A general approach to fuzzy community detection in social networks

Alfonso Niño, Sebastián Reyes, José Ángel Martín-Baos, Camelia Muñoz-Caro
Escuela Superior de Informática, Universidad de Castilla-La Mancha
Paseo de la Universidad, 4, 13004, Ciudad Real, Spain
Email: Alfonso.Nino@uclm.es

Abstract— The present work addresses the problem of community detection in social networks. Determination of the community structure permits to identify the organizational and functional units of the network. To such an end, the most powerful, and less studied, approach is the fuzzy one. Here, each network node participates in all the communities simultaneously. In this paper, we develop a general formalism for fuzzy community detection in weighted, unweighted, directed and undirected social networks. Using an error functional measuring the difference between the predicted and observed network topologies, the problem is transformed into an unconstrained minimization. This allows to define a general algorithmic pattern based on the greedy paradigm, which can be customized to several different fuzzy community detection algorithms. Analysis of the algorithmic complexity of the procedure permits to determine the factors responsible for the variation of its running time. Using this information, we define an efficient algorithm (TRIBUNE) by applying a conjugate gradient approach. To determine the performance of TRIBUNE, we introduce a new network model specially designed as a fuzzy community detection benchmark. Using this benchmark, we show that TRIBUNE greatly reduces the dependence on the number of iterations of the optimization procedure. As a result, TRIBUNE needs in average 84% less time than the steepest descent baseline to solve a test set of benchmark networks. In addition, the performance of TRIBUNE increases with the network size. Finally, we find that the current implementation of TRIBUNE can handle networks up to 100000 nodes in a time frame of 18 hours.

Keywords— Fuzzy communities; Social networks; Benchmark network

I. INTRODUCTION

The last decade has experienced an increasing interest in the description of complex systems defined as a network of interacting elements. This complex networks approach permits to describe and analyze the structure and behavior of real networked systems such as social networks [1, 2]. In this context, it results of particular interest to analyze the mesoscale structure of networks by determining sets of similar elements. This allows identifying the inner structure and group behavior of the system.

Finding groups (clusters, communities) of alike elements in a set is not a new problem. In the pattern recognition and machine learning fields, several efficient clustering algorithms solve this problem for multivariate data, *i. e.*, for elements characterized by a vector of attributes [3]. These algorithms allow to find sets, clusters, of similar elements using some distance metric derived from the corresponding attribute vectors. However, for relational systems where the elements are interrelated in form of a network, the topology of the problem imposes a restriction not considered in clustering algorithms. Therefore, identification of similar elements in networks needs to take into account the associated connectivity. The corresponding algorithms identify communities of elements more densely connected among them than with the rest of the network. For an excellent review on the topic see [4].

With respect to community detection in complex networks there are two specific problems deserving consideration. The first is that the number of communities is usually unknown beforehand. This problem was addressed by Newman and Girvan through the concept of modularity [5]. The key idea is that the existence of a community is evidenced by comparing the edge density of the community subgraph with the density expected if the subgraph were random. Thus, different community detection approaches have been developed based on the idea of maximizing modularity such as the Newman [6] and Louvain [7] algorithms. The second problem is that frequently the nodes belong to several communities as, typically, in social networks. This situation is addressed by overlapping communities algorithms [4]. Here, two different approaches are used. First, we have crisp community detection algorithms where the nodes are associated to one or more communities. This is by far the most popular approach. Second, we have fuzzy community detection algorithms, where each node has a fractional participation in every community. This last is a very realistic approach that allows quantifying the different roles the nodes play in the network. Thus, for instance, in a social network, it is possible to identify and quantify the participation of an individual in different groups (family, friends, work, …), which can overlap to a greater or lesser extent. The fuzzy approach represents the general case of community detection since the individual membership coefficients allow the adscription of each node to a single community or to a reduced set of them.

In pattern recognition and machine learning, the canonical fuzzy clustering algorithm is fuzzy c-means introduced by Dunn [8] and Bezdek [9]. This algorithm can be applied to sets of multivariate data, but it does not consider the restrictions imposed by the topology of a network. This problem was addressed by Zhang el al. in 2007 [10]. These authors, using the approach of White and Smyth [11], perform a spectral decomposition of the network to obtain a Euclidean embedding. In this form, the nodes are represented in a c-dimensional Euclidean space, being c the number of communities. Using the Euclidean representation of the network, the authors apply fuzzy c-means in its original formulation. The process is repeated for different values of c until the modularity is maximized. The

procedure is sound, although the Euclidean embedding is just an approximate way to represent a network.

A different approach is the one proposed by Nepusz et al. in 2008 [12]. These authors define an error functional where the topology of the network is accounted for through the network adjacency matrix. In particular, the probability of sharing the same communities is used as a similarity index between nodes. This index is compared to the corresponding element of the adjacency matrix. In this form, the error functional is defined as the sum of square differences between the real network topology, and the one predicted by the community distribution. To obtain the desired membership coefficients the authors apply steepest descent to minimize the functional [12]. As in the work of Zhang et al. [10] the number of communities is determined by maximizing a fuzzy modularity index. However, steepest descent is slow and unreliable, especially when the function to minimize is flat near the minimum [13].

A different standpoint was used by Zhang et al. [14]. These authors introduced a method based on Non-negative Matrix Factorization (NMF) [15]. Thus, NMF is applied to a normalized diffusion kernel that accounts for long-range relationships between nodes induced by the network local topology. The factorization is achieved by minimizing the root-mean-square difference between the normalized kernel matrix and the product of the factorizing matrices. The factorization matrices provide the information needed to assign memberships of nodes to communities. Again, the number of communities is determined by maximizing the modularity index. Zhao et al., [16] reformulate the method to transform the original NMF into a symmetric-NMF [15], which only uses one factorization matrix. This method is equivalent to Kernel K-means clustering and Laplace spectral clustering [17]. Later, Psorakis et al. [18] introduced a Bayesian NMF approach, which allows determining the number of communities as part of the procedure. Therefore, no maximization of the modularity function is needed. However, the intrinsic cubic complexity of matrix multiplication makes NMF based methods hard to apply on large networks.

A different approach to fuzzy community detection was presented by Liu [19]. This author proposed an approach derived from a deterministic framework for network partition based on the optimal prediction of a random walker Markovian dynamics [20]. The problem is transformed in a minimization of the quadratic error between the induced network and the fuzzy membership predicted transition matrices. For minimization, the author used several approaches based on projected steepest descend and conjugate gradient. Although from a different point of view, the procedure is similar to the one proposed in [12].

Recently, Wang et al. [21] have developed a new method that uses a structural similarity index to measure the fuzzy relationship between vertices based on local interactions between neighbors. The method operates determining the transitive closure of the fuzzy relation matrix defined by the similarity among nodes. The authors show that the developed algorithm exhibits a quadratic complexity on the number of nodes of the network.

