Approximation of the Maximal $\alpha-$Consensus Local Community detection problem in Complex Networks

Patricia Conde-Céspedes
Université Paris 13,
L2TI (EA 3043),
F-93430, Villetaneuse, France.
patricia.conde-cespedes@univ-paris13.fr

Blaise Ngonmang
Université Paris 13,
L2TI (EA 3043),
F-93430, Villetaneuse, France.
blaise.ngonmang@univ-paris13.fr

Emmanuel Viennet
Université Paris 13,
L2TI (EA 3043),
F-93430, Villetaneuse, France.
emmanuel.viennet@univ-paris13.fr

***Abstract*—The problem of community detection has received great attention by the complex networks researchers in the last decades. Although the notion of community does not actually have a unanimous accepted definition, it is generally admitted that it consists in a set of densely connected nodes. Moreover, density measures the strength of the relationships in the community. The need of these well connected and dense communities has led to the notion of $\alpha-$consensus community. An $\alpha-$consensus community, is a group of nodes where each member is connected to more than a proportion α of the other nodes. An $\alpha-$consensus community is maximal if and only if adding a new node to the set breaks the rule. Consequently, an $\alpha-$consensus community has a density greater than α. Existing methods for mining $\alpha-$consensus communities generally assume that the network is entirely known and they try to detect all such consensus communities. In some cases, the network can be so large that each node can only have local information or one can be only interested in the $\alpha-$consensus set of a particular node in the network. In this paper, we propose an efficient algorithm based on local optimizations to approximate the maximal $\alpha-$consensus local community of a given node. The proposed method is evaluated experimentally on real and artificial complex networks in terms of quality, execution time and stability. It provides better results than the existing methods.**

1. Introduction

Community detection in social networks has gained considerable attention in social network analysis see [1]. Although the notion of community does not actually have a unanimous accepted definition, it is generally admitted that it consists of a set where each node shares many connections with the others. Therefore, a community must contain densely connected nodes. Moreover, density plays an important role because it measures the strength of the relationships in the community. The need of these well connected and dense communities has led to the notion of α-consensus community. An α-consensus community, is a group of nodes where each member is connected to more than a proportion α of the other nodes, we call this rule *the majority rule*. An α- consensus community is maximal if and only if adding a new node to the set breaks the rule. Consequently, an α-consensus community has a density greater than α.

Mining all the maximal α-consensus sets of a network is known to be NP-Complete [2] and has received a great attention in the literature. Efficient exact methods or approximations to solve it are available. However, all these methods generally assume that the network is entirely known and they try to find all such consensus communities.

In some cases, the network can be so large that one can only have local information about some nodes or one can be only interested in the consensus set of a particular node in the network. In this paper, we present a novel local method that uses information on the neighborhood of a node to approach its maximal α-consensus community. The proposed method is evaluated experimentally on real and computer generated complex networks in terms of quality, execution time and stability. It provides better results than the existing ones.

This paper is organised as follows: section 2 presents useful definitions and notations. Section 3 formalises the problem using previously defined notations. Section 4 gives an overview of related works. section 5 draws our solutions to this problem. In section 6 the proposed solutions are evaluated and the results are discussed. Finally, section 7 draws some conclusions and perspectives.

2. Useful definitions and notations

A social network 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, formed by pairs of vertices. The two nodes u and v are the end vertices of the edge $e = (u, v)$. If the order of end vertices matters in an edge, then the graph is said to be directed otherwise, it is undirected. The neighborhood $\Gamma(u)$ of a node u, is the set

This paper is partially funded by the French REQUEST project.

of nodes v such that $(u, v) \in E$. The degree of a node u is the number of its neighbours or the cardinality of $\Gamma(u)$, i.e. $degree(u) = |\Gamma(u)|$. The degree of node u will be denoted by $d(u)$. Given this model, all graph theoretic tools can be reused in network analysis.

Given a community-detection method, if it starts from a given node and uses only local information the resulting community is called *local community*. The detected local community of a node n_0 will be denoted $C(n_0)$. If the detection method is iterative, at any iteration the following notation we denote D: set of nodes identified as members of $C(n_0)$ and S: set of all neighbors of nodes in D that do not belong to D.

3. Problem formulation

The problem, this paper tries to tackle, can be summarized as follows:

"Given a node n_0 of a graph $G(V, E)$ and a parameter α $(0 < \alpha < 1)$, the purpose is to find the biggest $\alpha-$consensus community $C(n_0)$ containing n_0".
Mathematically, this problem can be formalized as follows:

$$\begin{array}{ll} \underset{C}{\text{maximize}} & |C| \\ \text{subject to} & n_0 \in C \\ \text{and} & |\Gamma(n_i) \cap C| > \alpha(|C| - 1), \forall n_i \in C. \end{array} \quad (1)$$

Notice that an $\alpha-$consensus community has a density greater than α.

