Noname manuscript No.
(will be inserted by the editor)

An efficient method for mining the Maximal α-quasi-clique-community of a given node in Complex Networks

A method for mining the Maximal α-quasi-clique-community of a node

Patricia CONDE-CESPEDES[(1),(2)] · **Blaise NGONMANG**[(2)] · **Emmanuel VIENNET**[(2)]

Received: date / Accepted: date

Abstract Detecting communities in large complex networks is important to understand their structure and to extract features useful for visualisation or prediction of various phenomena like the diffusion of information or the dynamic of the network. A community is defined by a set of strongly interconnected nodes. An α-quasi-clique is a group of nodes where each member is connected to more than a proportion α of the other nodes. By construction, an α-quasi-clique has a density greater than α. The size of an α-quasi-clique is limited by the degree of its nodes. In complex networks whose degree distribution follows a power law, usually α-quasi-cliques are small sets of nodes for high values of α. In this paper, we present an efficient method for finding the maximal α-quasi-clique of a given node in the network. Therefore, the resulting communities of our method have two main characteristics: they are α-quasi-cliques (very dense for high α) and they are local to the given node. Detecting the local community of specific nodes is very important for applications dealing with huge networks, when iterating through all nodes would be impractical or when the network is not entirely known. The proposed method, called RANK-NUM-NEIGHS (RNN), is evaluated experimentally on real and computer generated networks in terms of quality (community size), execution time and stability. We also provide an upper bound on the optimal solution.

(1)LISITE - ISEP
10 rue de Vanves
92130 Issy-les-Moulineaux - France
Tel.: +33-01.49.54.52.41
E-mail: patricia.conde-cespedes@isep.fr

(2) L2TI-Université Paris 13
99 Avenue Jean Baptiste Clément
93430 Villetaneuse-france
E-mail: firstname.lastname@univ-paris13.fr

Keywords Community detection algorithms · Maximal α-quasi-clique · Local community detection · Complex networks · Density.

1 Introduction

A network is usually a complex system composed of a set of entities, usually called vertices or nodes, connected by links, also called edges. In the particular case where the entities are people, the system is called a social network. However, a lot of natural phenomena can be modeled by networks; those *complex networks* share important characteristics (degree distribution, local clustering) and often exhibit community structures. The study of the communities has attracted a lot of attention (see [24]). Detecting communities in large complex networks is important to understand their structure and allows to extract features useful for visualisation [33] or prediction of various phenomena like the diffusion of information or the dynamic of the network or social recommendation [23].

A community is defined by a set of strongly interconnected nodes. The density of links measures the strength of the relationships in the community[1]. However, most popular community detection methods do not guarantee anything about the density of the resulting communities (for instance, when using the Newman-Girvan modularity [38], the density of the output communities can become very low due to its resolution limit [25]). Complete cliques[2] are sets of

[1] The density of links δ of a graph G with $|E|$ edges et $|V|$ nodes is given by $\frac{2|E|}{|V|(|V|-1)}$.

[2] A complete clique is a set of node such as every two distinct nodes are connected to each other.

nodes with maximal density. However, the size of a clique is limited by the degree of its nodes. Particulary, in complex networks whose degree distribution follows a power law, the cliques can be very small or even trivial, such as pair of nodes or triangles. Moreover, the search of the largest clique in a graph is a well known problem studied in graph theory called *the maximum clique problem (MCP)* (see [27] and [11] for a survey). This led to the relaxation of the concept of a complete clique to an *almost complete subgraph*, also called *quasi-clique* (the interested reader can refer to [32], [43], [48] and [29] for surveys).

An α-quasi-clique (for $0 < \alpha < 1$) is a group of nodes where each member is connected to more than a proportion α of the other nodes[3]. Consequently, an α-quasi-clique has a density greater than α. By choosing α close to 1, an $\alpha-$quasi-clique becomes an almost complete clique. Considering an $\alpha-$quasi-clique instead of a complete clique can be preferable for applications where interaction between members of the community does not need to be direct and could be successfully accomplished through intermediaries. In the following, we will also call an α-quasi-clique community an α-consensus community as we did in [19], a preliminary version of this work.

Mining all the maximal α-quasi-cliques of a network is NP-complete [28] [4]. 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 existing α-quasi-cliques. In some cases, the network can be so large that one can have only local information about some nodes or one can be only interested in the community of a particular node in the network. Moreover, detecting the local communities of specific nodes may be very important for applications dealing with huge networks, when iterating through all nodes would be impractical or when the network is not entirely known. The detection of the community of a given node of interest is also called *local* community detection problem.

In this paper, we present an efficient method for approaching the $\alpha-$*quasi-clique community of a given node problem*. Therefore, the resulting communities have two main properties:

1. they are α-quasi-cliques.
2. they are local to the given node.

The maximal quasi-clique problem has already been addressed, however, all previous studies try to find a subgraph with no reference to a given node. To the best of our knowledge we are the first to address the maximal α-quasi-clique problem from a local-community point of view. Our method, called *RANK-NUM-NEIGHS* (RNN), is evaluated experimentally on real and artificial complex networks in terms of quality (community size), execution time and stability. We also provide an upper bound on the optimal solution and a practical application. The experiments show that it provides superior results than the existing methods in terms of time and stability.

This paper is organised as follows: Section 2 presents definitions and notations. Then, Section 3 presents the problem formulation. Section 4 gives an overview of related works. Section 5 draws our solution to this problem. In Section 6 the proposed solution is evaluated and the results are discussed. Besides, the application to a real network demonstrates the practical usefulness of the presented approach. In Section 7 we present a bound on the optimal solution. Finally, Section 8 draws some conclusions and perspectives.

2 Definitions and notations

A graph $G = (V, E)$, is defined by V the set of vertices or nodes, and E the set of edges or links, formed by pairs of vertices. In this paper we consider undirected graphs, where edges are not oriented. The neighborhood $\Gamma(u)$ of a node u is the set of nodes v such that $(u, v) \in E$. The degree of a node u, denoted $d(u)$, is the number of its neighbors, i.e. $d(u) = |\Gamma(u)|$. An $\alpha-$quasi-clique is defined as follows:

Definition 1 **$\alpha-$quasi-clique or $\alpha-$consensus community**
Given an undirected graph $G(V, E)$, and a parameter α with $0 < \alpha < 1$, an $\alpha-$quasi-clique or $\alpha-$consensus community is the subgraph induced by a subset of the node set $C \subseteq V$ if the following condition holds:

$$|\Gamma(n) \cap C| > \alpha(|C| - 1), \forall n \in C. \quad (1)$$

Equation (1) implies that each node in the quasi-clique C must be connected to more than a proportion α of the other nodes. In the following, we will call Equation (1) the **rule of an $\alpha-$quasi-clique**. This rule constitutes a lower bound on the minimal internal

[3] This definition of an α-quasi-clique is not unique. Most authors define an α-quasi-clique as a set of nodes that have a density greater than α, see for instance [1]. The definition considered in this paper constitutes a relative relaxation of a complete clique as it depends on the size of the quasi-clique.

connections of each node. Notice that for $\alpha = 1$ an $\alpha-$quasi-clique is a complete clique. Notice also that an $\alpha-$quasi-clique has a density greater than α (see appendix for the proof).

In the literature, one can find other definitions of the so-called $\alpha-$quasi-clique. For instance, in [34], the authors considered a nearly similar definition. The only difference is that in their definition the condiction (1) has an equality sign. Thus, their definition allows each node in the quasi-clique to be exactly connected to a proportion α of the other nodes. Other variants are:

1. **Definition of an $\alpha-$quasi-clique based on the density:** given a graph $G = (V, E)$, a subset of vertices $C \subseteq V$ is an $\alpha-$quasi-clique if the edge density of the induced subgraph is at least equal to a threshold parameter $\alpha \in (0, 1)$. The edge density of the set C is defined as $\delta(C) = \frac{2e[C]}{|C|(|C|-1)}$, where $e[C]$ is the number of edges induced by C.
   This is the most common used definition of $\alpha-$quasi-clique. It was considered by many authors, for instance [1][42] [47] [14], and it is less strict than definition 1 as it concerns the connections of the whole subset rather the connections of each node individually. Therefore, an $\alpha-$quasi-clique based on definition 1 is also an $\alpha-$quasi-clique in the sense of this definition.

2. **Definition of a (λ, γ)-quasi-clique:** in [12], the authors defined what they called a (λ, γ)-quasi-clique as follows:
   Given two parameters λ and γ with $0 \leq \lambda \leq \gamma \leq 1$, the subgraph induced by a subset of the node set $C \subseteq V$ is (λ, γ)-quasi-clique if, and only if, the following two conditions hold:
   $$\forall n \in C \quad |\Gamma(n) \cap C| \geq \lambda(|C| - 1) \text{ and}$$
   $$e[C] \geq \gamma \frac{|C|(|C| - 1)}{2}$$
   This definition is more strict that definition 1 as it imposes a lower bound for the density of the whole subset. However, by taking $\gamma = \lambda$ this definition is almost equivalent to definition 1, except for the equality sign in the first condition. Indeed, this condition allows the number of internal connections of a node to be *equal* or *greater* than a proportion α of the other nodes. Whereas, definition 1 is a little more strict by not admitting equality.

In this paper, we consider the definition 1 of an $\alpha-$quasi-clique. There are important reasons that support our choice.
The first variant based on the edge density of the whole $\alpha-$quasi-clique does not garantee that *each* node is strongly connected to the other nodes. For instance, consider the subgraphs in Figure 1. Both subgraphs contain 12 nodes and 42 edges, so the edge density is $\delta \approx 0.63$. They are both $0.6-$quasi-cliques according to the definition variant 1. However, only the graph on the right is a $0.6-$quasi-clique according to definition 1 as each node is connected to 7 out of 11 nodes. Since the nodes 10, 11 and 12 (in red) are weakly connected to the other nodes of the graph on the left, they do not verify the condition (1). Moreover, intuitively only the nodes in blue can form a community. One can also verify this situation by looking at the adjacency matrix. Indeed, nodes in red are not well connected to the other nodes whereas the number of connections is rather equitable for all the nodes of the graph on the right.