In this work, we develop a general treatment appropriate for fuzzy community detection in social networks based on a topological error minimization in weighted, unweighted, directed and undirected networks. An algorithmic design pattern is presented implementing the procedure. Furthermore, we analyze the corresponding algorithmic complexity to determine the factors responsible for its efficiency. From the results, we improve the performance by reducing the number of iterations in the minimization procedure resorting to a conjugate gradient approach. In this form, we define a new algorithm: the conjugaTe gRadIent Based fUzzy commuNity dEtection (TRIBUNE) algorithm. The performance of TRIBUNE is determined by defining and applying a new benchmark network tailored for fuzzy community detection studies. In addition, we analyze the behavior of TRIBUNE when dealing with networks of increasing size.

II. Methodology

In this section, we present a general approach to fuzzy community detection in networked systems such as social networks. In addition, a benchmark network especially suited for testing fuzzy community detection algorithms is also introduced.

A. General formalism

Let us consider a network represented by a graph $G\ (N, E)$, where $N = \{n_1, n_2, \ldots, n_N\}$ is the set of $N = |N|$ nodes and $E = \{(n_i, n_j) \mid n_i, n_j \in N\}$ is the set of edges defining graph G. In addition, assume we have a total of c fuzzy communities in G. Finally, let us have some relationship index between nodes, $s_{ij} \in [0, 1]$ and let u_{ik} be the membership index measuring the participation of node i in community k. Considering the participation of node i in the c communities, we have

$$\sum_{m=1}^{c} u_{im} = 1 \ \forall\ i \in N \tag{1}$$

The set of the u_{ik} values for a given node, i, defines the membership vector of i, $\mathbf{u_i}$. In turn, the set of membership vectors for the N nodes in the network defines the membership matrix,

$$\boldsymbol{V} \in \mathbb{R}^{N\,x\,c} \tag{2}$$

Finally, let us consider that the relationship between nodes i and j, s_{ij}, is functionally related to their membership vectors $\mathbf{u_i}$, $\mathbf{u_j}$, and, therefore, to the membership matrix,

$$s_{ij} = s_{ij}(\boldsymbol{V}) \tag{3}$$

A useful way to define the difference between the topology induced by a given distribution of nodes into communities and a reference topology given by a set of edges with normalized weight coefficients $r_{ij} \in [0, 1]$ is through the quadratic error functional,

$$D_G(\boldsymbol{V}) = W^2 \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \left(r_{ij} - s_{ij}\,(\boldsymbol{V})\right)^2 \tag{4}$$

where W is the weight normalization factor. Since W is constant, we can neglect it from the remaining treatment without loss of generality. The present approach, eq. (4), resembles the one proposed by Bezdeck [9] for multivariate data, and it is similar to the one used by Nepusz et al. [12].

To determine the membership matrix, $\boldsymbol{V}$, we minimize eq. (4) under the constrains defined by eq. (1). One possibility is to resort to Lagrange multipliers to introduce the N conditions of eq. (1) into eq. (4). However, here we introduce explicitly the N constrains represented by eq. (1) considering that

$$u_{ic} = 1 - \sum_{m=1}^{c-1} u_{im} \quad \forall\, i \in N \tag{5}$$

In this form, we reduce the number of variables in $D_G(\boldsymbol{V})$. This fact permits to reduce the size of the data structures used in software implementations of the current approach.

Using eq. (5), and taking into account that eq. (1) is a nonlinear functional, our problem reduces to the unconstrained minimization problem,

$$min\,\{D_G(\boldsymbol{U}) \mid \boldsymbol{U} \in \mathbb{R}^{N\,x\,c-1}\} \tag{6}$$

where now $\boldsymbol{U}$ is the matrix obtained from $\boldsymbol{V}$ by removing column c. To minimize eq. (6) let us consider the variation of $D_G(\boldsymbol{U})$ on a direction $\boldsymbol{h}$ given by an increment $\alpha\boldsymbol{U}$ from the original $\boldsymbol{U_0}$ matrix. Expanding D_G in a Taylor series we get,

$$D_G(\boldsymbol{U_0} + \alpha\boldsymbol{U}) = D_G(\boldsymbol{U_0}) + \alpha \cdot \nabla D_G(\boldsymbol{U_0})^T \cdot \boldsymbol{U} + (\alpha^2/2)\, \boldsymbol{U}^T \cdot \boldsymbol{H}(\boldsymbol{U_0}) \cdot \boldsymbol{U} + \cdots \tag{7}$$

where $\nabla D_G(\boldsymbol{U_0})$ is the gradient of D_G at $\boldsymbol{U_0}$ and $\boldsymbol{H(U_0)}$ represents the D_G Hessian at the same point.

Considering small variations from the original value $\boldsymbol{U_0}$, we can limit eq. (7) to the linear term obtaining,

$$D_G(\boldsymbol{U_0} + \alpha\boldsymbol{U}) \cong D_G(\boldsymbol{U_0}) + \alpha \cdot \nabla D_G(\boldsymbol{U_0})^T \cdot \boldsymbol{U} \tag{8}$$

Minimization can be achieved by selecting directions $\boldsymbol{U}$ which always decrease the value of D_G, *i.e,* descent directions. From a mathematical standpoint, $\boldsymbol{U}$ is a descent direction if the directional derivative of $D_G(\boldsymbol{U_0})$ in the $\boldsymbol{U}$ direction is negative. Therefore, for eq. (8) the descent direction condition yields,

$$\nabla D_G(\boldsymbol{U_0})^T \cdot \boldsymbol{U} < 0 \tag{9}$$

In addition, since we look for a minimum, α in eq. (8) must be chosen as the value minimizing $D_G(\boldsymbol{U})$ along the search direction [13].

The previous considerations can be used to derive a general algorithmic design pattern for unconstrained minimization-based fuzzy community detection. Thus, we resort to the greedy design paradigm [22] applying eq. (8) iteratively as a series of locally optimal minimizations subject to the constrain defined by eq. (9). The result is shown in Algorithm 1, where $\boldsymbol{U_0}$ represents the initial membership matrix. Algorithm 1 shows, see step 8, that a stopping condition is needed. Usually, this involves several factors such as a maximum gradient value, a maximum function change or a maximum number of iterations. The specific stopping condition for the algorithms considered in this work is detailed later in section IV. A.