4. Related works

We considered only existing methods that garantee that the resulting communities are α-conscensus: [1]

4.1. The *QUICK* method

The *QUICK* method [8] was designed to efficiently extract all the maximal $\alpha-$cliques of a given network. This algorithm uses a depth first search method to explore the search space: starting with a node n_0, it builds all the α cliques of increasing sizes that contain n_0. At each following step, it moves to the next unexplored node and builds α-cliques that do not contain already explored nodes. Because the search space is exponential on the number of nodes, some pruning techniques are used. They are based on the degrees of the nodes and the diameters of the constructed $\alpha-$cliques. Despite the proposed pruning techniques, this method is not suitable for very large graphs and it does not guarantee that each node will be assigned to the largest clique it belongs to.

1. A problem very similar to the one studied in this paper is *local community detection* as describe in some existing works [3], [4], [5], [6] and [7]. However, the methods described in those previous works do not constrain the resulting communities to respect the α-conscensus rule. Therefore, they do not address the same problem and they are not considered in this paper for comparisons.

4.2. The Louvain method

The Louvain's algorithm [9] is one of the fastest methods for global community detection in large networks. A global method decomposes the whole network into disjoint communities. That is, it returns a partition on the set of nodes that optimizes a given criterion or quality function. Therefore, it absolutely needs as input the whole network. In the opposite, a local community detection method searches for a community for a particular node, so it absolutely needs as inputs a node and, depending on the situation it can need the whole network or a part of it. Then, the output of a global method is a partition.

The *Louvain* heuristic approaches the partition that maximizes a given criterion or quality function. The original version of *Louvain* was designed to quickly optimize the modularity quality function or Newman-Girvan (NG) criterion [10]. Because of the rapidity of Louvain method, in [11] the authors have recently proposed to adapt the Louvain's method to other quality functions or global community detection criteria.

A global criterion of particular interest for us is the Zahn-Condorcet (ZC) criterion (see [12], [13], [14], [15] and [16]) since the resulting partition is composed of $\alpha-$consensus communities when choosing $\alpha = 0.5$ (see [15] for proof). Later in [17], the authors formulated a generalisation of the Condorcet's problem by introducing a parameter α, $0 < \alpha < 1$ in the Condorcet's function. We call this version the Owsiński-Zadrożny (OZ) criterion. The partition that optimizes the OZ is composed of $\alpha-$consensus communities (see [15] for proof). For $\alpha = 0.5$ this criterion is equivalent to Condorcet's criterion.

The *Louvain* method is very fast compared to many community detection methods. However, there is no guarantee on the size of the communities using this method.

5. Proposed Solutions

In order to approach the solution of the problem (1) we have elaborated three algorithms or solutions which we called *RANK-GAIN*, *RANK-GAIN+* and *RANK-NUM-NEIGHS*. Each new version has been proposed in order to face shortcomings not solved by the previous algorithms.

5.1. Solution 1: The *RANK-GAIN* Algorithm

For any node n belonging to the neighborhood S of the local community D, the gain resulting from the addition of n to D, noted g_n is given by:

$$g_n = \ell_n - \alpha|D|, \quad (2)$$

where ℓ_n is the number of links connecting n to D.

For a node n to respect the majority rule, its gain g_n must be positive. The greater this gain g_n is, the more neighbors in D node n has. Furthermore, a high value of gain contributes to increase the density of the community. This first algoritm is iterative, at each iteration at most one node is added to D. The algorithm chooses to add the node n^* whose gain is maximal and positive. If more than one node have the maximal gain we prioritize the one who has more neighbors in S. Indeed, when a node n is added to D the gain of all of its neighbors in S will increase by $(1-\alpha)$ for the next iteration. Therefore, there will be more candidates for the next iteration and more possibilities to enlarge the community to approach the solution of problem (1).

5.1.1. Disadvantages of this solution. Let us suppose we are looking for the maximal local consensus community of node 0 in figure 1 with $\alpha = 0.5$. Algorithm *RANK-GAIN* will return $C(n_0) = \{0, 1, 2, 3, 4\}$ whereas 0 does not respect the Condorcet's rule. The optimal $\alpha-$consensus local community of node 0, denoted $C^{opt}(n_0)$, is either $\{0, 1, 2, 3\}$ or $\{0, 1, 2, 4\}$.

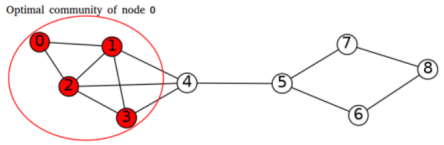

Figure 1. The problem of internal degree.

The majority rule states that every node in the local community must be connected to more than $\alpha\%$ of the rest of the nodes in D (see problem (1)). When choosing a new node to enter D the algorithm previously verifies if the new node satisfies the rule. However, it does not verify if the already existing nodes still verify the rule. Certainly, before the addition of the new node the condition was verified by all the existing nodes in D. Nonetheless, once a new node joins D the rule becomes more strict to be satisfied because every node must be connected to more nodes in D. In order to face this problem we propose the solution in the next section.