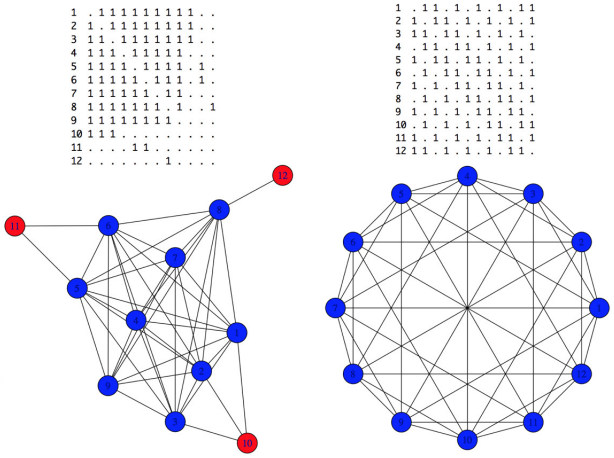


Fig. 1 Two graphs of 12 nodes and 42 edges and edge density 0.63. However, only the graph on the right is a $0.6-$quasi-clique community according to definition 1. Indeed, since nodes in red of the graph on the left are weakly connected to the other nodes, they do not verify the condition (1).

If we consider the second variant of the $\alpha-$quasi-clique definition, given by [12], with $\gamma = \lambda$ and the definition given by [34], we realize they are equivalent. However, both differ from our definition 1 in the equality sign of the constraint (1). We decided not to allow equality in order to guarantee the absolute majority in the internal connections of each node when $\alpha \geq 0.5$. Certainly, when $\alpha = 0.5$, accepting equality would lead to accept a node to have as many connections as non-connections in the quasi-clique. However, since we are looking for strongly connected dense sub-graphs, we want to ensure each node has more connections than non-connections.

In summary, by considering definition 1 we garantee that the detected communities are robust, con-

tain strongly connected nodes and have an edge-density greater than α.

3 Problem formulation

Since the size of an $\alpha-$quasi-clique is limited by the degree of its nodes, for complex networks whose degree distribution follows a power law, mining for the $\alpha-$quasi-clique community of specific nodes with low degree can lead to trivial solutions, such as pairs of nodes or triangles. Indeed, those are $\alpha-$quasi-cliques or even complete cliques as they achieve maximal density. Such trivial communities are not interesting for applications. Therefore, the purpose is to find quasi-cliques of maximal size.

In the literature, one can find several methods for detecting the maximal α-quasi-clique of an entire network: we mentioned some of them in Section 2. In this paper we tackle a different problem. We aim to find an $\alpha-$quasi-clique of maximal cardinality containing a given node. This problem can be formulated as follows[4]:

Problem 1 the maximal $\alpha-$quasi-clique community (or quasi-clique) of a given node problem

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 $\alpha-$quasi-clique or $\alpha-$consensus community $C(n_0)$ containing n_0, mathematically:

$$\begin{array}{ll} \underset{C}{\text{maximize}} & |C| \\ \text{subject to} & n_0 \in C \\ \text{and} & |\Gamma(n_i) \cap C| > \alpha(|C| - 1), \forall n_i \in C. \end{array} \quad (2)$$

In summary, the communities detected by the method proposed in this paper have two main characteristics:

1. They are α-quasi-cliques (according to definition 1), therefore each node is highly connected to the other nodes in the community and
2. they must contain a given node of interest. That is why the problem we try to solve is a particular case of the *local* community detection problem.

[4] Notice that we used the word *maximal* instead of *maximum*. In graph theory a maximal clique is a clique which is not a proper subset of another clique whereas a maximum clique is a clique of the maximum cardinality in the graph. Since we aim to find $\alpha-$quasi-cliques containing a given node of interest we are looking for *maximal* $\alpha-$quasi-cliques instead of for the *maximum* $\alpha-$quasi-clique.

Notice that a node, can belong to more than one maximal $\alpha-$quasi-clique community. Therefore the problem 1 can have multiple solutions.

4 Related works

We are the first to address the *maximal $\alpha-$quasi-clique community of a given node problem* to the best of our knowledge. Either the existing methods for local community detection do not constraint the output communities to be $\alpha-$quasi-cliques (see for example [16], [35], [5], [15], [39], [21]) or the existing methods for mining $\alpha-$quasi-cliques are not local to a given node (such as [34], [12]). We call non local methods *global*, opposite of *local*, as they consider the entire network to detect $\alpha-$quasi-clique communites (according to definition 1). We consider relevant to compare the results of our method to the results of *global* methods since there is a guarantee on the high density of the resulting communites. In the literature, we found three such methods:

- **The *QUICK* method** [34]: It was designed to efficiently extract all the maximal $\alpha-$quasi-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 $\alpha-$quasi-cliques of increasing sizes that contain n_0. At each following step, it moves to the next unexplored node and builds α-quasi-cliques that do not contain already explored nodes. Because the search space is exponential on the number of nodes, *QUICK* uses several effective pruning techniques based on the degree of the nodes, the diameters of the constructed quasi-cliques to prune unqualified vertices as early as possible. 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.

- **The *Louvain method* adapted to Zahn-Condorcet and Owsiński-Zadrożny criteria, also called *the generic Louvain method***[13]: The *generic Louvain method* is an extended version of the original *Louvain method* introduced in [10] to optimize the Newman-Girvan modularity [38]. Because of its rapidity, in [13] the authors proposed to adapt the Louvain method to other seven quality functions and called this extended version the *generic Louvain method*. Two quality functions of particular interest for us are the Zahn-Condorcet (ZC)[20], [51], [36] and the Owsiński-Zadrożny

(OZ)[41] criteria. Indeed, it was shown in [17], [18] that the partition corresponding to the optimal solution of the Zahn-Condorcet problem is composed of 0.5-quasi-clique communities. The Owsiński-Zadrożny criterion is a generalization of the Condorcet problem where the optimal partition can contain only $\alpha-$quasi-cliques (see [17] and [18] for proof). Although the *generic Louvain* method is very fast compared to many global community detection methods, it does not intend to maximize the size of alpha-cliques. Therefore, it is not competitive with the method proposed in this paper as it does not address problem 1. The performance in terms of quality of the generic Louvain method can be seen in reference [19].

– **The *RLS-DLS* method** [12]: This method is an heuristic for the problem of finding maximum $(\lambda,\gamma)-$quasi-cliques. The definition of a $(\lambda,\gamma)-$quasi-clique was given in Section 2. The *RLS-DLS* method aims at extending the work done by other authors in efficient clique algorithms, in particular *Reactive Local Search (RLS)* and *Dynamic Local Search for Maximum Clique (DLS-MC)*. Both algorithms are based on stochastic local search methods. The *DLS-MC* algorithm for the maximum clique problem is based on the idea of assigning penalties to nodes that are selected to be part of a clique [45]. The *RLS* algorithm uses a reactive mechanism to control the amount of diversification during the search process by means of prohibitions [8], [7].

5 The proposed method

This section presents the proposed heuristic to approach the solution of the *maximal $\alpha-$quasi-clique community of a given node problem*, the *RANK-NUM-NEIGHS (RNN)* algorithm. First, we will discuss about the scalability of the approach. Second, we will present a general greedy scheme for local community detection. Next, we will present introduce and describe the *RNN* algorithm.

5.1 Scalability

An important advantage of our approach is its scalability. In fact, consider the following theorem concerning the diameter of an $\alpha-$quasi-clique (see appendix for proof):

Theorem 1 *Let C be an $\alpha-$quasi-clique for $\alpha \geq 0.5$, then the diameter of C is at most 2.*

Theorem 1 implies that for $\alpha \geq 0.5$ the nodes in the optimal solution of problem 1 will be located at most at a distance 2 of the starting node n_0. Indeed, dense components naturally have small diameters.
Although the proposed method in this paper works for any $0 < \alpha < 1$, in the following we will focus on values of $\alpha \geq 0.5$ because we are interested in communities of high density.
If $\alpha \geq 0.5$ then Theorem 1 holds. In this case, to approach the problem 1 we do not need the whole graph as input, but only the first and second neighborhood (the neighbors $\Gamma(n_0)$ and the neighbors of the neighbors $\Gamma(\Gamma(n_0))$) of n_0[5]. In the following, given a node n, we will denote $\Gamma_{1,2}(n)$ the subgraph induced by n, its first and its second neighborhoods. That is:

$$\Gamma_{1,2}(n) = G(n \cup \Gamma(n)) \cup \Gamma(\Gamma(n))$$

5.2 A General greedy scheme for local community detection

Our method is based on a greedy and iterative algorithm. We will denote $C(n_0)$ the resulting community of the given node n_0. The local community will start with one node, n_0. At each iteration the set of nodes identified as members of $C(n_0)$ will be denoted D, and the set of all neighbors of nodes in D that do not belong to D will be denoted S (see Figure 2).

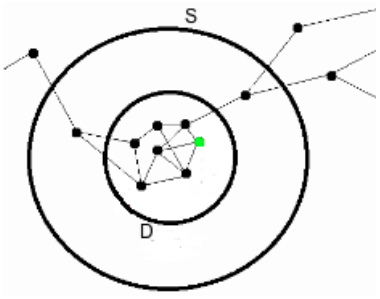


Fig. 2 A node n_0 (in green), the members of its local community D and the neighbors of nodes in D: S

The algorithm takes as inputs n_0, $\Gamma_{1,2}(n_0)$ and the parameter α. At each iteration, given D and S the algorithm selects the *best* nodes Π^* from S to enter D to according to a given criterion function F. We mean

[5] If $\alpha < 0.5$, Theorem 1 does not hold anymore, then the input of the algorithm will not be limited to the second neighborhood, but the whole graph specially for alpha small.

by *best* nodes the nodes that will make it possible to obtain a community of maximal size while respecting the contraint (1) of an $\alpha-$quasi-clique in order to approach the solution of problem 1. The algorithm stops when there is no possibility of improvement in the size of the community (see algorithm 1).

Algorithm 1 General greedy scheme for local community detection.