Algorithm 1 can generate different minimization algorithms depending on how the search directions, $\boldsymbol{h_i}$, are computed. The simplest approach is to consider the greatest decrease when going from $D_G(\boldsymbol{U_0})$ to $D_G(\boldsymbol{U_0} + \alpha\boldsymbol{U})$. To such an end, the term $\alpha \cdot \nabla D_G(\boldsymbol{U_0})^T \cdot \boldsymbol{U}$ in eq. (8) must be as small as possible. Since $\nabla D_G(\boldsymbol{U_0})^T \cdot \boldsymbol{U} = \| \nabla D_G(\boldsymbol{U_0})^T \| \cdot \| \boldsymbol{U} \| \cos\theta$, the minimum value is obtained for $\cos\theta = -1$. Therefore, $\boldsymbol{U} = -D_G(\boldsymbol{U_0})$ gives us the direction of maximum decrease. Using this result, we can customize Algorithm 1 by computing the search direction as $\boldsymbol{h_i} = -\nabla D_G(\boldsymbol{U_i})$. The resulting algorithm, see Algorithm 2, is just the classical steepest descent used, for instance, in [12].

Algorithm 1. *Algorithmic design pattern for minimization-based fuzzy community detection in networks*

1: Set $\boldsymbol{U_0}$
2: i ← 0
3: go_on ← true
4: compute initial search direction $\boldsymbol{h_0}$ such that $\nabla D_G(\boldsymbol{U_0})^T \cdot \boldsymbol{h_0} < 0$
5: **while** (go_on) **do**
6: get α_i for min $\{D_G(\boldsymbol{U_i} + \alpha_i \cdot \boldsymbol{h_i}) \mid \alpha_i \geq 0\}$
7: compute $\boldsymbol{U_{i+1}} \leftarrow \boldsymbol{U_i} + \alpha_i \cdot \boldsymbol{h_i}$
8: **if** stopping condition **then**
9: go_on ← false
10: **else**
11: compute search direction $\boldsymbol{h_{i+1}}$ such that $\nabla D_G(\boldsymbol{U_{i+1}})^T \cdot \boldsymbol{h_{i+1}} < 0$
12: i ← i+1
13: **end_if**
14: **end_while**

To implement any realization of the pattern shown in Algorithm 1, we need the gradient vector. Thus, we derive $D_G(\boldsymbol{U})$ with respect to each u_{lk}, with $1 \leq l \leq$ N and $1 \leq k \leq$ c-1 obtaining,

$$\partial D_G(\boldsymbol{U})/\partial u_{lk} = -2\left[\sum_{i=1}^{N}\sum_{j=1}^{N}(r_{ij} - s_{ij}(\boldsymbol{U})) \cdot \frac{\partial s_{ij}(\boldsymbol{U})}{\partial u_{lk}}\right] \tag{10}$$

Algorithm 2. *Steepest descent-based fuzzy community detection algorithm*

1: Set $\boldsymbol{U_0}$
2: i ← 0
3: go_on ← true
4: compute $\boldsymbol{h_0} = -\nabla D_G(\boldsymbol{U_0})$
5: **while** (go_on) **do**
6: get α_i for min $\{D_G(\boldsymbol{U_i} + \alpha_i \cdot \boldsymbol{h_i}) \mid \alpha_i \geq 0\}$
7: compute $\boldsymbol{U_{i+1}} \leftarrow \boldsymbol{U_i} + \alpha_i \cdot \boldsymbol{h_i}$
8: **if** stopping condition **then**
9: go_on ← false
10: **else**
11: $\boldsymbol{h_{i+1}} = -\nabla D_G(\boldsymbol{U_{i+1}})$
12: i ← i+1
13: **end_if**
14: **end_while**

and, since s_{ij} depends only on $\mathbf{u_i}$ and $\mathbf{u_j}$,

$$\partial D_G(\boldsymbol{U})/\partial u_{lk} = -2\left[\sum_{j=1}^{N}(r_{lj} - s_{lj}(\boldsymbol{U}))\cdot\frac{\partial s_{lj}(\boldsymbol{U})}{\partial u_{lk}} + \sum_{i=1}^{N}(r_{il} - s_{il}(\boldsymbol{U}))\cdot\frac{\partial s_{il}(\boldsymbol{U})}{\partial u_{lk}}\right] \quad (11)$$

Equation (11) represents a general formulation of $\nabla D_G(\boldsymbol{U})$ depending only on the specific form of the $s_{ij} = s_{ij}(\boldsymbol{U})$ relationship. In addition, eq. (11) considers independently the lj from the il edges. Therefore, it is possible to describe directed networks where the existence of a pq edge does not imply the existence of a qp one. In the usual case of an undirected network, where $r_{ij} = r_{ji}$ and $s_{ij} = s_{ji}$, eq. (11) simplifies to

$$\partial D_G(\boldsymbol{U})/\partial u_{lk} = -4\sum_{i=1}^{N}(r_{il} - s_{il}(\boldsymbol{U}))\cdot\frac{\partial s_{il}(\boldsymbol{U})}{\partial u_{lk}} \quad (12)$$

B. Asymptotic complexity

Algorithm 1 represents the general structure of any minimization-based fuzzy community detection algorithm. So, it can be used to identify the factors determining the time asymptotic complexity of these algorithms, *i. e.*, its running time.

Algorithm 1 shows, step 1, that an initialization of the N · (c-1) u_{ik} coefficients is carried out. This is a procedure of at least O(N·c) complexity. Next, we find a loop between steps 5 and 14, which is repeated a total of m times. This m represents the number of iterations needed for any minimization-based method to converge. Thus, the loop exhibits O(m) complexity. In turn, within the loop, we compute the α factor in step 6. This factor is obtained using a numerical procedure iterated a total of p times. Within each of these p iterations, D_G is needed, which, as shown in eq. (4), uses N^2 differences involving the s_{ij} coefficients. These coefficients are computed from the membership vectors. Due to the orthogonality of the c-1 independent vector components, this is at least a process of O(c) complexity. Thus, step 6 exhibits $O(p\cdot N^2\cdot c)$ complexity. In addition, updating of the membership matrix in step 7 contributes as a O(N·c) procedure. Also, within the loop, step 11, we determine the search direction $\boldsymbol{h}_i$ which involves at least computation of the gradient as exemplified in the steepest descent approach above introduced. Determination of the N·(c-1) gradient components, eq. (11), is a $O(N^2\cdot c^2)$ complexity process since each gradient component involves at least N s_{ij} terms. Considering that the asymptotic behavior is determined by the fastest growing term [22] and that the number of nodes is the dominant factor here, we obtain,

$$O(N\cdot c) + O(m)\cdot\left(O(p\cdot N^2\cdot c) + O(N\cdot c) + O(N^2\cdot c^2)\right) = O(m\cdot N^2\cdot[p\cdot c + c^2]) \quad (13)$$

Since the number of communities for a given problem, c, is fixed, the running time depends quadratically on the number of nodes, N, and linearly on both, the number of iterations in the minimization procedure, m, and in the computation of the α factor, p. The quadratic dependence on N was first pointed out by Nepusz et al. [12] and is inherent to any method based on the functional given by eq. (4). To increase performance, decreasing the running time, in this work we focus in reducing the value of the m multiplicative factor.