5.2. Solution 2: The *RANK-GAIN+* algorithm

5.2.1. The internal degree. We have seen in section 5.1.1 that given a set of nodes D, the growth of this community is limited by the *internal degree* of the existing nodes in D. We call *the internal degree of* n, denoted $d^{in}(n)$, the number of internal connections of n to nodes in D. For the example in figure 1, if $\alpha = 0.5$ and $D = \{0, 1, 2, 3\}$, the internal degree of those nodes are 2, 3, 3 and 2 respectively, and the community size is 4. Since every node must be connected to more than half of its neighbors, adding a new node may cause nodes 0 and 3 to break the rule because in a community of size 5 every node must be connected to at least 3 nodes. So, the addition of a new node is not possible unless the new node is connected to nodes 0 and 3. This leads to the following theorem:

Theorem 5.1. Given a node n with internal degree $d^{in}(n)$ the maximum possible size of an α-consensus community it can belong to, denoted $D^{max}(n)$, is given by:

$$D^{max}(n) = \left\lfloor \left(\frac{d^{in}(n)}{\alpha} \right) \right\rfloor. \tag{3}$$

The notations $\lceil x \rceil$ and $\lfloor x \rfloor$ represent the ceiling and the floor functions of a real number x respectively.

Now, let us suppose D is an α-consensus community. We denote $d_{min} = \min_{n \in D}\{d^{in}(n)\}$ the *minimal internal degree* of all the nodes in D. If we want to add nodes to D to increase its size the maximum possible size D can reach and still be an α-consensus community, denoted D^{max}, is:

$$D^{max} = \left\lfloor \left(\frac{d_{min}}{\alpha} \right) \right\rfloor \tag{4}$$

Given a community D, a node $n \in D$ is *saturated* if $d^{in}(n) = d^{min}$ or $|D| = D^{max}(n)$.

If we consider the example in figure 1 for $D = \{0, 1, 2, 3\}$, $d_{min} = 2$, $D^{max} = 4$, so nodes 0 and 3 are saturated.

5.2.2. Modifications to the *RANK-GAIN* algorithm. If no node is saturated it is possible to choose any node from S to enter D as long as it verifies the rule. However, if the local community contains saturated nodes the only possibility to increase its size is to choose a node in S from the set of commun neighbors of all saturated nodes. If a node n is saturated and $d(n) = d^{in}(n)$ (its degree is equal to its internal degree) there is no possibility to add more nodes to D because n has no neighbors in S. Otherwise n would break the rule. The only possibility to increase D when it contains saturated nodes is if $d_n > d^{in}(n)$. Let us consider once more the graph in figure 1 and $D = \{0, 1, 2, 3\}$. Node 0 is saturated and $d(0) = d^{in}(0)$. In this case, there is no any possibility of improvement, so the algorithm must stop.

Henceforth, we will say that a node is *supersaturated* if it is saturated and its degree is equal to its internal degree. Taking into account the constraint of internal degree, some modifications were made to algorithm 1. At each iteration a previous verification if there is at least one saturated node is done. There are three possibilities:

1) If no node is saturated, then the new node is chosen according to the highest positive gain (and neighbors if ties) as in algorithm 5.1.
2) If there is at least one saturated node and no node is supersaturated, then the eventual new node is chosen from the subset in S made of the common neighbors

of all saturated nodes, if this set is empty, the algorithm stops.
3) If there is at least one supersaturated node the algorithm stops.

5.2.3. Shortcomings of the *RANK-GAIN+* algorithm. Let us consider the following examples and $\alpha = 0.5$:

Shortcoming 1: Let us consider the graph in the figure 2. Let us suppose we are looking for the maximal α-consensus community of node 0. The optimal solution is $D^{opt} = \{0, 1, 2, 3, 4, 5\}$ (nodes in red in the graph on the right). Let us suppose at the end of the third iteration when applying *RANK-GAIN+* the resulting community is $D = \{0, 1, 2\}$ (as shown in the graph on the left). Among all the candidates in $S = \{3, 4, 5, 6\}$ for the next iteration the only candidate with positive gain is node 6. So, it will be chosen to enter D and the algorithm will stop and return the suboptimal solution $D = \{0, 1, 2, 6\}$ (nodes in red in the graph on the left). Indeed, once node 6 enters the community no more improvement is possible.

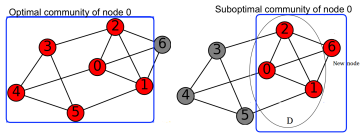

Figure 2. Shortcoming 1

If the only parameter taken into account is the gain when searching for a new node, nodes 3, 4 and 5 will never enter the local community as they have a negative gain.

Shortcoming 2: Now, consider the graph in the figure 3. Let us suppose we are looking for the maximal $\alpha-$consensus community of node 5. The optimal solution is $D^{opt} = \{5, 6, 7, 8\}$ (nodes in red). After the first iteration, the *RANK-GAIN+* algorithm will add either node 6 or 7 to the community and then stops. Therefore, the resulting community will be of size 2. Indeed, the other two nodes will never be able to enter the community as their gain is null.