Require: A node n_0, $\Gamma_{1,2}(n_0)$ and a parameter α.
Ensure: A local $\alpha-$quasi-clique community $C(n_0)$ containing n_0.
1: Initialize $D = \{n_0\}$, $S = \Gamma(n_0)$.
2: **while** It is possible to add nodes from S to D **do**
3: Select the *best* nodes Π^* from S to enter D according to a given selection criterion function F.
4: Update D, S.
5: **end while**
6: **return** $C(n_0) = D$

Now, let us define explicitly the condition in the while loop in algorithm 1, line 2. This condition is *true* if it exits the possibility of improvement in the size of the community, in other words, the possibility to add at least one node from S to D, such that the resulting community is an $\alpha-$quasi-clique. To state this condition we need to give some definitions.

Consider the graph in Figure 3. Suppose we are mining for the maximal $0.6-$quasi-clique of node 0. Suppose that at a given iteration we obtain $D = \{0, 1, 2, 3\}$ and $S = \{4\}$. For the next iteration, if node 4 is added to D all the nodes in the resulting community $\{0, 1, 2, 3, 4\}$ will verify the rule of an $0.6-$quasi-clique, except node 0, because it will be connected to only 2 nodes out of 4, less than 60%.

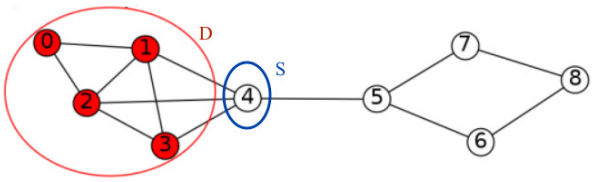


Fig. 3 When mining for the maximal $0.6-$quasi-clique of node 0, if node 4 is added to D the resulting community will not be an $0.6-$quasi-clique as node 0 will not verify the rule of an $\alpha-$quasi-clique (1)

Certainly, given an $\alpha-$quasi-clique D, if a new node is added to D then the rule (1) would become more strict to be satisfied, because every node must be connected to more nodes in the resulting community. In other words, increasing the size of the community causes the rule of an $\alpha-$quasi-clique to become more strict. Indeed, the growth of the community is limited by the *internal degree* of the existing nodes in D. Therefore, before considering the addition of a new node it is necessary to ensure that the already existing nodes will *still* verify the rule after the addition. Let us formalize all these concepts.

Given a node n, we call *the internal degree of* n the number of internal connections of n to nodes in D and we denote it $d^{in}(n)$.

For example, in Figure 3, the internal degree of nodes $D = \{0, 1, 2, 3\}$, are 2, 3, 3 and 2 respectively, and the community size is 4. Since every node must be connected to more than 60% of the other nodes, 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. These explanations lead to the following theorem (see appendix for proof):

Theorem 2 *Given a node n with internal degree $d^{in}(n)$ the maximum possible size of an $\alpha-$quasi-clique or α-consensus community it can belong to, denoted $D^{max}(n)$, is given by:*

$$D^{max}(n) = \left\lceil \left(\frac{d^{in}(n)}{\alpha} \right) \right\rceil. \quad (3)$$

Likewise, given an $\alpha-$quasi-clique or α-consensus community D the minimal internal degree a node in D can have, denoted d^{min} is:

$$d^{min} = \lfloor \alpha(|D| - 1) \rfloor + 1. \quad (4)$$

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 $\alpha-$quasi-clique 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, while keeping an $\alpha-$quasi-clique, the maximum size of the resulting community must not exceed:

$$D^{max} = \left\lceil \left(\frac{d_{min}}{\alpha} \right) \right\rceil \quad (5)$$

Definition 2 A saturated node
Given a community D, a node $n \in D$ is *saturated* if $d^{in}(n) = d^{min}$ or $|D| = D^{max}(n)$.

For the example in Figure 3, $d_{min} = 2$, $D^{max} = 4$, therefore, nodes 0 and 3 are saturated.

From theorem 2 and definition 2 we can state the following corollary (proof is omitted):

Corollary 1 *Given a community D, if there is a saturated node n, the following equalities take place:*

$$d^{in}(n) = d_{min} = d^{min} \text{ and } D^{max}(n) = D^{max} = |D|.$$

On one hand, if no node is saturated it is possible to choose any node from S to enter D as long as it verifies the rule of an $\alpha-$quasi-clique. On the other hand, 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 common neighbors of all saturated nodes (so, their internal degree can increase). However, if a node n is saturated and its degree is equal to its internal degree ($d(n) = d^{in}(n)$) there is no possibility of improvement because n has no neighbors in S.

Henceforth, we will say that a node is *supersaturated* if it is saturated and its degree is equal to its internal degree. For example, in Figure 3, node 0 is supersaturated. In this situation, there is no possibility of improvement, so the algorithm must stop. Let *ssat* be a boolean variable which is set to *true* if at least one node in D is supersaturated and *false* otherwise. Therefore, if *ssat* is *true* the algorithm must stop.

In the following, if a subset of nodes denoted $\Pi \in S$ is such that after being added to D all nodes in the resulting community verify the constraint of an $\alpha-$quasi-clique (1), we will say that nodes in Π are *good* candidates to enter D. If there are no *good* candidates, the algorithm must stop.

At each iteration the algorithm will select the *best* nodes from S to enter D according to a given criterion function F (line 3 of algorithm 1). We will denote these selected nodes Π^*. If this function returns the empty set the algorithm must stop.

Now, let us go back to the condition in the while loop in line 2 of algorithm 1. The condition must be *false* if and only if the algorithm must stop. Therefore, the condition in the while loop would be compound:

while ($|S| > 0$ and $ssat = false$ and $\Pi^* \neq \varnothing$)

In order for this condition to be verified for the first iteration, *ssat* and Π^* are initialized to *ssat* =*false* $\Pi^* = \{n_0\}$ in line 1 of algorithm 1.

The proposed method in this paper is based on the *general greedy scheme for local community detection*. In the next section, we will decribe the criterion function F used in line 3 of algorithm 1. This function characterizes our method.

5.3 The *RANK-NUM-NEIGHS* method

5.3.1 Introduction and motivation

In [19], we presented the *RANK-GAIN+ (RG+)* algorithm. This algorithm is based on the *general greedy scheme* described in Section 5.2 and its selection criterion F is based on the gain.

Definition 3 Gain
For any node n in the neighborhood S of the local community D and connected to ℓ_n nodes in D, the gain resulting from the addition of n to D, denoted g_n, is given by:

$$g_n = \ell_n - \alpha|D|. \quad (6)$$

For any node n to respect the rule of an $\alpha-$quasi-clique, g_n must be positive. The *gain* measures the strenght of the connection between n and D. A high value of gain contributes to increase the density of the community.

In [19], we discussed about some shortcomings not solved by the *RG+* algorithm. Indeed, the gain is not the most important variable when selecting a node to enter D. At this point, let us remind the problem of the maximal $\alpha-$quasi-clique community (see equation (2)):

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

So, the purpose is to maximize the community size, not directly the density, as long as all the nodes verify the rule (1). By choosing a high value of α, the verification of this rule will garantee a density higher than α (see section 2 for details). The gain is certainly important as it contributes strongly to the value of the density, however, as mentionned in the previous sections, a very dense small community lacks of interest and interpretability. In other words, there is a trade-off between the size and the density of a community.

Indeed, consider the *gain* the criterion to select nodes can lead to sub-optimal solutions consisting in high dense communities of very small size.

For instance, consider the graphs in figure 4. Let us suppose we are looking for the maximal 0.55-quasi-clique community of node 0 for graph a). The optimal solution is $D^{opt} = \{0, 1, 2, 3, 4, 5\}$ (nodes in red in the graph on the left). At a given iteration, we have $D = \{0, 1, 2\}$ (as shown in the graph on the right). 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 and the algorithm will stop and return the suboptimal solution $D = \{0, 1, 2, 6\}$. A similar problem would happen for the graph b) when looking for the community of node 1. The optimal solution is a community of size 6 (nodes in red of the graph on the left). However, the *RG+* algorithm will return a community of size 4 (nodes in red of the graph on the right). Indeed, once nodes 1, 2, 3 and 4 have entered the community, the algorithm will judge nodes 5 and 6 as *bad* candidates as their gains are null.

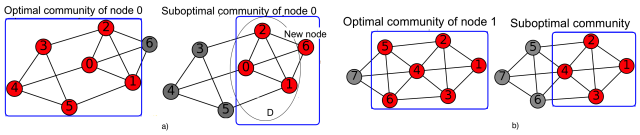


Fig. 4 a) *RG+* will choose 6 as the best candidate to join D. b) *RG+* will return a local community of size 4 for node 1 (left) whereas the maximal $\alpha-$ consensus community is of size 6 (right)

To overcome these shortcomings, we introduced the *RANK-NUM-NEIGHS (RNN)* algorithm, the algorithm we propose in this paper. The *RNN* algorithm is also based on the *general greedy scheme* and it distinguishes from the *RG+* algorithm in the selection criterion F.

The *RNN* method selection criterion is based on the number of common neighbors with the local community. 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 as stated in lemma 1 (see appendix for the proof).

Lemma 1 *Given a node $n \in S$ if n is added to D the gain of all its neighbors will increase by $(1 - \alpha)$.*

Besides, the more neighbors the selected nodes have, the more possibilities there are to have *good* candidates for the next iteration. Therefore, there will be more possibilities to enlarge the community and approach the optimal solution of the *maximal $\alpha-$quasi-clique community of a given node problem* described in (1).

The *RNN* method considers the gain but in second place. Certainly, the gain is certainly important as it contributes strongly to the value of the density, however, as mentioned in the introduction, a very dense small community lacks of interest and interpretability.

5.3.2 Description of the RNN method: the selection criterion F

The selection criterion F used by the *RANK-NUM-NEIGHS (RNN)* algorithm in the *general greedy scheme* (see algorithm 1) has two main characteristics:

1. It prioritizes the number of common neighbors with the local community over the gain when choosing a new node (or more nodes) to enter the local community.
2. At each iteration either none, one node or a set of nodes might enter the local community.