C. Increasing performance

To determine the performance of a given realization of the design pattern collected in Algorithm 1, we need an appropriate baseline. As such, we choose steepest descent, which is the simplest approach and represents the canonical example of gradient-based minimization algorithms. A key characteristic of steepest descent is that its rate of convergence is slow. The main problem is that each new gradient direction is perpendicular to the previous one. So, when the curvature is very different in different directions or the function is very flat near the minimum the method needs a large, sometimes huge, amount of iterations [13].

A much more efficient option is given by the conjugate gradient method, originally introduced by Hestenes and Stiefel [23] as an iterative way to solve systems of linear equations. Later, Fletcher and Reeves [24] adapted it to minimize non-linear functions. The main characteristic of the method is that it generates, in a very simple way, linearly independent search directions with the conjugation property [13, 25, 26]. This prevents spoiling the function decreases obtained in previous iterations. The method, customized to our case, generates two sequences of gradient and search direction vectors, $\boldsymbol{g}_i$ and $\boldsymbol{h}_i$, according to,

$$\begin{aligned}\boldsymbol{g}_i &= \boldsymbol{g}_{i-1} - \lambda_{i-1}\boldsymbol{H}\,\boldsymbol{h}_{i-1}\\ \boldsymbol{h}_i &= \boldsymbol{g}_i + \beta_{i-1}\,\boldsymbol{h}_{i-1}\end{aligned} \quad (14)$$

where $\boldsymbol{H}$ is the $D_G(\boldsymbol{U})$ Hessian matrix and $\boldsymbol{h_0} = \boldsymbol{g_0} = -\nabla D_G(\boldsymbol{U}_0)$. It can be proved that the Hessian is not explicitly needed if $\boldsymbol{g_{i+1}}$ is computed as the minus gradient at the minimum along the new search direction [13]. Thus, the new membership matrix can be computed as,

$$\boldsymbol{U_{i+1}} = \boldsymbol{U_i} + \alpha_i\cdot\boldsymbol{h_i} \quad (15)$$

where $\boldsymbol{\alpha}_i$ is such that,

$$D_G(\boldsymbol{U_i} + \alpha_i\cdot\boldsymbol{h_i}) = min\{D_G(\boldsymbol{U_i} + \alpha\cdot\boldsymbol{h_i}) \mid \boldsymbol{\alpha} \geq \boldsymbol{0}\} \quad (16)$$

The resulting conjugaTe gRadIent-Based fUzzy commuNity dEtection (TRIBUNE) algorithm is shown in Algorithm 3.

The key step in Algorithm 3 is the computation of the β_i factor in step 12. The original β_i expression given by Fletcher and Reeves [24] generates cycles of small variations on the search direction when a small variation is found [27, 28]. Therefore, we use the Polak-Ribière-Polyak variant [29, 30], which has shown to be a very efficient way to compute β_i [13]. Thus,

$$\beta_i = (\boldsymbol{g}_{i+1} - \boldsymbol{g}_i)^T\cdot\boldsymbol{g}_{i+1} / (\boldsymbol{g}_i{}^T\cdot\boldsymbol{g}_i) \quad (17)$$

Eq. (17), is able to self-reset the method, using a steepest descent step, when a small variation of search direction appears, which solves the cycle formation problem.

Algorithm 3. *ConjugaTe gRadIent-Based fUzzy commuNity dEtection (TRIBUNE) algorithm*

1: Set $\boldsymbol{U}_0$
2: i ← 0
3: go_on ← true
4: compute $\boldsymbol{h}_0 \leftarrow \boldsymbol{g}_0 \leftarrow -\nabla D_G(\boldsymbol{U}_0)$
5: **while** (go_on) **do**
6: get α_i for min $\{D_G(\boldsymbol{U}_i + \alpha_i \cdot \boldsymbol{h}_i) \mid \alpha_i \geq 0\}$
7: compute $\boldsymbol{U}_{i+1} \leftarrow \boldsymbol{U}_i + \alpha_i \cdot \boldsymbol{h}_i$
8: **if** stopping condition **then**
9: go_on ← false
10: **else**
11: compute $\boldsymbol{g}_{i+1} \leftarrow -\nabla D_G(\boldsymbol{U}_{i+1})$
12: compute β_i
13: compute $\boldsymbol{h}_{i+1} \leftarrow \boldsymbol{g}_{i+1} + \beta_i \cdot \boldsymbol{h}_i$
14: i ← i+1
15: **end_if**
16: **end_while**

From the algorithmic complexity point of view the difference between Algorithm 3 and Algorithm 2 is the computation and use of $\boldsymbol{g}_{i+1}$ and β_i in steps 11 and 12, respectively. These steps represent an additional $O(m \cdot N \cdot c)$ term, see eq. (17). Including this term in eq. (13), and considering that for the asymptotic complexity only the term with highest variation rate is meaningful, the dependence of Algorithm 3 on m, p and N is the same as in Algorithm 2.

What remains is to quantify the performance increase of TRIBUNE, Algorithm 3, when compared to the steepest descent baseline, Algorithm 2. For that, we need to select a specific network type and a benchmark network model.

D. The case of undirected and unweighted networks

The general treatment presented above can be customized just by defining the nature of both the relationship function, $s_{ij}(\boldsymbol{U})$, and the reference values r_{ij}. Thus, for the typical case in social network analysis of undirected and unweighted networks we have,

$$s_{ij}(\boldsymbol{U}) = \boldsymbol{u_i} \cdot \boldsymbol{u_j} = \sum_{k=1}^{c} u_{ik} \cdot u_{jk} = \sum_{k=1}^{c-1} u_{ik} \cdot u_{jk} + (1 - \sum_{k=1}^{c-1} u_{ik}) \cdot (1 - \sum_{k=1}^{c-1} u_{jk}) \quad (18)$$
$$r_{ij} = a_{ij}$$

where we made use of the relationship function introduced by Nepusz et al. [12]. This function is defined by the dot product of the membership vectors $\boldsymbol{u_i}$ and $\boldsymbol{u_j}$. In probabilistic terms, this represents the probability that nodes i and j belong simultaneously to some community. On the other hand, the reference value r_{ij}, see eq.(4), corresponds to a_{ij}, the ij element of the adjacency matrix, **A**. This value is one if i and j are connected and zero otherwise. In addition, for undirected networks we have

$$s_{ij} = s_{ji} \ \wedge \ a_{ij} = a_{ji} \ \wedge \ a_{ii} = 0 \ \forall \ i \quad (19)$$