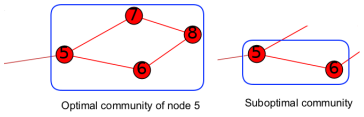

Figure 3. Shortcoming 2

In the next section we will propose a method which solves this type of Shortcomings.

5.3. *RANK-NUM-NEIGHS*: An Algorithm based on the number-of-neighbors

5.3.1. The importance of the number of neighbors. For the shortcomings described in section 5.2.3, the right nodes could not enter the community because their gains were negative or null. Actually, allowing one of them to enter the community would break the majority rule. However, in the optimal solution all nodes verify the rule. For example, node 3 in case 1 has negative gain equal to -0.5, so if it joins the community it will not verify the rule. Nevertheless in the optimal partition, $D^{opt} = \{0, 1, 2, 3, 4, 5\}$, it does verify the rule as it is connected to two more nodes, 4 and 5. However, the gain of each of these three nodes becomes positive and equal to 0.5 if they enter the community together.

Why not to let enter more than one node at each iteration? Why not to prioritize the number of neighbors in S rather than the gain? Indeed, the gain is not the most important thing when choosing the new node to enter D. At this point, let us remind the problem of the maximal $\alpha-$consensus community we want to solve (equation (1)):

Given a node n_0 in the graph, the purpose is the find the biggest community containing n_0 where each node is connected to more than $\alpha\%$ of the other nodes.

So, the most important thing is the size of the community. We are not looking for the densest community but for the biggest community where all nodes must respect the majority rule. To this end, we propose an algorithm that prioritizes the entry of nodes with the **highest number of connections in** S over the gain and allows to enter one node or more at each iteration. We call this new version *RANK-NUM-NEIGHS*.

5.3.2. *The RANK-NUM-NEIGHS* algorithm. According to what was discussed in the previous section, this new algorithm presents two main innovations:

1) It prioritizes the number of common neighbors over the gain when choosing a new node (or more nodes) to enter the local community.
2) At each iteration more than one node might enter the local community.

The algorithm is iterative. At each iteration either, no node, one node or a set of nodes is or are chosen to enter D. It contains four functions:

1) Function *rank_by_neighbors()*: It returns a ranking of nodes in S according to their number of neighbors in S.
2) Function *rank_by_gain()*: given the graph G, two sets of disjoint nodes: D and S, and α; it returns a ranking for all the nodes in S according to their gain if added to D (see equation (2)).
3) Function *add_nodes()*: this is the principal function of the algorithm. This function takes as parameters the graph G, the local community D, a set of nodes S as the possible candidates to enter D and the parameter α. It returns a set of nodes Π^* from S to add to D.
4) The main function *algorithm()*: it takes as inputs the graph G, the node n_0 and the parameter α. The algorithm adds a set of nodes Π^* to D as long as it is possible

based on the existing saturated nodes. At each iteration the set of nodes to add is returned by the *add_nodes()* function. If the *add_nodes()* function returns an empty set, it means that there is no possibility of improvement, then the algorithm stops and returns the local community D.

The *add_nodes()* function is decribed in algorithm 1. When selecting the best candidates to enter D this function makes a previous verification of the respect of the majority rule. In the following, if a subset of nodes denoted $\Pi \in S$ are such that after being added to D all nodes in the resulting community verify the majority rule we will say that nodes in Π are *good* candidates to enter D.

First, the function calculates the following variables:

- RC is the number of remaining candidates.
- $Rank$: the ranking of nodes in S calculated by function *rank_by_neighbors*.
- r: the *current rank* of neighbors, initialized to the highest rank in $Rank$.
- $\mathfrak{C}_r$: the set of nodes whose rank is r.
- $Gain \leftarrow$ rank_by_gain$(G, D, \mathfrak{C}_r, \alpha)$: the ranking of nodes in $\mathfrak{C}_r$ according to their gain.
- g: the current gain, it is initialized to the highest gain found in $Gain$).
- $\mathfrak{Cand}$, called the set of current candidates, is the set of nodes in S with rank r and gain g.
- Π^*: the list of selected nodes to enter D. It can contain no node, only one or more than one nodes.
- A boolean variable I indicating if a set of *good* candidates has been found (*true*) or not (*false*).

Next, the function selects randomly a node n^* from the set of candidates $\mathfrak{Cand}$. According to the values of current rank r and current gain g, four situations can take place:

Situation 1: $g > 0$, the gain of n^* is positive. If all nodes in D verify the majority rule after the addition of n^*, it means that $n*$ is a *good* candidate and the function returns $\Pi^* = n^*$, otherwise n^* is dropped from the list of candidates $\mathfrak{Cand}$ and the number of candidates RC decreases by one. This process is repeated until either a *good* node has been found (therefore I is *true*) or there are no more candidates in $\mathfrak{Cand}$.

Situations 2, 3 and 4 take place when $g \leq 0$. In this case n^* can not be added alone to D because, it does not verify the rule. In this case, the function tests if it is possible to let it enter simultaneously with some of its neighbors in S. The number of neighbors n^* will need is announced in theorem 5.2.