We call *add_nodes()* this selection function. It is described in algorithm 2. It takes as parameters the local community D, its neighborhood S and the parameter α. It returns a set of *good* nodes Π^* from S to add to D. It uses two functions:

1. Function *rank_by_neighbors()*: this function returns a ranking of nodes in S according to their common neighbors with the community. It takes as input parameter the induced subgraph by nodes in S, denoted $G(S)$. For example, consider the graph in Figure 5, $S = \{3, 4, 5, 6\}$. The function *rank_by_neighbors* will rank first nodes $\{3, 4, 5\}$ because they have 2 neighbors in S and the second place will be for node $\{6\}$ since it has no neighbors in S.
2. Function *rank_by_gain()*: given a set of nodes, this function returns a ranking for all these nodes according to their gain if added to D (Equation (6)). For example, for the graph of Figure 5, $D = \{0, 1, 2\}$, $S = \{3, 4, 5, 6\}$ and $\alpha = 0.5$. The function *rank_by_gain* will rank first node $\{6\}$ as its gain is 0.5 and the second rank will be for nodes $\{3, 4, 5\}$ as their gain is -0.5.

The Figure 6 shows an example of the calculation of the outputs *rank_by_neighbors()* and *rank_by_gain()* functions for $\alpha = 0.5$. First, nodes are ranked according to their number of neighbors in S (represented by the green edges). This is the first criterion to select a

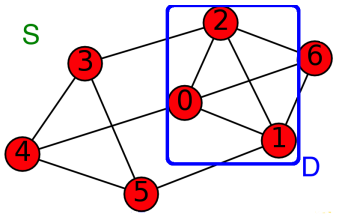


Fig. 5 Illustration of the functions *rank_by_neighbors()* and *rank_ by_gain()* used by the *add_nodes* function.

new node. Second, if ties, nodes are ranked according to their gain, that is, the number of connections with D (represented by the red edges). For the graph in the figure, the function *rank_by_neighbors()* will rank node 3 first as it has 3 neighbors in S, next 1, 2 and 4 (two connections), then 5 (1 connection) and finally 6 (0 connections). Ties between nodes 1, 2 and 4 will be broken by the *rank_by_gain()* function given in equation (6). For the example, node 1 has the greatest gain followed by nodes 2 and 4. Therefore, the algorithm will consider the nodes in the following order: 3, 1, 2 and 4 (or 4 and 2 ties will be broken randomly), 5 and finally 6 (see the table of Figure 6). When considering each node 4 situations can take place. These situations are explained in algorithm 2 and schematized in Figure 7.

The function *add_nodes* is described in algorithm 2. First, the following variables are initialized (line 1 of algorithm 2):

- RC: the number of remaining candidates, at the begining there are as many candidates as nodes in S.
- $Rank$: the ranking of nodes in S obtained by the *rank_by_neighbors()* function.
- r: the *current rank* of number neighbors, it is initialized to the highest value.
- $\mathfrak{C}_r$: the set of nodes whose rank is r.
- $Gain$: the ranking of nodes in $\mathfrak{C}_r$ according to their gain, calculated by the *rank_by_gain()* function.
- g: the current gain, initialized to the highest gain found in the ranking$Gain$.
- $\mathfrak{Cand}$, 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.

Next, (line 2-40 of algorithm 2) as long as there are still candidates ($RC > 0$) and a *good* set of nodes has not been found yet the function will continue to search for the *best* nodes. According to the values of current rank r and current gain g, 4 situations can take place:

- **Situation 1** (lines 3-12 of algorithm 2): $g > 0$, the gain of n^* is positive: If all nodes in D verify the rule of an $\alpha-$quasi-clique after the addition of n^* (n^* is a *good* candidate), then the function returns $\Pi^* = \{n^*\}$, otherwise n^* is dropped from the list of candidates $\mathfrak{Cand}$ and RC decreases by one. This process is repeated until either a *good* node has been found 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 of an $\alpha-$quasi-clique. In this case, the function tests if it is possible to let it enter simultaneously with all or some of its neighbors in S. The number of connections n^* will need is announced in theorem 3 (see appendix for proof).

Theorem 3 *Given an $\alpha-$quasi-clique D, its neighborhood S and a node $n \in S$ with negative gain, to obtain a positive gain the number of additional connections that n needs, denoted x, is:*

$$x = \left\lfloor \left(\frac{\alpha|D| - \ell_n}{(1-\alpha)} \right) \right\rfloor + 1 \qquad (7)$$

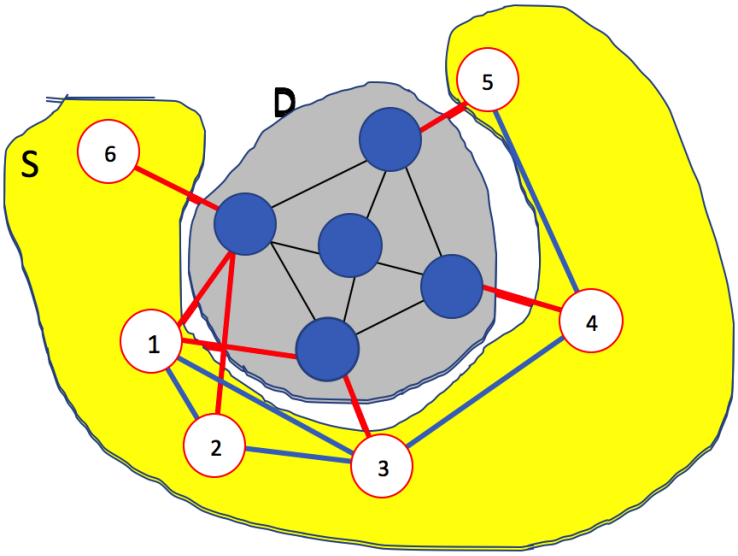
where ℓ_n in the number of links between n and D.

This means that n needs x additional connections, besides ℓ_n to respect the rule of an $\alpha-$quasi-clique in the resulting community.

Given a node $n \in S$, let us denote $S(n)$ the set of its neighbors in S, that is, its common neighbors with the community D. According to the value of x we distinguish the following situations:

- **Situation 2** (lines 16-17 of algorithm 2): $g \leq 0$ and $r < x$, so any node n among the candidates $\mathfrak{C}_r$ has fewer neighbors than required to get a positive gain. All the nodes in $\mathfrak{Cand}$ and all nodes in $\mathfrak{C}_r$ will be in the same situation as they have the same rank r. Therefore, no candidate in $\mathfrak{Cand}$ is a *good* candidate, $\mathfrak{Cand}$ must be set to the empty set and the number of remaining candidates RC decreased by $|\mathfrak{C}_r|$.

- **Situation 3** (lines 19-27 of algorithm 2): $g \leq 0$ and $r = x$, so any node n among the candidates $\mathfrak{C}_r$ has exactly as many neighbors in S as required. The function chooses one node n^* from $\mathfrak{Cand}$ and tests if all nodes in the set $\{n^* \cup S(n^*)\}$ are *good* candidates. If that is the case, the function returns $\Pi^* = \{n^* \cup S(n^*)\}$. Otherwise n^* is dropped from $\mathfrak{Cand}$ and the number of candidates RC decreases by one. This process is repeated until either a *good* set of nodes has been found or there are no more candidates in $\mathfrak{Cand}$.



Rank by Neighbors			Rank by Gain	
Rank	# Neighs	Nodes	Rank	Nodes
1	3	3	-	-
2,3,4 (ties)	2	1, 2, 4	1	1
			2	2, 4
5	1	5	-	-
6	0	6	-	-

Fig. 6 Scheme of the *RNN* algorithm: the calculation of *rank_by_neighbors()* and *rank_by_gain()* functions for $\alpha = 0.5$. First, nodes are ranked according to their number of neighbors in S, represented by the green edges. Second, if ties, nodes are ranked according to their gain, that is, the number of connections with D, represented by the red edges. For the graph in the figure, the function *rank_by_neighbors()* will rank node 3 first as it has 3 neighbors in S, next 1, 2 and 4 (two connections), then 5 (1 connection) and finally 6 (0 connections). Ties between nodes 1, 2 and 4 will be broken by the *rank_by_gain()* function given in equation (6). For the example, node 1 has the greatest gain followed by nodes 2 and 4. Therefore, the algorithm will consider the nodes in the following order: 3, 1, 2 and 4 (or 4 and 2 ties will be broken randomly), 5 and finally 6.

– **Situation 4** (lines 30-38 of algorithm 2): $g \leq 0$ and $r > x$, the function chooses one node n^* randomly from $\mathfrak{Cand}$ 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 they are chosen randomly among all possible combinations.

Different combinations are tested until either one set $\Pi = \{n^* \cup S(n^*)(x)\}$ composed of *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 or there are no more candidates in $\mathfrak{Cand}$.

Figure 7 shows an example for the 4 situations. Consider $D = \{0, 1, 2\}$ and $\alpha = 0.5$:

– Situation 1: $S = \{3, 4, 5, 6\}$, the four nodes have rank $r = 0$ (as they have 0 neighbors). Node 6 has the highest and the only with positive gain $g = 1.5$, so $\mathfrak{Cand} = \{6\}$. The algorithm will select $n^* = 6$. Since it is a *good* candidate the function will return $\Pi^* = \{6\}$.
– Situation 2: $S = \{3, 4, 5, 6\}$, $RC = 4$, $Rank = \{1 : 3, 4;\ 0 : 5, 6\}$, $r = 1$, $\mathfrak{C}_r = \{3, 4\}$ and $\mathfrak{Cand} = \{3, 4\}$ (as $\{3, 4\}$ have the highest rank). However, both of them have a negative gain $g = -0.5$. To get a positive gain they need $x = 2$ neighbors. Since $r < x$ then $\{3, 4\}$ will be discarded from the set of candidates, $\mathfrak{Cand}$ will be set to the the empty set and RC will decrease by 2. For the next iteration the algorithm will skip to the seconde highest rank and gain, so $r = 0$, $\mathfrak{C}_r = \{5, 6\}$, $\mathfrak{Cand} = \{6\}$, $g = 0.5 > 0$. In this second iteration situation 1 takes place. Since 6 is a *good* candidate the function will return $\Pi^* = \{6\}$.
– Situation 3: $S = \{3, 4, 5, 6\}$, $Rank = \{2 : 3, 4, 5;\ 0 : 6\}$, $r = 2$, $\mathfrak{C}_r = \{2, 3, 4\}$. The possible candidates are $\mathfrak{Cand} = \{2, 3, 4\}$, but their gains are negative $g = -0.5$. They need $x = 2$ neighbors to get a positive gain. They have exactly as many neighbors as required. The function chooses one node n^* of the three randomly to test if it can be added to the community with its neighbors. Since, they are *good* candidates, the function will return $\Pi^* = \{3, 4, 5\}$.