Then, considering eq. (5) and substituting eq. (18) and eq. (19) in eq. (4) we obtain,

$$D(\boldsymbol{U})_G = \sum_{i=1}^{N}\left[\sum_{m=1}^{c-1} u_{im}{}^2 + \left(1 - \sum_{m=1}^{c-1} u_{im}\right)^2\right]^2 + 2\sum_{i}^{N}\sum_{j>i}^{N}\left[a_{ij} - \sum_{m=1}^{c-1} u_{im} \cdot u_{jm} - \left(1 - \sum_{m=1}^{c-1} u_{im}\right) \cdot \left(1 - \sum_{m=1}^{c-1} u_{jm}\right)\right]^2 \quad (20)$$

In turn, from eq. (18) we have

$$\partial s_{ij}/\partial u_{lk} = \left(u_{jk} - 1 + \sum_{m=1}^{c-1} u_{jm}\right)\delta_{il} + \left(u_{ik} - 1 + \sum_{m=1}^{c-1} u_{im}\right)\delta_{jl} \quad (21)$$

where δ_{ab} represents the Kronecker delta, which equals one if a=b and is zero otherwise. Substituting eq. (21) in eq. (12) we get

$$\partial D_G(\boldsymbol{U})/\partial u_{lk} = 4\sum_{i=1}^{N}[a_{il} - \sum_{m=1}^{c-1} u_{im} \cdot u_{lm} - (1 - \sum_{m=1}^{c-1} u_{im}) \cdot (1 - \sum_{m=1}^{c-1} u_{lm})] \left(1 - \sum_{m=1}^{c-1} u_{im} - u_{ik}\right) \quad (22)$$

Equations (20) and (22) are the ones used to test Algorithm 3. What remains is to define a benchmark network model appropriate for analyzing the correctness of results when applying fuzzy community detection algorithms.

E. A benchmark network model for fuzzy community detection

The experimental analysis of the performance of Algorithm 3 for community detection needs an appropriate benchmark network model. As requirements, the benchmark must contain the characteristics found in real social networks and exhibit an unambiguous community structure easy to analyze. This last property is particularly important for large networks. Therefore, on one hand, the proposed model includes the scale-free degree distribution found in real network by using Barábasi-Albert networks, BA, as in the Lancichinetti and Fortunato benchmark [31]. On the other hand, the model incorporates the ease of interpretation of the butterfly data model [9, 32].

Thus, our model consists of a BA network, which is repeated, cloned, c times defining a set of c different communities. In addition, the last node of each repeated network is connected to a central node in a similar way to the connection of the two identical sets of data through a central point in the butterfly model [9, 32]. The resulting network model is shown in Fig. 1.

Given N nodes for each identical BA subnetwork, the total number of nodes in the benchmark network is c·N+1. Let us represent the membership vector of node i in subnetwork (community) k as $\boldsymbol{u}_i(k)$ and the membership vector of the central node as $\boldsymbol{u}$. Due to the symmetry of the system, we have for the $\boldsymbol{u}_i(k)$ Euclidean norm and for the $\boldsymbol{u}$ vector components:

$$\|\boldsymbol{u}_i(k)\|_2 = \|\boldsymbol{u}_i(l)\|_2 \quad \forall k \neq l \mid 1 \leq k,l \leq c \quad \wedge \quad 1 \leq i \leq N$$
$$u_1 = u_2 = \cdots = u_c = \frac{1}{c} \tag{23}$$

The previous expressions will be used in all our tests to determine whether Algorithms 2 and 3 correctly identify the community structure in BA-communities benchmark networks.

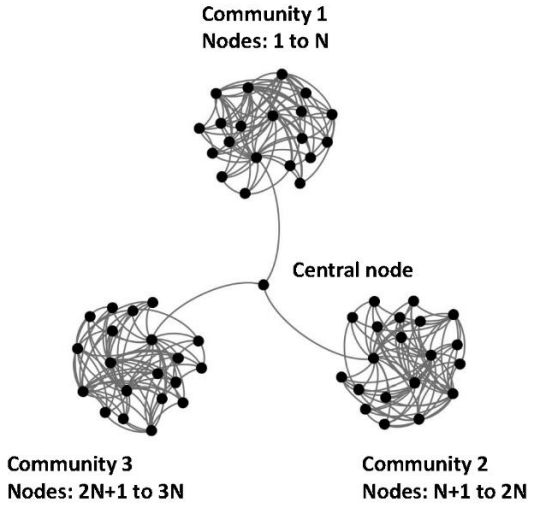

Fig. 1. Schematic representation of the BA-communities benchmark network model proposed in this work. The figure represents a network with three communities.

III. Computational details

As shown in Algorithm 1, fuzzy community detection in networks needs an initial membership matrix $\boldsymbol{U}_0$. A random initialization is a possible solution, as proposed in [12]. However, the dependence of the error functional on the N · (c-1) membership coefficients, see eq. (4) and eq. (20), defines a very large error hypersurface. On this hypersurface many different minima are possible. Therefore, using a random initialization, the starting point of different minimizations could be in the vicinity of different error minima. The problem is that any gradient-based minimization leads to the minimum closest to the initial point on the hypersurface. So, to obtain comparable results corresponding to the same initial conditions, we need to ensure the starting point on the error hypersurface to be always the same. To such an end, considering a given membership value "a", we initialize the components of the membership vectors of each BA subnetwork as $u_{ik}(k)$ = a and u_{im} $(m \neq k)$ = b with b = (1-a)/(c-1). For the central node, we use u_1 = a and $u_{m\neq 1}$ = b. As "a" value we arbitrarily select a = 0.6.

For the generation of the BA-communities benchmark networks, we adapt the BA network generation approach described in [33] to the case of undirected networks. As a good and efficient general random number generator for creation of BA networks, we implement the XORShift64* Marsaglia generator [34]. Finally, in the different performance tests carried out with Algorithms 2 and 3, we apply the Brent method [35] for computing the α_i parameter through an exact line search.

Since the adjacency matrices for the benchmark networks are sparse, we develop a sparse array data structure based on hash tables. This data structure defines an N sized array of hash tables. Each element of this array represents a row of the adjacency matrix. Only non-zero elements are added to each row.

Algorithms 2 and 3 and all the associated classes and methods have been implemented in Java and included as a new package, networkCommunities, of the APINetworks Java software tool [36, 37]. In all the tests performed, the system used has been an Octa-Octa-Core Intel® Xeon® E5-2630 v3 (2.4 GHz) with 64 GB. This machine is a node of a cluster running OpenHPC (based on CentOS 7.3) [38].

