20. Remind that k controls the average degree of incoming edges. Simulation results are summarized in Tables 10-12.

	M	**LR**	**ACODIG**
NMI (*g*1)	0.784	0.458	0.968
NMI (*g*2)	0.791	0.496	0.933

Table 10. NMI for Sparse Graphs

	M	**LR**	**ACODIG**
CPU (*g*1)	50.456	513.148	49.498
CPU (*g*2)	51.195	582.391	48.342

Table 11. CPU Time (in Seconds) for Spares graphs

	M	**LR**	**ACODIG**	**PE**
# comm (*g*1)	51	48	67	68
# comm (*g*2)	52	46	69	69

Table 12. Number of Computed Communities Compared to the Exact one (EP) for Sparse Graph

Here again *ACODIG* outperforms *LR* and *M* with respect to *NMI*, the number of computed communities as well as the CPU time. We should notice also that *M* performs better than *LR*.

4.1.5 Fifth case: very dense graphs

As a final simulation with LFR, we considered dense graphs with k = 100. Simulation results are summarized in Tables 13–15

	M	**LR**	**ACODIG**
NMI ($g1$)	0.598	0.651	0.978
NMI ($g2$)	0.568	0.698	0.966

Table 13. NMI for Dense Graphs

	M	**LR**	**ACODIG**
CPU ($g1$)	68.592	899.664	69.497
CPU ($g2$)	66.544	789.734	65.453

Table 14. CPU Time (in Seconds) for Dense graphs

	M	**LR**	**ACODIG**	**PE**
# comm ($g1$)	25	30	48	51
# comm ($g2$)	28	33	50	52

Table 15. Number of Computed Communities Compared to the Exact one (*EP*) for Dense Graph

Considering dense graphs, we can notice that our model performs much better than *LR* and *M*. It took less CPU time and computes partitions that are very close to the exact ones. This may be explained by the fact that, in principle, *ACODIG* is based on the concept of density (see Definition II-B).

4.2 Evaluation on real networks

In this second set of runs, we consider real networks. Unfortunately, the real hidden structure for these networks is, in general, unknown. For this, we can not use the normalized mutual information as evaluation criteria. Instead, in addition to the objective function of each model, we will consider the density, coverage and performance criteria in equations (18), (20) and (21), respectively. The used real graphs are described in Table 16.

	N	**E**
Football network [14]	10	44
Neurons network [11]	306	2345
Political blogs network [1]	1490	19090

Table 16. Characteristics of the Real Graphs: N Denotes the number of Nodes and E Denotes the number of Edges

Subsequently, we will describe each network along with the simulation results.

4.2.1 Football network

The football network is a small one with ten vertices. It models the relationships of interaction between teams (winning and losing) football for the 2005 season. *ACODIG* divides this network into two communities depicted in Figure 4. This partitioning corresponds perfectly to that in [17]. Values of the considered criteria for this network are reported in Table 17. From this table we can see that our model *ACODIG* is the best for *Density* and our objective function DP as well as the CPU time. The model *M* is the best w.r.t. the *modularity* and Coverage criteria; whereas the model *LR* is the best w.r.t. *Performance* criteria.

4.2.2 Neurons network

This neurons networks of *C*. Elegans is made a directed network of 306 nodes (neurons), connected through 2345 links (synapsis, gap junctions) [11]. It represents the structure of the nervous system of the nematode. We can notice from the simulation results reported in Table 18 that our model is better w.r.t. *Density*, *DP*, *Performance* and CPU time. However, the model

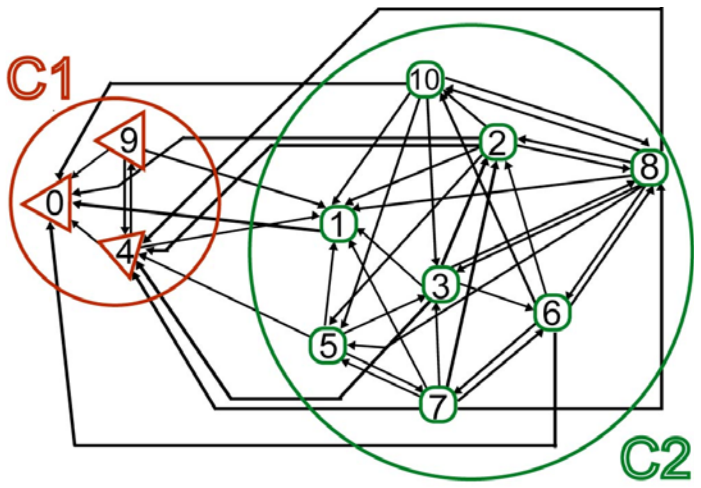

Figure 4. Communities detected by ACODIG in the football network

	M	**LR**	**ACODIG**
Modularity	**0.691**	0.592	0.571
Density	0.354	0.374	**0.581**
DP	0.589	0.548	**0.642**
Coverage	**0.481**	0.451	0.367
Performance	0.582	**0.699**	0.669
CPU (s)	2.133	268.541	**2.012**

Table 17. Quality Measures for the "*Football Network*"

	M	**LR**	**ACODIG**
Modularity	**0.681**	0.611	0.474
Density	0.659	0.298	**0.692**
DP	0.478	0.329	**0.667**
Coverage	**0.577**	0.532	0.423
Performance	0.641	0.414	**0.691**
CPU (s)	10.422	995.258	**9.589**

Table 18. Quality Measures for the "*Neurons Network*"

M is he best considering the modularity and coverage criteria. Although the model *LR* has the worst performance w.r.t. to all criteria, but it remains not so far from both *ACODIG* and *M* especially for the modularity and coverage criteria.