Theorem 5.2. Given an α-consensus community D, its neighborhood S and a node $n \in S$ whose gain is negative, the number of neighbors in S that n needs to enter D to obtain a positive gain, denoted x, is:

$$x = \left\lfloor \left(\frac{\alpha|D| - \ell_n}{(1-\alpha)} \right) \right\rfloor + 1 \tag{5}$$

where ℓ_n in the number of links between n and D.

This means that n needs x more connexions, besides ℓ_n to respect the majority rule in the resulting community.

Let us denote $S(n)$ the set of the neighbors of any node n in S. According to the value of x we distinguish the following situations:

Situation 2: $g \leq 0$ and $r < x$, so any node n among the candidates $\mathfrak{C}_r$ has fewer neighbors in S than required to enter D. Even if all of its neighbors enter D its gain will stay negative or null. This will happen with all the nodes in $\mathfrak{Cand}$ and with all nodes in $\mathfrak{C}_r$ as they have the same rank (number of neighbors) and at most the gain g. Therefore, $\mathfrak{Cand}$ is set to the empty set because no candidate in $\mathfrak{Cand}$ is a *good* candidate. The number of remaining candidates RC is decreased by $|\mathfrak{C}_r|$.

Situation 3: $g \leq 0$ and $r = x$, so any node n among the candidates $\mathfrak{C}_r$ has as many neighbors in S as required to enter D. The function chooses one node n^* from $\mathfrak{Cand}$ and tests if all nodes in the set $\Pi = \{n^* \cup S(n^*)\}$ are *good* candidates. If that is the case, the function returns $\Pi^* = \Pi$ as the set of nodes to enter D. Otherwise n^* is dropped from the list of candidates $\mathfrak{Cand}$ and the number of candidates RC decreases by one. This process is repeated until either a *good* set of nodes has been found (therefore I is *true*) or there are no more candidates in $\mathfrak{Cand}$.

Situation 4: $g \leq 0$ and $r > x$, the function chooses one node n^* randomly from $\mathfrak{C}_r$ and verifies if n^* can enter D with x of its neighbors. The set of chosen neighbors of n^* is denoted $S(n^*)(x)$. The function chooses up to x nodes from $S(n^*)$ according to the following rule:
rule 1: Nodes in $S(n^*)$ with the highest ranks are chosen first. If there are ties and more nodes are needed, the function selects nodes with the highest gain. If still more nodes are required and there are ties in rank and gain the remaining nodes are chosen randomly among all possible combinations.
Different combinations are tested until either one combination $\Pi = \{n^* \cup S(n^*)(x)\}$ composed of only *good* nodes has been found or all the possible combinations of $S(n^*)(x)$ nodes have been tested. Once one combination of *good* candidates has been found the function returns $\Pi^* = \Pi$. Otherwise n^* is dropped from the list of candidates $\mathfrak{Cand}$ and the number of candidates RC decreases by one. A new node n^* is chosen randomly from $\mathfrak{Cand}$. This process is repeated until either a *good* set of nodes has been found (therefore I is *true*) or there are no more candidates in $\mathfrak{Cand}$.

If the set of candidates is empty and no solution Π^* of *good* nodes has been found (I is *false*), the set of

candidates $\mathfrak{Cand}$ is updated. If $\mathfrak{Cand}$ is empty because there are no more candidates with the current gain, the functions skips to the next value of gain g. If $\mathfrak{Cand}$ is empty because there are no more candidates with the current rank, the current rank r is updated to the next value. Finally we update $\mathfrak{Cand}$ as the set of nodes in S with rank r and gain g. If all the ranks have been tested the function stops and returns the set Π^*. This operation repeats as long as there are still candidates, i.e. $RC > 0$ and no set of *good* nodes has been found, i.e. $I =$ *false*.

The function might return an empty set Π^* if no set of *good* nodes has been found.

The algorithm proposed in this section solves all the shortcomings previously studied. In order to generalize the performance of this third version we compared the sizes of the local communities obtained by *RANK-NUM-NEIGHS* to those obtained by *RANK-GAIN+*. We ran the two algorithms on real and artificial networks (the networks characteristics are described on 6). The obtained results show that *RANK-NUM-NEIGHS* performs better than *RANK-GAIN+* in most cases in both real and artificial networks.

5.3.3. A post-processing step. We added a post-processing step to the algorithm to enhance the results. When looking for the maximal $\alpha-$concencus community of a node n_0 in a graph, the local communities of all the neighbors of n_0 are estimated as well. Let n be the neighbor of n_0 whose community $C(n)$ contains n_0 and is the biggest among those of all its neighbors. If $C(n)$ is bigger that $C(n_0)$ the function sets the local community of n_0 to $C(n)$.