IV. Results and discussion

In this section, we determine the relative performance of the new TRIBUNE algorithm against the baseline, its dependence on the network size, and its ability to deal with large networks. As previously stated, the starting point is to determine an appropriate stopping condition for Algorithms 2 and 3.

A. Stopping condition

Any realization of Algorithm 1 needs an appropriate stopping condition to determine that the minimization has achieved a sufficiently good level. A direct and simple condition based on the Euclidian norm such as $\|\nabla D_G(\boldsymbol{U_i})\|_2 \leq \epsilon$ is inadequate since the magnitude of $\nabla D_G(\boldsymbol{U_i})$ changes greatly with the size of the network. A popular approach is to relate the gradient Euclidian norm to the function using the expression [26],

$$\|\nabla D_G(\boldsymbol{U_i})\|_2 \leq \varepsilon\left(1 + D_G(\boldsymbol{U}_i)\right) \tag{24}$$

where ϵ is a small value. Eq. (24) is relatively invariant on the gradient and function scaling [39], and the 1 added to the right side prevents the use of a very strict condition if the function is too small. However, eq. (24) can be considered just as a first-order approach to the actual variation of the gradient Euclidian norm with the function. Here, to define a more realistic stopping condition for the tests considered, this variation is empirically determined. Thus, we generate benchmark networks with four communities and sizes for each individual BA subnetwork ranging from 100 and 2200 nodes in increments of 100. This corresponds to benchmark network sizes ranging from 401 to 8801 nodes. To each benchmark network, we apply the baseline, Algorithm 2, until the memberships of the central node are $\leq 10^{-2}$ times (0.01%) the exact value corresponding to four communities: 0.25. In addition, the difference between the Euclidean norm of membership vectors for equivalent nodes in different communities, see first relationship in eq. (23), is also required to be $\leq 10^{-2}$. Since the generation of the BA subnetworks involves a random process, we average the results for three equal sized benchmark networks. The resulting average variation of the gradient Euclidian norm with the function is shown in Fig. 2. The results show that the error function takes large values at the minimum. Thus, for instance, even for the smallest network, 401 nodes, the error function stabilizes at a value around 10500. In addition, we observe that the gradient

Euclidian norm follows closely a power law variation on $D_G(\boldsymbol{U})$. In particular, a regression analysis yields $\|\nabla D_G(\boldsymbol{U})\|_2 = 0.042 \cdot D_G(\boldsymbol{U})^{0.479}$ with coefficient of determination $r^2 = 0.985$, see continuous line in Fig. 2. To ensure convergence for networks of very different sizes, we choose a tighter convergence criterion: $\|\nabla D_G(\boldsymbol{U})\|_2 \leq 2 \cdot 10^{-2} \cdot \sqrt{D_G(\boldsymbol{U})}$, see dotted line in Fig. 2. Thus, we select this expression as the first component of the stopping condition. Furthermore, for those cases where this requirement is too demanding, we introduce a complementary condition related to consecutive values of $D_G(\boldsymbol{U})$: $|D_G(\boldsymbol{U}_{i+1}) - D_G(\boldsymbol{U}_i)| \leq 10^{-1}$. This value has been selected since all the $D_G(\boldsymbol{U})$ differences for the two last iterations in all the tests carried out are at most 10^{-1}. In addition, to ensure the procedure always ends, we set a maximum of 300 iterations for the optimization algorithm.

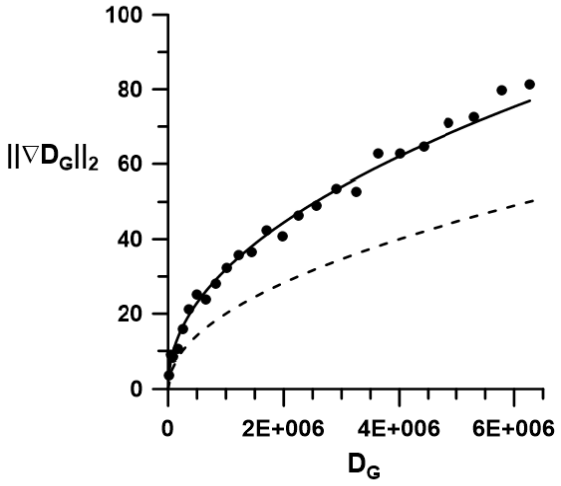

Fig. 2. Average variation of the gradient Euclidian norm versus the error function. The circles indicate the experimental data used in the study. The continuous line is the regression function obtained from the experimental data as described in the text.

B. Relative performance of TRIBUNE

To determine the relative performance of TRIBUNE against the baseline, Algorithm 2, we generate BA-communities benchmarks with four BA subnetworks (*i.e.*, communities). The BA subnetwork sizes range from 200 to 4000 nodes in increments of 200. This generates BA-communities benchmark networks of sizes ranging from 801 to 16001 nodes. Fig. 3. collects the time in seconds, T, needed by each algorithm to process the different benchmarks as a function of the network size, N. Considering the quadratic dependence of the time on N, see eq. (13), Fig. 3 includes fits to quadratic functions of the form, $T = a \cdot N^2$.

The results show that in both cases, the time follows closely the predicted variation on N with a regression equation of $T=3.9 \cdot 10^{-5} \cdot N^2$ and coefficient of determination $r^2 = 0.985$ for the baseline, and $T=6.7 \cdot 10^{-6} \cdot N^2$ with coefficient of determination $r^2 = 0.935$ for TRIBUNE. Fig. 3 also shows that TRIBUNE is significantly more efficient than the baseline. To quantify this effect, we compute speedups as the quotient between the time used by Algorithm 2 with respect to TRIBUNE for the same network size. We find the speedup to follow an increasing trend with network size. The speedup ranges from 3.5 for the smallest network (N=801) to 5.6 for the largest one (N = 16001). The average (including standard deviation) is a speedup of 6.2±1.5, which corresponds to a reduction in running time of 84%.

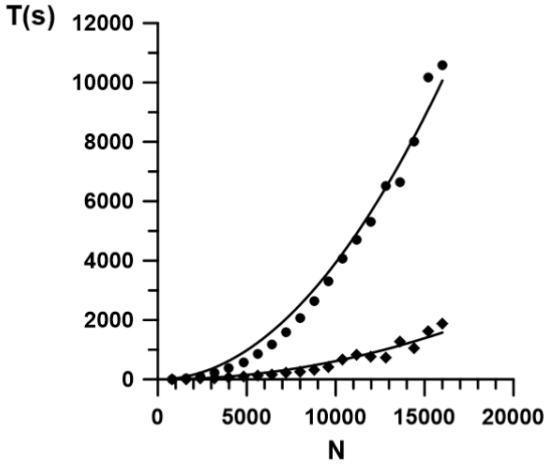

Fig. 3. Running time, in seconds, needed to determine the fuzzy community structure of several BA-communities benchmark networks versus network size, see text. The circles represent the experimental results for the steepest descent baseline whereas the diamonds correspond to the TRIBUNE algorithm. The continuous lines represent the regression curves for the two sets of data as described in the text.