– Situation 4: $S = \{3,4,5,6,7\}$, $Rank = \{3:3;\ 2:4,5\ 1:7;\ 0:6\}$, $r = 3$, $\mathfrak{C}_r = \{3\}$ and $\mathfrak{Cand} = \{3\}$. Node 3 has a negative gain $g = -0.5$. It needs $x = 2$ neighbors to get a positive gain and it has 3 ($r > x$). The algorithm will choose 2 out of the 3 according to rule 1 (based on the rank and on the gain if necessary). The winner will be nodes 4 and 5. Then, the function will return $\Pi^* = \{3,4,5\}$ because they are *good* nodes.

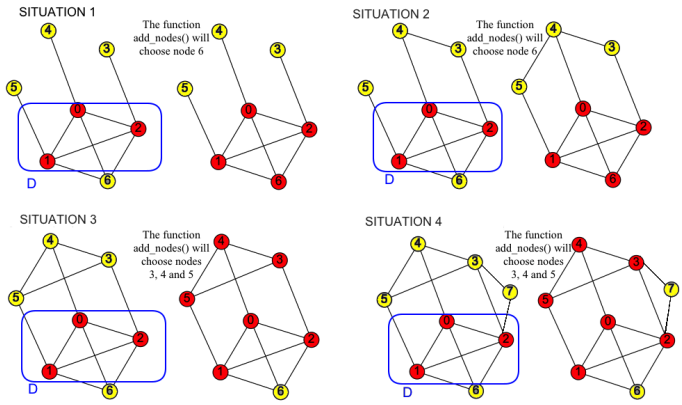


Fig. 7 The 4 situations of the function *add_nodes()* of the *RANK-NUM-NEIGHS* algorithm.

Finally, if the set of candidates is empty and no solution has been found yet (lines 41-43 of algorithm 2), then the set of candidates $\mathfrak{Cand}$ should be updated. If $\mathfrak{Cand}$ is empty (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 second highest rank and so on. Then, $\mathfrak{Cand}$ is updated as the set of nodes in S with rank r and gain g.
After the end of the while loop, in line 45, if no *good* nodes have been found and there are no more candidates ($RC = 0$), then the function returns the empty set.

In order to enhance the results, we added a **post-processing** step to our approach. When looking for the maximal $\alpha-$quasi-clique community of a node n_0 in a graph, the local communities of all its neighbors are estimated as well, by the *RNN* algorithm. Among all the communities of its neighbors that contain n_0, if the biggest one is larger than that of n_0, then the local community of n_0 is set to this larger community.

Algorithm 2 The *add_nodes()* function for the *RANK-NUM-NEIGHS* algorithm.

Require: set D, set S, parameter α
Ensure: A set of *good* nodes Π^* from S to enter D.

1: Calculate
   – $RC \leftarrow |S|$, $Rank \leftarrow rank\_by\_neighbors(S, G(S))$,
   – r (highest rank in $Rank$), $\mathfrak{C}_r$.
   – $Gain \leftarrow rank\_by\_gain(D, \mathfrak{C}_r, G(D \cup \mathfrak{C}_r), \alpha)$
   – g (highest gain in $Gain$).
   – $\mathfrak{Cand} \leftarrow$ set of nodes in $\mathfrak{C}_r$ whose gain is g.
   – Set $\Pi^* = \{\varnothing\}$.
2: **while** $(RC > 0)$ **do**
3: **if** $(g > 0)$ **then**
4: // SITUATION 1
5: **while** $(\mathfrak{Cand} \neq \{\varnothing\})$ **do**
6: choose a node n^* randomly from $\mathfrak{Cand}$.
7: **if** n^* is a *good* candidate **then**
8: **return** $\Pi^* = \{n^*\}$.
9: **else**
10: Drop n^* from $\mathfrak{Cand}$ and $RC = RC - 1$.
11: **end if**
12: **end while**
13: **else**
14: Calculate: $x \leftarrow \left\lfloor \left(\frac{\alpha|D|-\ell}{(1-\alpha)} \right) \right\rfloor + 1$, where $\ell = (g + \alpha|D|)$.
15: **if** $(x > r)$ **then**
16: // SITUATION 2
17: Set $RC = RC - |\mathfrak{C}_r|$, $\mathfrak{Cand} = \{\varnothing\}$
18: **else if** $(x = r)$ **then**
19: // SITUATION 3
20: **while** $(\mathfrak{Cand} \neq \{\varnothing\})$ **do**
21: choose a node n^* randomly from $\mathfrak{Cand}$.
22: **if** $\Pi = \{n^* \cup S(n^*)\}$ are *good* candidates **then**
23: **return** $\Pi^* = \Pi$.
24: **else**
25: Drop n^* from $\mathfrak{Cand}$ and $RC = RC - 1$.
26: **end if**
27: **end while**
28: **else**
29: // SITUATION 4
30: **while** $(\mathfrak{Cand} \neq \{\varnothing\})$ **do**
31: choose a node n^* randomly from $\mathfrak{Cand}$.
32: Set $S(n^*)(x) \leftarrow$ Choose up to x nodes from $S(n^*)$ according to rule 1.
33: **if** $\Pi = \{n^* \cup S(n^*)(x)\}$ are *good* candidates **then**
34: **return** $\Pi^* = \Pi$.
35: **else**
36: Drop n^* from $\mathfrak{Cand}$ and $RC = RC - 1$.
37: **end if**
38: **end while**
39: **end if**
40: **end if**
41: **if** $(|\mathfrak{Cand}|=0)$ **then**
42: Update r, g and the set of candidates $\mathfrak{Cand}$.
43: **end if**
44: **end while**
45: **return** Π^*

5.3.3 Complexity of the proposed method

Consider the *general greedy scheme for local community detection* described in algorithm 1. The while loop is the most time consuming operation. This loop executes as long as it is possible to add nodes to the local community. Each iteration of the loop depends strongly on the selection criterion F. For the *RNN* method, this is the *add_nodes* function (see Algorithm 2).

The *add_nodes* function calls two functions (see algorithm 2, line 1): the *rank_by_neighbors* function and the *rank_by_gain* function.

At the first iteration, the *rank_by_neighbors* function elaborates a ranking of the neighbors of the starting node n_0 based on the number of common neighbors with n_0. This operation takes as much time as sorting d_0 elements, where d_0 is the degree of n_0. Depending on the sorting algorithm, this might take either $O(d_0 \log(d_0))$ or $O(d_0^2)$ time in the worst case. The output of this function is contained in the variable *Rank*.

Next, if r is the highest rank and $\mathfrak{C}_r$ is the set of nodes whose rank is r. The function *rank_by_gain* makes a ranking of nodes in $\mathfrak{C}_r$ based on the gain. In the best case there is only one node in $\mathfrak{C}_r$ and in the worst case all the nodes have the same rank and *rank_by_gain* makes a second ranking of d_0 elements again. The set of nodes with the highest gain g is denoted $\mathfrak{Cand}$.

The operations in the while loop of the *add_nodes* function (lines 2 until the end) are less time consuming than the two ranking functions as they consider only a small subset of S, that is $\mathfrak{Cand}$. Furthermore, according to the values of r, g and x just one of the four operations takes place.

At the end of the first iteration, one or more nodes have been added to the local community. D and S must be updated in algorithm 1. At the second iteration, the *add_nodes* function is called with new input parameters. The ranking *Rank* is not calculated again for all the nodes in S but it is just updated with the new neighbors of the new local community D.

According to theorem 1, the optimal solution contains at most as many nodes as those contained in the first and in the second neighborhood of node n_0. This means, that the *rank_by_neighbors* function will make a sorting of at most $d_0 + |\Gamma(\Gamma(n_0))| = |\Gamma_{1,2}(n_0)|$. As this is the most time consuming operation, the complexity of the algorithm is determined by this operation and its worst time complexity is the time required to sort $|\Gamma_{1,2}(n_0)|$ elements. Depending on the data structure, this operation can perform in $O(|\Gamma_{1,2}(n_0)| \log |\Gamma_{1,2}(n_0)|)$ or $O(|\Gamma_{1,2}(n_0)|^2)$ time in the worst case. This will be the worst time complexity of the *while* loop in algorithm 2.

Now, we have to consider the *post-processing* step included described by the end of the last section. This step imples to perform the sorting operations d_0 times. This will lead to a complexity of $O(d_0|\Gamma_{1,2}(n_0)| \log |\Gamma_{1,2}(n_0)|)$ or $O(d_0|\Gamma_{1,2}(n_0)|^2)$ in the worst case.

Notice that this calculation is mostly pesimistic because real networks are mostly free-scale. Therefore, most nodes have a low degree and the algorithm will stop earlier. It is unlikely to get a local community of size $\Gamma_{1,2}(n_0)$ that is, a community which contains the entire first and second neighborhood.

6 Evaluations

The *RANK-NUM-NEIGHS* method is evaluated according to 3 criteria: the quality of the detected communities, the execution time and the stability of the algorithm. Quality refers to the community sizes. Indeed, the *maximal $\alpha-$quasi-clique community of a given node problem* aims to detect $\alpha-$quasi-cliques of maximal size. Therefore, the bigger the output communities are, the higher the quality of the method is.

We compared our method to the *QUICK* and the *RLS-DLS* methods described in Section 4. The executable versions of both methods were kindly provided by their authors. At the end of this Section, we present a practical application on "The Zachary Karate Club network" (*karate*) [50].

The evaluations are made on real and artificial networks. Concerning the artificial networks, we generated benchmark LFR graphs (see [31]) of sizes ranging from 1000 to 10000 nodes by increments of 1000. The input

parameters are the same as those considered in [19][6]. The real networks are the same used in [19]:

- "The Zachary Karate Club network" (*karate*) [50], 34 nodes and 78 edges.
- "The College football network" (*football*) [26], 114 nodes and 613 edges.
- "Books about US politics" (*polbooks*) [30], 104 nodes and 441 edges.
- "Political blogosphere" (*polblogs*) [2], 1490 nodes and 16715 edges.