6. Evaluations

The *RANK-NUM-NEIGHS* algorithm is evaluated according to three criteria: the sizes of the detected communities, the stability and the execution time. The evaluations concerning the stability are not presented in this paper. We tested the stability of the algorithm on artificial networks of different sizes. We executed the algorithm for different values of the parameter α. For every node n of each graph we made 10 iterations of the algorithm. For these 10 sets we calculated the Jaccard index for the 10 sets. More than 80% of times we obtained a Jaccard index equal to 1, which means that on average for more of 80% of the nodes we obtained the same result after 10 iterations. The remaining 20% presented a lot of variability.

We generated artificial LFR graphs (see [18]) of sizes ranging from 1000 to 10000 nodes by increments of 1000. The input parameters are the same as those considered in [19]. We used 4 real social networks: "The Zachary Karate Club network (karate)" [20], "The College football network (football)" [21], "Political blogosphere (polblogs)" [22] and "Books about US politics (polbooks)" [23].

Algorithm 1 The *add_nodes()* function for *RANK-NUM-NEIGHS* algorithm.

Require: A non-weighted graph $G = (V, E)$, a set of nodes D, a set of nodes S, the parameter α
Ensure: A set of *good* nodes Π^* from S to enter D.

Calculate
- $RC \leftarrow |S|$, $Rank \leftarrow$ *rank_by_neighbors*(G, S), r (highest rank in $Rank$), $\mathfrak{C}_r$.
- $Gain \leftarrow$ *rank_by_gain*$(G, D, \mathfrak{C}_r, \alpha)$, g (highest gain in $Gain$).
- $\mathfrak{Cand} \leftarrow$ set of nodes in $\mathfrak{C}_\mathfrak{r}$ whose gain is g.
- Set $I \leftarrow$ *false*, $\Pi^* = \{\varnothing\}$.

while ($RC > 0$ and I= *false*) **do**
  if $g > 0$ **then**
    while ($\mathfrak{Cand} \neq \{\varnothing\}$ and I=*false*) **do**
      choose a node n^* randomly from $\mathfrak{Cand}$.
      if n^* is a *good* candidate **then**
        Set $\Pi^* = \{n^*\}$ and I= *true*.
      else
        Drop n^* from $\mathfrak{Cand}$ and $RC = RC - 1$.
      end if
    end while
  else
    Calculate: $x \leftarrow \left\lfloor \left(\frac{\alpha|D|-\ell}{(1-\alpha)} \right) \right\rfloor + 1$, where $\ell = (g + \alpha|D|)$.
    if ($x > r$) **then**
      Set $RC = RC - |\mathfrak{C}_r|$, $\mathfrak{Cand} = \{\varnothing\}$
    else if ($x = r$) **then**
      while ($\mathfrak{Cand} \neq \{\varnothing\}$ and I=*false*) **do**
        choose a node n^* randomly from $\mathfrak{Cand}$.
        if $\Pi = \{n^* \cup S(n^*)\}$ are *good* candidates **then**
          Set $\Pi^* = \Pi$ and I= *true*.
        else
          Drop n^* from $\mathfrak{Cand}$ and $RC = RC - 1$.
        end if
      end while
    else
      while ($\mathfrak{Cand} \neq \{\varnothing\}$ and I=*false*) **do**
        choose a node n^* randomly from $\mathfrak{Cand}$.
        Set $S(n^*)(x) \leftarrow$ Choose up to x nodes from $S(n^*)$ according to rule 1.
        if $\Pi = \{n^* \cup S(n^*)(x)\}$ are *good* candidates **then**
          Set $\Pi^* = \Pi$ and I= *true*.
        else
          Drop n^* from $\mathfrak{Cand}$ and $RC = RC - 1$.
        end if
      end while
    end if
  end if
  if ($|\mathfrak{Cand}|$=0) and I =*false* **then**
    Update r, ℓ and the set of candidates $\mathfrak{Cand}$.
  end if
end while
return Π^*

6.1. Sizes of the detected communities

For all the datasets we ran *RANK-NUM-NEIGHS* 10 times for each node.

6.1.1. Comparison of Community sizes with the global *Louvain* method. The boxplots of the community sizes for the artificial and real networks are shown in figures 4 and 5 respectively. Figure 4 presents the results for only three sizes 1000, 5000 and 10000 (the results for the other sizes have nearly the same behavour).

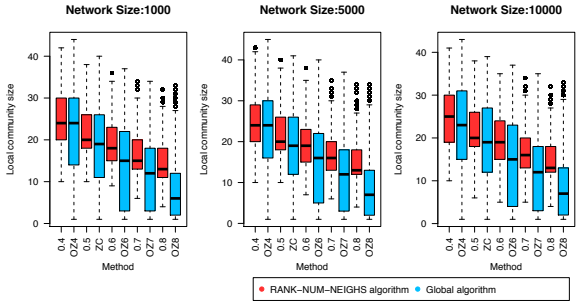

Figure 4. Community sizes for the artificial networks.