The reason behind the better performance of TRIBUNE with respect to the baseline becomes apparent when we analyze the number of iterations used in the optimization, see Fig. 4. Thus, in Algorithm 2 (steepest descent), the iterations increase with the network size. However, the number of iterations used by TRIBUNE is fairly independent from the network size. So, in practice, for the set of considered networks, TRIBUNE removes the dependence on the m factor from the computational complexity of the procedure, see eq. (13). In addition, the efficiency difference between the two algorithms increases with the network size as a consequence of the dependence, in Algorithm 2, of the number of iterations on N. In particular, Algorithm 2 needs 42 iterations to achieve convergence for the smallest network (N=801), whereas 162 are needed for the largest one (N=16001). In average (including standard deviation), 121.1±35.0 iterations are used. TRIBUNE, on the other hand, needs 14 iterations to attain convergence for the smallest network and just 26 for the largest one. In this case, the average is 17.3±3.6 iterations. Using the previous average values, we observe that TRIBUNE, is seven times more efficient than the baseline in achieving convergence.

C. Performance of the TRIBUNE algorithm on large networks

To consider real social networks, it is necessary to know the practical limits of the TRIBUNE algorithm. The quadratic

dependence of the running time on the number of nodes is a limiting factor for dealing with large problems. To determine the

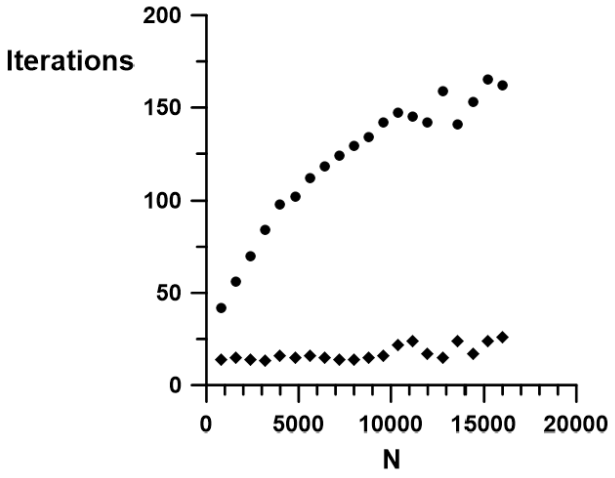

Fig. 4. Number of iterations in the optimization procedure for BA-communities benchmark networks of different sizes. The circles represent the experimental results for the steepest descent baseline whereas the diamonds correspond to the TRIBUNE algorithm.

practical upper limit, we measure the time needed to apply TRIBUNE for networks of increasing size. As a reference, we select arbitrarily a limit of 24 hours. In the tests, we consider BA-communities benchmark networks with four BA subnetworks. The initial size of each subnetwork is 5000 nodes and increments of 5000 nodes are used until the running time exceeds the maximum allowed. Fig. 5 collects the results of running time, in hours, versus network size, including a fit to a quadratic function of the form, $T = a \cdot N^2$. We observe that BA-communities benchmark networks of sizes ranging from 20001 to 100001 nodes are tractable within the 24 hours limit. In addition, the results show that the running time follows the predicted variation on N with a regression function of $T=1.8 \cdot 10^{-9} \cdot N^2$ and coefficient of determination $r^2 = 0.979$. The largest network, 100001 nodes, is processed in a time frame of 18 hours. This network size is not especially large. However, it is enough for the analysis of many instances of social, citation, collaboration or temporal networks [40] or for the treatment of bioactive compounds networks generated within the context of the chemical space [41].

V. Conclusions

In this work, we present a general formalism for fuzzy community detection in weighted, unweighted, directed and undirected social networks. The key point is the definition of a functional measuring the error between the actual network topology, and the one predicted by the distribution of nodes into communities. Thus, the problem is transformed into an unconstrained minimization, which allows to develop a general algorithm design pattern based on the greedy paradigm. The complexity order of the procedure is found to depend quadratically on the number of network nodes, N, and linearly on the number of iterations, both in the minimization procedure, m, and in the linear search applied in gradient-based minimizations, p.

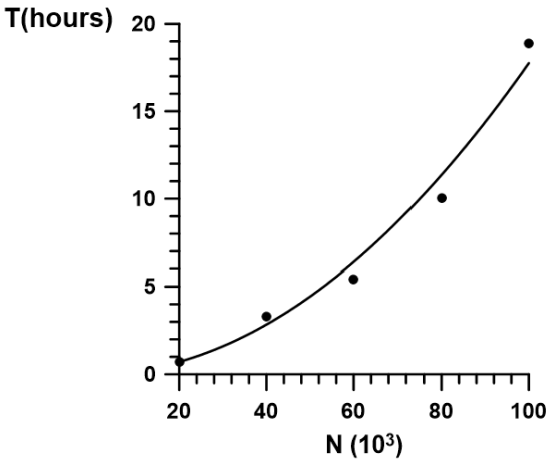

Fig. 5. Running time, in hours, needed to determine the fuzzy community structure of BA-communities benchmark networks of increasing size (measured in thousands of nodes). The continuous line corresponds to a quadratic regression as described in the text.

To reduce the effect of m, we introduce a conjugate gradient-based approach leading to a new algorithm, TRIBUNE. In addition, to perform all the tests needed, we develop a fuzzy community detection benchmark network combining the real behavior of Barábasi-Albert networks with the ease of interpretation of the butterfly data model. Using this benchmark, TRIBUNE is found to be significantly more efficient than the steepest descent baseline. In particular, in average TRIBUNE reduces the running time for the set of test networks in 84%. More importantly, the efficiency is found to increases with the network size. Analyzing the variation of the algorithm running time with the network size, we find that networks of 100000 nodes can be processed in a time frame of 18 hours using a Java implementation of TRIBUNE in current computational systems. This result can be clearly improved by using a compiled language such as C++, more efficient data structures, and parallelism. However, the quadratic dependence of the running time on the network size is a serious limiting factor for dealing with very large problems. It is then necessary to formulate new solutions with better algorithmic complexity.

Acknowledgements

This work was supported by the Universidad de Castilla-La Mancha [grant numbers GI20163409, GI20173975] and the Junta de Comunidades de Castilla-La Mancha [grant number SBPLY/17/180501/000149].

REFERENCES

[1] A. L. Barabási, *Network Science*. Cambridge: Cambridge University Press, 2016.