6.1 Sizes of the detected communities

The boxplots of the community sizes for real and artificial networks of 1000 nodes[7] are shown in Figure 8. For all the datasets we ran the *RNN* and the *RLS-DLS* heuristics 10 times. For each dataset, the figure compares the community sizes obtained by our algorithm, *RNN*, to those obtained by the *QUICK* (QCK) and by the *RLS-DLS* (RLS) methods. The horizontal axis presents each method for different values of α, ranging from 0.6 to 0.9.

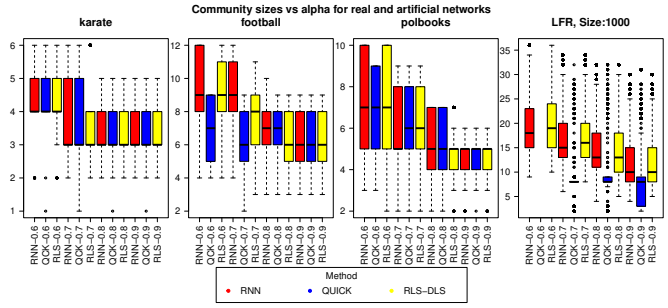


Fig. 8 Sizes for artificial and real networks, comparison with the global *QUICK* and *RLS-DLS* methods

Concerning the real networks, clearly the results of our method are at least as good as those of the other two methods. Although, the *QUICK* method seems to give smaller communities for $\alpha = 0.6$ or $\alpha = 0.7$ than the other two methods, for values of α closer to 1 the distributions of sizes look quite similar for the three methods.

Concerning the LFR networks in Figure 8, clearly our method and the *RLS-DLS* method give better results than *QUICK*. Indeed, *QUICK* does not guarantee that each node will be assigned to the largest clique it belongs to as mentioned in Section 4. Besides that, it is very slow in comparison to the other two methods (as we will mention in Section 6.3). For instance, in Figure 8, the results for the LFR networks for $\alpha = 0.6$ are missing because the algorithm was still running afer 24 hours of execution. Furthermore, we could not obtain the results for the *polblogs* network (1490 nodes and 16715 edges) for a high value of $\alpha = 0.8$ as the output file consumed more than 782Gb in stockage and was still running. Certainly, the output of this method is a list of all the existing α-quasi-cliques of a network for a given α, which is so space-consuming.

6.2 Stability of the algorithm

We compared the stability of the *RANK-NUM-NEIGHS* algorithm with that of the *RLS-DLS* algorithm on artificial networks of sizes ranging from 1000 to 10000 (The *QUICK* method is deterministic). We present in this paper the results for networks of 1000 nodes only (the results for the other sizes are quite similar). We executed both algorithms for values of the parameter α ranging from 0.5 to 0.9. Concerning the *RLS-DLS* method, we executed this heuristic 10 times. Concerning the *RNN* heuristic, for every node n of each graph we made 10 iterations of the algorithm. So, for both methods, we obtained 10 local communities for each node. Let us denote $C(n)_i$ the local community of the node n obtained at the i^{th} iteration. Then, the Jaccard index for these 10 sets, denoted $J(n)$, is given by the following formula:

$$J(n) = \frac{|\bigcap_{i=1}^{10} C(n)_i|}{|\bigcup_{i=1}^{10} C(n)_i|} \quad (8)$$

Remark that $J(n)$ ranges from 0 to 1 and it is equal to 1 if the algorithm is 100% stable. $J(n) = 1$ would signify that for 10 executions the algorithm returned identical results. The Figure 9 shows the frequency at which we obtained critical values of the Jaccard index for both methods.

Concerning our method, we remark from Figure 9 that more than 80% of times we obtained a Jaccard index equal to 1. This means that on average for more of 80% of the nodes the algorithm detected the same resulting community after 10 iterations. The remaining

[6] The average degree is 20, the maximum degree 50, the exponent of the degree distribution is -2 and that of the community size distribution is -1. We chose three values of mixing parameter λ, 0.10, 0.20 and 0.30. The results presented in this paper are those for $\lambda = 0.10$ to evaluate size, density and stability. The results have nearly the same behavior for the other values of mixing parameter.

[7] The results obtained for other network sizes have nearly the same behavour.

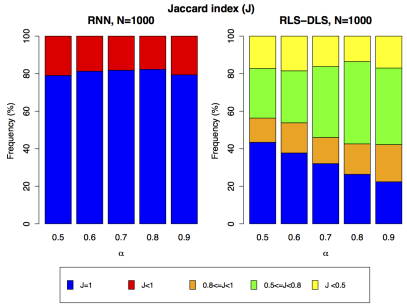


Fig. 9 Stability for artificial LFR graphs.

20% of times presented a lot of variability. In contrast, for the *RLS-DLS* method, the value 1 is obtained in less than 40% of times (except for $\alpha = 0.5$) and this percentage decreases with increasing α. However, about 20% of times the Jaccard index is smaller than 0.5. This emplies that about half of the times the Jaccard index is bigger or equal to 0.5 but smaller than 1. This lack of stability can be explained by the fact that the *RLS-DLS* method includes a *remove* step at each iteration. Certainly, even if a node has already joined the community at a given iteration later it can be removed. Whereas in the *RNN* method, once a node joined the community it will stay until the end.

However, we have to mention that although the *RLS-DLS* method might provide different results at each iteration, in most of the cases they are of the same good quality. Indeed, we mentioned in Section 3 that the *maximal $\alpha-$quasi-clique community of a given node problem* does not have a unique solution. This means that the same node can belong to different $\alpha-$quasi-clique communities of the same maximal size. Then *RLS-DLS* will give a choice. This can be either a disadvantage or advantage.

6.3 Execution time

We executed the *RANK-NUM-NEIGHS* algorithm on artificial networks of sizes ranging from 1000 to 10000 (see [19] for more details). The Figure 10 shows the execution time in seconds for each entire network.

The Figure 10 shows that the execution time of the algorithm is quite stable. The execution time seems to increase smoothly with the network size. One reason that explains the rapidity of the *RNN* method is that at each iteration it is possible to add more than one node, so the nodes become saturated faster than just adding one node per iteration. We ran the *RNN* algorithm on bigger graphs. The results are shown in Table 1.

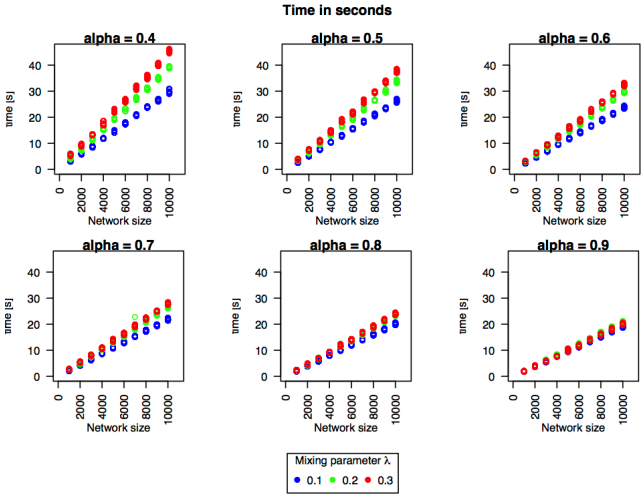


Fig. 10 *RNN* method: execution time to detect the local community of all nodes of every artificial LFR graph.

Table 1 Time for big graphs in seconds

Size	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
100000	269	223	192
500000	1345	1130	978

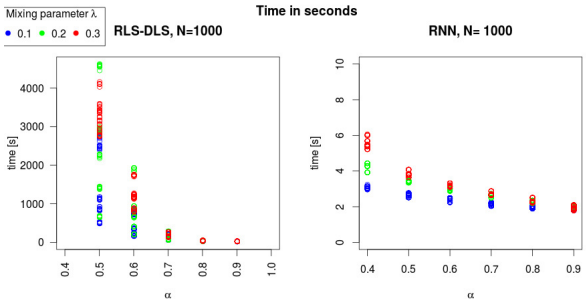


Fig. 11 Comparison of the execution time *RNN* method versus *RLS-DLS* method.

We do not present the comparison between our method and the *QUICK* method as this last one took more than one day for a graph of 9000 nodes. We also remarked that it was very slow for the *polblogs* network of 1490 nodes and 16715 edges. We compared the execution time of our method to that of the *RLS-DLS* method for a LFR network of size 1000. The results are presented in Figure 11

By looking the Figure 11 we remark that the *RLS-DLS* method is much slower than the *RNN* method and this difference increases with decreasing α. For $\alpha = 0.8$ the execution time is about 30 seconds and for $\alpha = 0.9$ about 20 seconds for the *RLS-DLS* method.

6.4 Practical application on a real social network

Analysis of quasi-cliques have been proven to be useful in many problems (see for example [37], [1], [44]). Among examples in big data applications we can mention:

- Consider a telecommunication network, where each node represents a person and there is an edge between two nodes if and only if they exchanged a phone call. An application in anomaly detection is finding sets of vertices that are *almost* cliques. Large sets of such vertices are contradicting to the average human habits: *who talks to say 50 people who all talk among each other as well?*. According to [3]: "Detecting those nodes whose neighbors are very well connected (near-cliques) or not connected (stars) turn out to be *strange*: in most social networks, friends of friends are often friends, but either extreme (clique or star) is suspicious".
- In bioinformatics, many problems have been modeled using cliques. For instance, the problem of clustering gene expression data is usually modeled as graph decomposition problem into disjoint cliques (see [9] and [46]).
- Another application in biology is the clustering of protein-protein interaction (PPI) networks to detect functional groups. Researchers in biology attempt to understand cellular organization and function by analyzing these PPI networks. As discussed in [52] recently, experimental techniques have generated a large amount of protein-protein interaction (PPI) data.
- In social network analysis, as pointed out by [49], on real data the communities are far away from being highly dense. Therefore, the detection of *quasi-clique* can be much more appropiated than detection complete cliques.

Furthermore, the mining of subgraphs of high density lies at the core of large scale data mining (the interested reader can see [6]).