4.2.3 Political blogs network

The "*Political Blog Network*" is a large directed network between a set of weblogs about US politics recorded by Adamic and Glance [1]. In this network there are totally 1490 nodes and 19090 links. Each node is labelled as either conservative or liberal. Simulation results for this network are summarized in Table 19. From this table, we notice that the model *LR* was unable to compute a partitioning due to the large size of this network. We can state also that *ACODIG* performs better considering the *Density*, *DP*, *Performance* and CPU criteria. However, the model *M* is better w.r.t. modularity.

As a summary to all these simulation results on real world and artificial networks, we can say that our algorithm *ACODIG* was

	M	LR	ACODIG
Modularity	**0.695**	-	0.551
Density	0.569	-	**0.684**
DP	0.482	-	**0.564**
Coverage	**0.512**	-	0.454
Performance	0.458	-	**0.624**
CPU (s)	90.122	-	**89.152**

Table 19. Quality Measures for the "Political Blogs" Network

able to compute good partitions in reasonable time scales.

5. Conclusion

We have proposed in this paper *ACODIG*, a model for community detection in oriented social networks. *ACODIG* qualifies a given partitioning using an objective function modeling both concepts of density and purity. While the density tends to produce dense communities; the purity guarantees the "*typicality*" of each vertex to its own community. Doing it this way, we guarantee the computation of a partitioning in which we have dense communities well separated from the rest. *ACODIG* uses ant colony for the optimization of this objective function. Simulation results on real world and synthetic social networks reveal a good performance of our proposal in computing optimal partitions in reasonable time scales. These results should stimulate future research. Extending *ACODIG* to weighted social graphs with overlapping communities (i.e., in which a single vertex may belong to several communities) constitute our immediate focus.

References

[1] Lada Adamic, Natalie Glance. (2005). The political blogosphere and the 2004 u.s. election: Divided they blog. *In*: LinkKDD 05: Proceedings of the 3rd International Workshop on Link discovery, p. 36–43.

[2] Réka Albert, Albert-László Barabási. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74, 47–97, Jan.

[3] Brandes, U., Gaertler, M. (2003). Experiments on graph clustering algorithms. LNCS, 14 (1) 51–65.

[4] Chandrasekhar, U., Naga, P. R. P. (2011). Recent trends in ant colony optimization and data clustering: A brief survey. *In*: Intelligent Agent and Multi-Agent Systems (IAMA), 2011 2nd *International Conference on*, p. 32–36.

[5] Chen, J., Saad, Y. (2010). Dense subgraph extraction with application to community detection. *Transactions on knowledge and data engineering*, 14 (1) 78–92.

[6] Chen, J., Saad, Y. (2010). Dense subgraph extraction with application to community detection. *Transactions on knowledge and data engineering*, 14 (1) 78–92.

[7] Gianni Di Caro, Marco Dorigo. (1998). Antnet: distributed stigmergetic control for communications networks. *J. Artif. Int. Res.*, 9 (1) 317–365, December.

[8] Dorigo, M. (1992). Optimization, learning and natural algorithms. *Politecnico di Milano*, 32 (2) 137–172.

[9] San Fortunato. Benchmark graphs to test community detection algorithms. http ://sites.google.com/site/santofortunato/in the press, 2.

[10] Santo Fortunato. (2009). Community detection in graphs. *Physics Report*, p. 75–174.

[11] Santo Fortunato, Marc Barthélemy. (2007). Resolution limit in community detection. *In*: Proceedings of the National Academy of Sciences of the United States of America, p. 36–41.

[12] Gambardella, L., Taillard, E., Agazzi, G. (1999). New ideas in optimization, chapter macs-vptw: A multiple ant colony system for vehicle routing problems with time windows. McGraw Hill, London, UK, 79 (1) 379– 458.

[13] Rongwei Gan, Qingshun Guo, Huiyou Chang, Yang Yi. (2010). Improved ant colony optimization algorithm for the traveling salesman problems. *Systems Engineering and Electronics, Journal of,* 21 (2) 329–333.

[14] Girvan, M., Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99 (12) 7821–7826, June.

[15] Youngdo Kim, Seung-Woo Son, Hawoong Jeong. (2010). Finding communities in directed networks. *Phys. Rev.* E, 81, 016103, Jan .

[16] Darong Lai, Hongtao Lu, Christine Nardini. (2010). Finding communities in directed networks by pagerank random walk induced network embedding. *Physica A: Statistical Mechanics and its Applications*, 389 (12) 2443–2454.

[17] Leicht, E. A., Newman, M. (2007). Community structure in directed networks. *physics.data-an*, 5 (1) 91–96, January.

[18] Merkle, D., Middendorf, M., Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on*, 6 (4) 333–346.

[19] Chandra Mohan, B., Baskaran, R. (2012). A survey: Ant colony optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 39 (4) 4618 – 4627.

[20] Suri, B., Singhal, S. (2012). Literature survey of ant colony optimization in software testing. *In*: Software Engineering (CONSEG), 2012 CSI Sixth International Conference on, p. 1–7.

[21] Zardi, H., Ben Romdhane, L. (2013). An $o(n2)$ algorithm for detecting communities of unbalanced sizes in large scale social networks. *Know.- Based Syst.,* 37, 19–36, January.

[22] Xiaohang Zhang, Ji Zhu, Qi Wang, Han Zhao. (2013). Identifying influential nodes in complex networks with community structure. *Knowledge-Based Systems*, 42 (0) 74 – 84.