[2] M. E. J. Newman, *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc, 2010.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed., 2006, 2nd printing, 2011.

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

[5] M. E. J. Newman, N. Girvan, "Finding and evaluating community structure in networks", *Phys. Rev. E.*, vol. 69, 2004, pp. 026113-026127.

[6] M. E. J. Newman, "Fast algorithm for detecting community structure in networks", *Phys. Rev. E.*, vol. 69, 2004, pp. 066133-066137.

[7] V.D. Blondel, J.L. Guillaume, R. Lambiotte, E. Lefebvre, "Fast unfolding of communities in large networks2, *J. Stat. Mech. Theory Exp.*, vol. 10, 2008, pp. P10008 .

[8] J. C. Dunn, "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters", *Journal of Cybernetics*, vol. 3, 1973, pp. 32-57.

[9] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function algorithms*, Advanced Applications in Pattern Recognition, Plenum Press, 1981.

[10] S. Zhang, R.-S. Wang, X.-S. Zhang, "Identification of overlapping community structure in complex networks using fuzzy c-means clustering", *Phys. A: Stat. Mech. Appl.*, vol. 374(1) (2007) 483–490.

[11] S. White, P. Smyth, "A spectral clustering approach to finding communities in graphs", *SIAM International Conference on Data Mining*, 2005.

[12] T. Nepusz, A. Petróczi, L. Négyessy, F. Bazsó, "Fuzzy communities and the concept of bridgeness in complex networks", *Phys. Rev. E.*, 77 (2008) 016107.

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical recipes. The Art of Scientific Computing*. Third edition, Cambridge University Press, 2007.

[14] S. Zhang, R. S. Wang, X. S. Zhang, "Uncovering fuzzy community structure in Complex networks", *Phys. Rev. E.*, vol. 76, No. 4, 2007, pp. 046103.

[15] Cichocki, R. Zdunek, A. H. Phan, S.-I. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*, John Wiley & Sons, 1st ed., 2009.

[16] K. Zhao, S. W. Zhang, Q. Pan, "Fuzzy analysis for overlapping community structure of complex network", *Proceedings of Control and Decision Conference (CCDC)*, 2010, pp. 3976–3981.

[17] C. Ding, X. He, and H.D. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering", *Proceedings of SIAM International Conference on Data Mining (SDM'05)*, 2005, pp. 606–610.

[18] Psorakis, S. Roberts, M. Ebden, B. Sheldon, "Overlapping community detection using Bayesian non-negative matrix factorization", *Phys. Rev. E.*, vol. 83, No. 6, 2011, pp. 066114.

[19] J. Liu, "Detecting the fuzzy clusters of complex networks", *Pattern Recognition*, vol. 43, 2010, pp. 1334–1345.

[20] W.E.T. Li, E. Vanden-Eijnden, "Optimal partition and effective dynamics of complex networks", *Proc. Natl. Acad. Sci. USA*, vol. 105, 2008, pp. 7907–7912.

[21] X. Wang, G. Liu, L. Pan, J. Li, "Uncovering fuzzy communities in networks with structural similarity", *Neurocomputing*, vol. 210, 2016, pp. 26–33.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Third Edition, MIT Press, 2009.

[23] M. R. Hestenes, E. Stiefel, "Methods of conjugate gradients for solving linear systems", *J. Research Nat. Bur. Standards*, vol. 49, 1952, pp. 409–436.

[24] R. Fletcher, C. M. Reeves, "Function minimization by conjugate gradients", *Comput. J.*, vol. 7, 1964, pp. 149–154.

[25] E. Polak, *Computational Methods in Optimization. A Unified Approach*. R. Bellman, Ed., Mathematics in Science and Engineering, Vol. 77, New York, NY: Academic Press, 1971.

[26] J. Nocedal, S. J. Wright, *Numerical Optimization*, Second Edition, Springer Series in Operations Research and Financial Engineering, Springer, 2006.

[27] M. J. D. Powell, "Restart procedures of the conjugate gradient method", *Mathematical Programming*, vol. 2, 1977, pp. 241-254.

[28] J. C. Gilbert, J. Nocedal, "Global convergence properties of conjugate gradient methods for optimization", *SIAM J. Optimization*, vol. 2, 1992, pp. 21–42.

[29] E. Polak, G. Ribière, "Note sur la convergence de methods de directions conjugées", *Rev. Fr. Inform. Rech. Operation.*, vol. 16-R1, 1969, pp. 35–43.

[30] B. T. Polyak, "The conjugate gradient method in extreme problems", *USSR Comp Math and Math. Phys.*, vol. 9, 1969, pp. 94-112.

[31] A. Lancichinetti, S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities", *Phys. Rev. E.*, vol. 80, 2009, pp. 016118.

[32] E. H. Ruspini, "Numerical methods for fuzzy clustering", *Information Sciences*, vol. 2, No. 3, 1972, pp. 319-350.

[33] B. J. Prettejohn, M. J. Berryman, M. D. McDonnell, "Methods for generating complex networks with selected structural properties for simulations: a review and tutorial for neuroscientists", *Front. Comput. Neurosci.*, vol. 5, 2011, pp. PMC3059456.

[34] S. Vigna, "An experimental exploration of Marsaglia's xorshift generators, scrambled", arXiv:1402.6246v5, 2014.

[35] R. P. Brent, *Algorithms for Minimization without Derivatives*, Englewood Cliffs, NJ: Prentice-Hall, 1973 (reprinted 2002, New York: Dover).

[36] A. Niño, C. Muñoz-Caro, S. Reyes, "APINetworks: A General API for the Treatment of Complex Networks in Arbitrary Computational Environments", *Comput. Phys. Commun.*, vol. 196, 2015, pp. 446-454.

[37] C. Muñoz-Caro, A. Niño, S. Reyes, M. Castillo, "APINetworks Java: A Java approach to the treatment of large complex networks", *Comput. Phys. Commun.*, vol. 207, 2017, pp. 549-552.

[38] "OpenHPC. Community building blocks for HPC systems". [Online]. Available: https://openhpc.community/

[39] T. Schlick, "Optimization Methods in Computational Chemistry", in: K. B. Lipkowitz , D. B. Boyd (Eds.), *Reviews in Computational Chemistry*. Volume 3, VCH, 1992, pp. 26-28.

[40] "SNAP: Stanford Large Network Dataset Collection". [Online]. Available: https://snap.stanford.edu/data/

[41] A. Niño, C. Muñoz-Caro, S. Reyes, "The Network Representation of Chemical Space: A New Paradigm", in: *Theoretical & Quantum Chemistry at the Dawn of the 21st Century*, R. Carbó-Dorca & T. Chakraborty (Eds.). Toronto: Apple Academic Press, 2018, pp. 249-282.