In this section we present a practical example of our method in a well-known social network of friendships, the *Zachary karate club* studied by [50]. A social network between 34 members of a karate club at a US university in the 1970s shown in Figure 12. The club hired a karate instructor, Mr. Hi (node 1). There was a conflict between Mr. Hi and the club president, John A. (node 34) over the price of the lessons. The entire club became divided over this issue during the decision meetings before spliting completely into two parts. Mr. Hi was fired and created another club. About half of the active members joined him whereas the other half supported John. Not all individuals in the network were solidly members of one faction or the other. Some vacillated between the two ideological positions, and others were simply satisfied not to take sides. These individuals are key nodes in the network, in that it was through them that information was likely to pass from one faction to the other.
An interesting application of our method would be to detect the local community of Mr. Hi (node 1) and the community of John A (node 34) in order to detect the individuals strongly connected to those key nodes.
The Figure 12 shows the local communities of nodes 1 and 34 after applying the *RNN* method for $\alpha = 0.8$ and $\alpha = 0.9$.

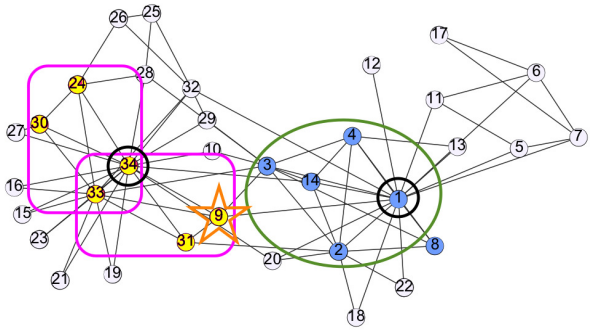


Fig. 12 Local communities of nodes 1 and 34 of the karate club network.

Concerning node 1, its community contains 6 members. According to [50] all those members strongly supported the instructor, except node 14 whose affiliation was weak. Concerning node 34 we found 2 optimal solutions of size 4, both of them are complete cliques as α is high. All these members were strongly connected to the administrator, except node 9 (present in one of the optimal solutions). indeed node 9 is also connected to Mr. Hi (node 1) and was a weak supporter of John (node 34).

7 Bound on the optimal solution

In order to evaluate the quality we calculated an upper bound on the optimal solution of the *maximal $\alpha-$quasi-clique community of a given node problem* stated in Section 3.

Consider a node n_0 with degree $d(n_0)$, then the size of its maximal $\alpha-$quasi-clique community can not exceed the following quantity, denoted $B_0(n)$ (the proof

is similar to that of theorem 2, taking $d(n_0)$ instead of d^{in}):

$$B_0(n_0) = \left\lceil \left(\frac{d(n_0)}{\alpha} \right) \right\rceil. \quad (9)$$

$B_0(n_0)$ constitutes an upper bound for the optimal solution of our problem. This bound is satisfied if and only if all the neighbors of n_0 belong to its local community. That means when n_0 is saturated. The bound $B_0(.)$ can be reduced if we consider that the optimal solution contains at least two nodes (for any α), that is n_0 and at least one of its neighbors. Then, another bound for the optimal solution will be:

$$B_1(n_0) = \min \left(\max_{n \in \Gamma(n_0)} B_0(n), B_0(n_0) \right). \quad (10)$$

Therefore, the optimal solution, denoted $C^*(n_0)$, is bounded by:

$$|C^*(n_0)| \leq B_1(n_0) \leq B_0(n_0).$$

Now, let us consider the node 0 in Figure 13 and $\alpha = 0.5$, we have $B_0(0) = 8$ and $B_1(0) = 6$, therefore, $|C^*(0)| \leq 6$. However, this bound is satisfied if and only if exactly three of the neighbors of node 0 are in $C^*(0)$. We remark that nodes 3 and 4 can not belong to a community of size 6. Since their degrees are worth 1 then $B_0(3) = B_0(4) = 2$. We can deduce that this bound can not be reached. Therefore, there are only two possibilities: either nodes 3 and 4 do not belong to for the optimal solution or at least one of them belongs to it. In the first case, the community size of the optimal solution is at most 4 and in the second case the community size of the optimal solution is at most 2. In conclusion, the optimal solution is bounded by 4, we denote this new bound is B_2 and we have:

$$|C^*(n_0)| \leq B_2(n_0) \leq B_1(n_0) \leq B_0(n_0).$$

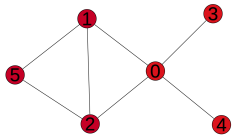


Fig. 13 Example for the calculation of the upper bound.

By looking the Figure 13 we can easily deduce that the optimal solution is of size 4: $C(0)^* = \{0, 1, 2, 5\}$.

We can repeat this operation iteratively to obtain a smaller upper bound. Bound B_2 is useful especially for nodes that have a high degree. For example, consider the node that represents the instructor in the Zachary Karate Club network [50] (node 1). We obtain $B_0(1) = 32$, $B_1(1) = 20$ and $B_2(1) = 10$. The solution we found with the *RANK-NUM-NEIGHS* is of size 8.

We calculated the upper bound B_2 for all the nodes of the 4 real networks. The Figure 14 shows the histogram of the *difference* between the upper bound B_2 and the results found by the *RNN* algorithm.

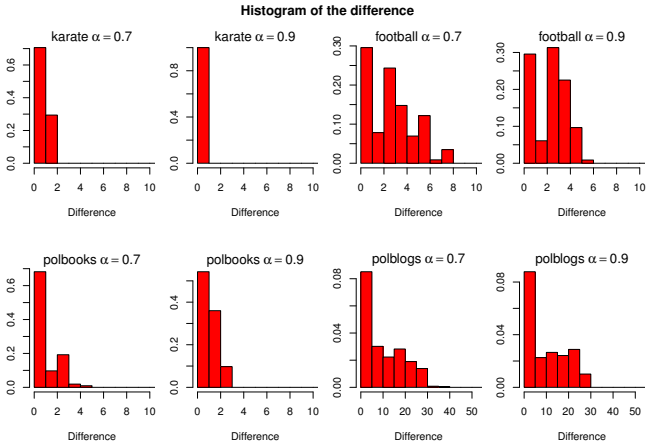


Fig. 14 Histogram of the difference between the upper bound B_2 and the results of our algorithm for real networks.

The Figure 14 shows that the difference is concentrated in small values for 3 networks: karate, polbooks and polblogs because the frequency is decreasing. Especially for the *karate* dataset, we reached the optimal solution for all but two nodes. A value of the difference equal to zero means that our algorithm reached the optimal solution. That is the case for most nodes of the three networks. Concerning the network football the difference is rarely null. Indeed, this network contains mainly complete cliques, so the nodes in the communities are not saturated. B_2 constitutes an upper bound, and it is satisfied if at least one node in the community is saturated.

8 Conclusions and perspectives

In this paper we tackled what we called the *maximal $\alpha-$quasi-clique community of a given node problem*. In order to approach the solution of this problem, we proposed a community detection method that has two main characteristics. First, the resulting communities are α-quasi-cliques, very dense communities. Second, it

is a *local* community detection method. Indeed, instead of partitioning the entire network our method detects the community of a given node of interest. We showed that this approach can be useful when we are interested in some particular nodes in the network or when we do not have acces to the whole network data. We showed that our method, called *RANK-NUM-NEIGHS* (RNN), is scalable. To the best of our knowledge we are the first to address the maximal α-quasi-clique problem from a local-community point of view.

Some perspectives to this work can be taking into account the attributes of nodes (for example, following ideas from [22]). This problem can also be the building block of more complex applications where the internal density of community plays an important role. For example, in friend recommendation, the missing links of an α-quasi-clique community can be recommended. In churn prediction, the communities can better model the notion of the closest friends as studied in [40].

APPENDIX

Proof that an $\alpha-$quasi-clique in the sens of definition 1 has a density greater than α

Proof. Let C be an $\alpha-$quasi-clique in the sens of definition 1. Then, every node $n \in C$ is connected to more than $\alpha(|C|-1)$ nodes. Therefore, $e[C]$, the number of edges induced by C verifies:

$$2e[C] > |C|\alpha(|C|-1).$$

Then, the edge density $\delta(C)$ of C verifies $\delta(C) = \frac{e[C]}{\frac{|C|(|C|-1)}{2}} > \alpha$ □

Proof of theorem 1

Proof. Given an $\alpha-$quasi-clique C, for any two distinct nodes $u, v \in C$, either u and v are directly connected or not. Let us suppose they are not connected. For $\alpha \geq 0.5$, according to the definition 1, $|\Gamma(u)| > \frac{|C|-1}{2}$ and $|\Gamma(v)| > \frac{|C|-1}{2}$. Since $|\Gamma(u) \cup \Gamma(v)| \leq |C|$, then $\Gamma(u) \cap \Gamma(v) \neq \emptyset$ and u and v have at least one common neighbor. Therefore, the distance between any two distinct nodes is at most 2. □

Proof of lemma 1

Proof. Let us suppose that at iteration i the node n is added to D. Let us denote m a neighbor of n, such that $m \in S$, at iteration i the gain of m is:

$$g_m^i = l_m - \alpha(|D|-1)$$

The gain of m at iteration $i+1$, that is once $n \in D$ will be:

$$g_m^{I+1} = l_m + 1 - \alpha|D|$$

So, the gain of m increases by:

$$g_m^{I+1} - g_m^i = (1-\alpha)$$

Therefore the gain of all the neighbors of n will increase by $(1-\alpha)$ □

Proof of theorem 2

Proof. For any α-consensus community D, any node $n \in D$ with internal degree $d^{in}(n)$ respects the majority rule:

$$d^{in}(n) > \alpha(|D|-1) \iff \frac{d^{in}(n)}{\alpha} + 1 > |D|$$

we are interested in the maximum possible integer $D^{max}(n)$ that respects this condition. There are two possibilities:

$$D^{max}(n) = \begin{cases} \frac{d^{in}(n)}{\alpha} & \text{If } \left(\frac{d^{in}(n)}{\alpha} + 1\right) \text{ is integer.} \\ \left\lfloor \left(\frac{d^{in}(n)}{\alpha}\right) \right\rfloor + 1 & \text{otherwise.} \end{cases} \tag{11}$$