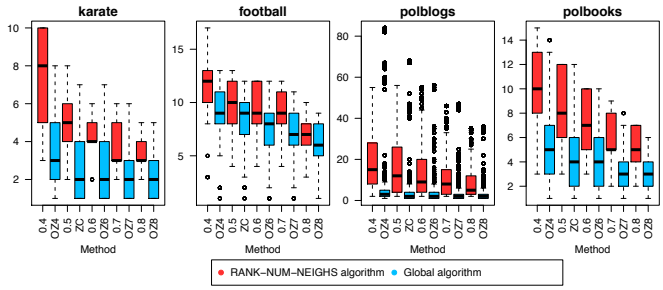

Figure 5. Community sizes for real graphs.

For each network or dataset, the figures 4 and 5 compare the community sizes obtained by our algorithm (local) to those obtained by the global criteria Zahn-Condorcet (ZC) and Owsiński-Zadrożny (OZ), whose solutions are approached by the *Louvain* method (as explained in section 4.2). To simplify we denote by OZx for $x \in [5, 6, 7, 8]$ the Owsiński-Zadrożny criterion with parameter $\alpha = x/10$. Notice that the criterion $OZ5$ is equivalent to the criterion ZC. There is a boxplot for each value of the parameter α, which ranges from 0.5 to 0.8.

Concerning the artificial networks in figure 4, clearly our algorithm returns bigger communities than those obtained by the *Louvain* method with global criteria, except for $\alpha = 0.4$. Moreover, the results obtained by our algorithm present less variability since the interquartile range of the boxplots are much smaller than those of global criteria. Concerning the real networks, we confirm once more that our algorithm returns bigger communities for the four datasets. The difference is even much more remarkable, specially for the *polblogs* dataset.

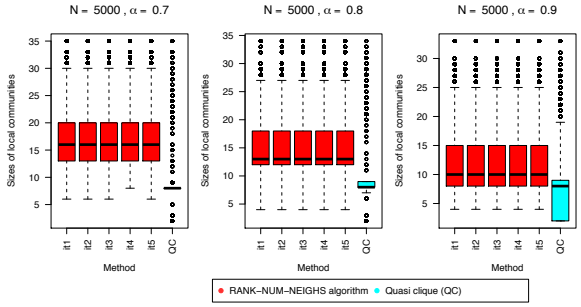

Figure 6. Comparison of size with the *QUICK* method

6.1.2. Comparison of Community sizes with the *QUICK* method. We compared the sizes of local communities obtained with our algorithm to those obtained by the *QUICK* method (see [8]) described in section 4.1. We ran our algorithm 10 times for the LFR graphs of different sizes ranging from 1000 to 8000 nodes by increments of 1000 for three values of the $alpha$ parameter: 0.7, 0.8 and 0.9.

The figure 6 shows the boxplots of the community sizes obtained by 5 iterations (labeled as itx for $x \in [1, 2, 3, 4, 5]$) of our algorithm and by the *QUICK* (QC) method. The figure 6 shows the results only for graphs of 5000 nodes, the results are quite similar for the other sizes.

The figure 6 shows clearly that the communities obtained by the method proposed in this paper are bigger than those proposed in [8]. However, this difference decreases with α as communities are smaller and closer to complete subgraphs. We remark as well that the variability increases for the *Quasi-clique* method. The figure 6 is also useful to evaluate the stability of our algorithm in terms of community sizes when comparing the boxplots of the 5 iterations.

6.2. Execution time

We executed the algorithm on artificial networks of sizes ranging from 1000 to 10000; for three different values of the mixing parameter λ: 0.1, 0.2 and 0.3; for values of the parameter α ranging from 0.4 to 0.9 by increments of 0.1. We ran the algorithm 10 times for all the nodes of each graph. The computer used for the experimentations has a Quad Core processor running at 3.33 GHz and 15GB RAM. The algorithm is written in python. The figure 7 shows the execution time in seconds for each entire network.

The figure 7 shows that the execution time of the algorithm is quite stable. The execution time seems to increase smoothly with the network size. It increases with the mixing parameter λ. The time decreases with the parameter α, since the communities become smaller when α increases as it becomes more difficult for a node to verify the rule to enter the local community. One reason that explains the rapidity of this method is that at each iteration it is possible to add more than one node, so the nodes become saturated fast.

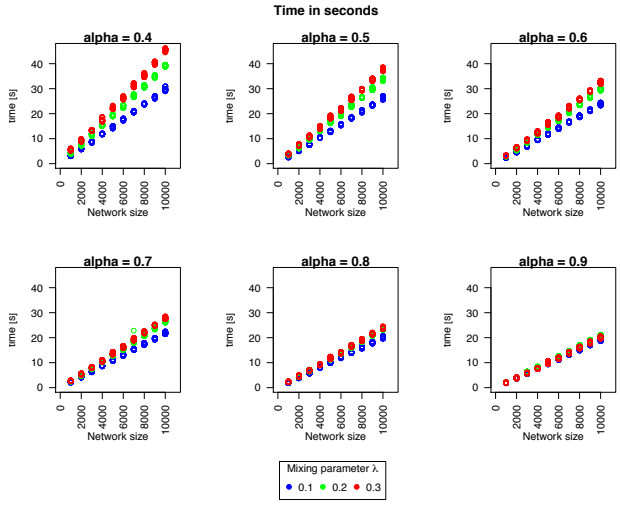

Figure 7. Execution time to detect the local community of all nodes of artificial LFR graphs.