Expression (11) is equivalent to:

$$D^{max}(n) = \left\lceil \left(\frac{d^{in}(n)}{\alpha}\right) \right\rceil$$

Analogously, given an α-consensus community D the minimal internal degree a node in D can have, denoted d^{min} verifies:

$$d^{min} > \alpha(|D|-1)$$

since d^{min} is the minimum possible integer that respects this condition. Two situations can take place:

$$d^{min} = \begin{cases} \alpha(|D|-1)+1 & \text{If } \alpha(|D|-1) \text{ is integer.} \\ \lceil \alpha(|D|-1) \rceil & \text{otherwise.} \end{cases} \tag{12}$$

which is the same as

$$d^{min} = \lfloor \alpha(|D|-1) \rfloor + 1$$

□

Proof of theorem 3

Proof. Given a node n connected by ℓ_n links to an α-consensus community D, if its gain is negative, it will need x more connections to D such as:

$$\ell_n + x > \alpha(|D| + x) \Leftrightarrow x > \frac{\alpha|D| - \ell_n}{1-\alpha}$$

since x is the minimum possible integer that respects this condition. Two situations can take place:

$$x = \begin{cases} \left(\frac{\alpha|D|-\ell_n}{1-\alpha} + 1\right) & \text{If } \left(\frac{\alpha|D|-\ell_n}{1-\alpha}\right) \text{ is integer.} \\ \left\lceil \frac{\alpha|D|-\ell_n}{1-\alpha} \right\rceil & \text{otherwise.} \end{cases} \quad (13)$$

which is the equivalent to:

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

□

Acknowledgment

This work is supported by REQUEST project.

References

1. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: Proceedings of the 5th Latin American Symposium on Theoretical Informatics, LATIN '02, pp. 598–612. Springer-Verlag, London, UK, UK (2002)
2. Adamic, L.A., Glance, N.: The political blogosphere and the 2004 u.s. election. In: Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem, pp. 36–43. ACM, NY, USA (2005)
3. Akoglu, L., Mcglohon, M., Faloutsos, C.: Anomaly detection in large graphs. In: In CMU-CS-09-173 Technical Report (2009)
4. Asahiro, Y., Hassin, R., Iwama, K.: Complexity of finding dense subgraphs. Discrete Appl. Math. **121**(1-3), 15–26 (2002). DOI 10.1016/S0166-218X(01)00243-8. URL http://dx.doi.org/10.1016/S0166-218X(01)00243-8
5. Bagrow, J.P.: Evaluating local community methods in networks. In: Journal of Statistical Mechanics, p. 05001 (2008)
6. Bahmani, B., Kumar, R., Vassilvitskii, S.: Densest subgraph in streaming and mapreduce. CoRR **abs/1201.6567** (2012). URL http://arxiv.org/abs/1201.6567
7. Battiti, R., Mascia, F.: Reactive local search for maximum clique: A new implementation. Tech. Rep. DIT-07-018, Informatica e Telecomunicazioni, University of Trento, Trento, Italy (2007)
8. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem **29**(4), 610 (2001)
9. Ben-Dor, A., Shamir, R., Yakhini, Z.: Clustering gene expression patterns
10. Blondel, V.D., Guillaume, J., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. In: Journal of Statistical Mechanics: Theory and Experiment (2008)
11. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Handbook of Combinatorial Optimization, pp. 1–74. Kluwer Academic Publishers (1999)
12. Brunato, M., Hoos, H.H., Battiti, R.: On effectively finding maximal quasi-cliques in graphs. In: V. Maniezzo, R. Battiti, J.P. Watson (eds.) LION, *Lecture Notes in Computer Science*, vol. 5313, pp. 41–55. Springer (2007)
13. Campigotto, R., Conde-Céspedes, P., Guillaume, J.: A generalized and adaptive method for community detection. CoRR **abs/1406.2518** (2014). URL http://arxiv.org/abs/1406.2518
14. Chen, J., Saad, Y.: Dense subgraph extraction with application to community detection. IEEE Transactions on Knowledge and Data Engineering **24**(7), 1216–1230 (2012)
15. Chen, J., Zaiane, O.R., Goebel, R.: Local communities identification in social networks. In: ASONAM, pp. 237–242 (2009)
16. Clauset, A.: Finding local community structure in networks. In: Physical Review, vol. 72, p. 026132 (2005)
17. Conde-Céspedes, P., Marcotorchino, J., Viennet, E.: Comparison of linear modularization criteria using the relational formalism, an approach to easily identify resolution limit. Revue des Nouvelles Technologies de l'Information **Extraction et Gestion des Connaissances, RNTI-E-28**, 203–214 (2015)
18. Conde-Céspedes, P., Marcotorchino, J., Viennet, E.: Comparison of linear modularization criteria using the relational formalism, an approach to easily identify resolution limit. Advances in Knowledge Discovery and Management (AKDM-6) pp. 101–120 (2017)
19. Conde-Céspedes, P., Ngonmang, B., Viennet, E.: Approximation of the maximal α-consensus local community detection problem in complex networks. In: IEEE SITIS 2015, Complex Networks and their Applications. Bangkok, Thailand (2015)
20. Condorcet, C.A.M.d.: Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. Journal of Mathematical Sociology **1**(1), 113–120 (1785)
21. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, pp. 991–1002. ACM, New York, NY, USA (2014)
22. Dang, T.A., Viennet, E.: Community detection based on structural and attribute similarities. In: International Conference on Digital Society (ICDS), pp. 7–14 (2012)
23. Dang, T.A., Viennet, E.: Collaborative filtering in social networks: A community-based approach. In: IEEE ComManTel 2013, Int. Conf. on Computing, Management and Telecommunications (2013)
24. Fortunato, S.: Community detection in graphs. In: Physics Reports, vol. 486, pp. 75–174 (2010)
25. Fortunato, S., Barthelemy, M.: Resolution limit in community detection. In: Proceedings of the National Academy of Sciences of the United States of America (2006)
26. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences of the United States of America **99**(12), 7821–7826 (2002)

27. Harary, F., Ross, I.C.: A procedure for clique detection using the group matrix. Sociometry **20**, 205–215 (1957)
28. Karp, R.M.: Reducibility among combinatorial problems. In: R.E. Miller, J.W. Thatcher (eds.) Complexity of Computer Computations, The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
29. Komusiewicz, C.: Multivariate algorithmics for finding cohesive subnetworks. Algorithms **9**(1) (2016)
30. Krebs, V.: Books about US politics (2004). URL http://www.orgnet.com/
31. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**(4) (2008)
32. Lee, V.E., Ruan, N., Jin, R., Aggarwal, C.C.: A survey of algorithms for dense subgraph discovery. In: C.C. Aggarwal, H. Wang (eds.) Managing and Mining Graph Data, *Advances in Database Systems*, vol. 40, pp. 303–336. Springer (2010)
33. Liang, R., Hua, J., Wang, X.: Vcd: A network visualization tool based on community detection. In: Control, Automation and Systems (ICCAS), 2012 12th International Conference on, pp. 1221–1226 (2012)
34. Liu, G., Wong, L.: Effective pruning techniques for mining quasi-cliques. In: W. Daelemans, B. Goethals, K. Morik (eds.) Machine Learning and Knowledge Discovery in Databases, *Lecture Notes in Computer Science*, vol. 5212, pp. 33–49. Springer Berlin Heidelberg (2008)
35. Luo, F., Wang, J.Z., Promislow, E.: Exploring local community structure in large networks. In: WI'06., pp. 233–239 (2006)
36. Marcotorchino, F., Michaud, P.: Optimisation en Analyse ordinale des données. Masson, Paris (1979)
37. Matsuda, H., Ishihara, T., Hashimoto, A.: Classifying molecular sequences using a linkage graph with their pairwise similarities. Theoretical Computer Science **210**(2), 305 – 325 (1999)
38. Newman, M., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E. **69**(2) (2004)
39. Ngonmang, B., Tchuente, M., Viennet, E.: Local communities identification in social networks. Parallel Processing Letters **22**(1) (2012). DOI 10.1142/S012962641240004X
40. Ngonmang, B., Viennet, E., Tchuente, M.: 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, pp. 282–288 (2012)
41. Owsiński, J., Zadrożny, S.: Clustering for ordinal data: a linear programming formulation. Control and Cybernetics **15**(2), 183–193 (1986)
42. Pattillo, J., Veremyev, A., Butenko, S., Boginski, V.: On the maximum quasi-clique problem. Discrete Applied Mathematics **161**, 244 – 257 (2013)
43. Pattillo, J., Youssef, N., Butenko, S.: On clique relaxation models in network analysis. European Journal of Operational Research **226**(1), 9–18 (2013)
44. Pei, J., Jiang, D., Zhang, A.: On mining cross-graph quasi-cliques. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05, pp. 228–238. ACM, New York, NY, USA (2005)
45. Pullan, W.J., Hoos, H.H.: Dynamic local search for the maximum clique problem. J. Artif. Intell. Res. (JAIR) **25**, 159–185 (2006)
46. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. In: In Proceedings of ISMB 2002, pp. 136–144 (2002)
47. Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., Tsiarli, M.: Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, pp. 104–112. ACM, New York, NY, USA (2013)
48. Wu, Q., Hao, J.K.: A review on algorithms for maximum clique problems. European Journal of Operational Research **242**(3), 693–709 (2015)
49. Yang, J., Leskovec, J.: Overlapping communities explain core-periphery organization of networks. Technical report, Stanford University (2014). URL http://ilpubs.stanford.edu:8090/1103/
50. Zachary, W.W.: An information flow model for conflict and fission in small groups. Journal of Anthropological Research (1977)
51. Zahn, C.: Approximating symmetric relations by equivalence relations. SIAM Journal on Applied Mathematics **12**, 840–847 (1964)
52. Zhang, Y., Lin, H., Yang, Z., Wang, J.: Construction of dynamic probabilistic protein interaction networks for protein complex identification. BMC Bioinformatics **17**, 186 (2016). DOI 10.1186/s12859-016-1054-1. URL http://dx.doi.org/10.1186/s12859-016-1054-1