We do not compare this execution time to that of the *QUICK* method as this last one took more than one day for a graph of 9000 nodes.

7. Conclusions and perspectives

In this paper a novel method called *RANK-NUM-NEIGHS* for the problem of mining the maximal local $\alpha-$consensus community of a given node of a network has been presented. Starting from a baseline method, the next methods are introduced to address some shortcomings not solved by the previous ones.

The method we propose in this paper prioritizes the nodes who have the biggest number of neighbors in the neighborhood of the community, over the gain. Besides that, It allows to enter more than one node at each iteration. The method verifies the chosen nodes guarantee the respect of the majority rule.

RANK-NUM-NEIGHS gives good results in terms of size, stability and execution time. It also performs better in terms of community size than the existing method for mining quasi-cliques called *QUICK* and than global community detection methods.

Concerning the perspectives, this problem can be the building block of more complex applications where the internal density of community plays an important role. For example, in friend recommandation, the missing links of an α-consensus community can be recommended. In churn prediction, the communities can better model the notion of closest friends as studied in [24].

References

[1] S. Fortunato, "Community detection in graphs." in *Physics Reports*, vol. 486, 2010, pp. 75–174.

[2] Y. Asahiro, R. Hassin, and K. Iwama, "Complexity of finding dense subgraphs," *Discrete Appl. Math.*, vol. 121, no. 1-3, pp. 15–26, Sep. 2002. [Online]. Available: http://dx.doi.org/10.1016/S0166-218X(01)00243-8

[3] A. Clauset, "Finding local community structure in networks," in *Physical Review*, vol. 72, 2005, p. 026132.

[4] F. Luo, J. Z. Wang, and E. Promislow, "Exploring local community structure in large networks," in *WI'06.*, 2006, pp. 233–239.

[5] J. P. Bagrow, "Evaluating local community methods in networks," in *Journal of Statistical Mechanics*, 2008, p. 05001.

[6] J. Chen, O. R. Zaiane, and R. Goebel, "Local communities identification in social networks," in *ASONAM*, 2009, pp. 237–242.

[7] B. Ngonmang, M. Tchuente, and E. Viennet, "Local communities identification in social networks," *Parallel Processing Letters*, vol. 22, no. 1, Mar. 2012.

[8] G. Liu and L. Wong, "Effective pruning techniques for mining quasi-cliques," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, W. Daelemans, B. Goethals, and K. Morik, Eds. Springer Berlin Heidelberg, 2008, vol. 5212, pp. 33–49.

[9] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks." in *Journal of Statistical Mechanics: Theory and Experiment*, 2008, p. 10008.

[10] M. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E.*, vol. 69, no. 2, 2004.

[11] R. Campigotto, P. Conde-Céspedes, and J. Guillaume, "A generalized and adaptive method for community detection," *CoRR*, vol. abs/1406.2518, 2014. [Online]. Available: http://arxiv.org/abs/1406.2518

[12] C. A. M. d. Condorcet, "Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix," *Journal of Mathematical Sociology*, vol. 1, no. 1, pp. 113–120, 1785.

[13] C. Zahn, "Approximating symmetric relations by equivalence relations," *SIAM Journal on Applied Mathematics*, vol. 12, pp. 840–847, 1964.

[14] F. Marcotorchino and P. Michaud, *Optimisation en Analyse ordinale des données*. Paris: Masson, 1979.

[15] P. Conde-Céspedes, "Modélisations et extensions du formalisme de l'analyse relationnelle mathématique à la modularisation des grands graphes," Thèse de doctorat, Université Pierre et Marie Curie, 2013.

[16] P. Conde-Céspedes, J. Marcotorchino, and E. Viennet, "Comparison of linear modularization criteria using the relational formalism, an approach to easily identify resolution limit," *Revue des Nouvelles Technologies de l'Information*, vol. Extraction et Gestion des Connaissances, RNTI-E-28, pp. 203–214, 2015.

[17] J. Owsiński and S. Zadrożny, "Clustering for ordinal data: a linear programming formulation." *Control and Cybernetics*, vol. 15, no. 2, pp. 183–193, 1986.

[18] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, no. 4, Oct. 2008.

[19] A. Lancichinetti and S. Fortunato, "Community detection algorithms: A comparative analysis," *Phys. Rev. E*, vol. 80, p. 056117, Nov 2009.

[20] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, 1977.

[21] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821–7826, June 2002.

[22] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 u.s. election," in *Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem*. NY, USA: ACM, 2005, pp. 36–43.

[23] V. Krebs. (2004) Books about US politics. [Online]. Available: http://www.orgnet.com/

[24] B. Ngonmang, E. Viennet, and M. Tchuente, "Churn prediction in a real online social network using local community analysis," in *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012, Istanbul, Turkey, 26-29 August 2012*, 2012, pp. 282–288. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ASONAM.2012.55