LETTER Communicated by Yufeng Liu

A DC Programming Approach for Finding Communities in Networks

Hoai An Le Thi
hoai-an.le-thi@univ-lorraine.fr
Manh Cuong Nguyen
manh-cuong.nguyen@univ-lorraine.fr
Laboratory of Theoretical and Applied Computer Science, University of Lorraine, Ile du Saulcy, 57045 Metz, France

Tao Pham Dinh
pham@insa-rouen.fr
Laboratoire of Mathematics, National Institute for Applied Sciences—Rouen, 76801 Saint-Étienne-du-Rouvray cedex, France

Automatic discovery of community structures in complex networks is a fundamental task in many disciplines, including physics, biology, and the social sciences. The most used criterion for characterizing the existence of a community structure in a network is modularity, a quantitative measure proposed by Newman and Girvan (2004). The discovery community can be formulated as the so-called modularity maximization problem that consists of finding a partition of nodes of a network with the highest modularity. In this letter, we propose a fast and scalable algorithm called DCAM, based on DC (difference of convex function) programming and DCA (DC algorithms), an innovative approach in nonconvex programming framework for solving the modularity maximization problem. The special structure of the problem considered here has been well exploited to get an inexpensive DCA scheme that requires only a matrix-vector product at each iteration. Starting with a very large number of communities, DCAM furnishes, as output results, an optimal partition together with the optimal number of communities c^*; that is, the number of communities is discovered automatically during DCAM's iterations. Numerical experiments are performed on a variety of real-world network data sets with up to 4,194,304 nodes and 30,359,198 edges. The comparative results with height reference algorithms show that the proposed approach outperforms them not only on quality and rapidity but also on scalability. Moreover, it realizes a very good trade-off between the quality of solutions and the run time.

Neural Computation **26, 2827–2854** (2014)
doi:10.1162/NECO_a_00673

1 Introduction

In recent years, the study of complex networks has attracted a great deal of interest in many disciplines, including physics, biology, and the social sciences. Examples of such networks include web graphs, social networks, citation networks, and biochemical networks. Despite the fact that these networks belong to very distinct fields, they all surprisingly share some common structural features, such as power law degree distributions, small-world average shortest paths, and high clustering coefficients.

Community structure is one of the important network features. The entire network is in fact composed of densely connected subnetworks, with only sparse connections between them. Such subnetworks are called communities or modules. Detection of communities is of significant practical importance as it allows analyzing networks at a megascopic scale: identification of related web pages, uncovering of communities in social networks, and decomposition of metabolic networks in functional modules.

If there is a community structure in a network, the intracommunity edges should be significantly denser than the intercommunity edges. Hence, detecting communities amounts to searching for the structure that maximizes the number of intracommunity edges while minimizing the number of intercommunity edges. In 2004, Girvan and Newman (Newman, 2010; Newman & Girvan, 2004) proposed a quantitative measure, the modularity (Q measure), for characterizing the existence of community structure in a network. The modularity Q of a particular partition is defined as the number of edges inside clusters, minus the expected number of such edges if the graph were random, conditioned on its degree distribution.

More formally, consider an undirected unweighted network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes ($\mathcal{V} = \{1, \ldots, n\}$), and m edges ($m = Card(\mathcal{E})$). Denote by A the adjacency matrix: $A_{ij} = 1,\ if\ (i, j) \in \mathcal{E}$, 0 otherwise. The degree of node i is denoted ω_i ($\omega_i = \sum_{j=1}^{n} A_{ij}$), and ω stands for the vector whose components are ω_i.

Let $\mathcal{P}$ be a partition of $\mathcal{V}$, and let $\delta(i, j)$ be a function that takes the value 1 if nodes i, j are in the same community and 0 otherwise. The modularity measure is defined as

$$Q(\mathcal{P}) = \frac{1}{2m} \sum_{i,j=1}^{n} \left(A_{ij} - \frac{\omega_i \omega_j}{2m} \right) \delta(i, j).$$

The first part of $Q(\mathcal{P})$, say, $\frac{1}{2m} \sum_{i,j=1}^{n} A_{ij} \delta(i, j)$, corresponds to the fraction of intracommunity links, whereas the second part $\frac{1}{2m} \sum_{i,j=1}^{n} \frac{\omega_i \omega_j}{2m} \delta(i, j)$ corresponds to the same fraction in a random network. The modularity values range from -0.5 to 1 (Brandes et al., 2008), with a higher positive value indicating stronger support for community structure in the network.

Hence, community discovery can be formulated as the so-called modularity maximization problem, which consists of finding a partition of nodes of a network with the highest modularity. Since its introduction, the modularity measure has become a central tool for network analysis, and a great deal of work has studied its properties (Danon, Duch, Diaz-Guilera, & Arenas, 2005; Fortunato & Barthelemy, 2007; Arenas, Fernandez, & Gomez, 2008; Brandes et al., 2008; Djidjev, 2008; Twan & Elena, 2013), and numerous algorithms have been developed to maximize modularity. Brandes et al. (2008) has proved that the modularity maximization is an NP-hard problem. Hence, exact methods are computationally intractable even for very small networks, and most existing methods are approximate or heuristic.

In terms of clustering, the approaches differ in whether a hierarchical partition (recursively subdividing communities into subcommunities) is sought and whether the number of communities is prespecified by the user or decided by the algorithm, as well as other parameters. In this sense, three classes of algorithms can be distinguished: divisive hierarchical clustering (tackle multiple two-partition problems repeatedly: Agarwal and Kempe, 2008; Duch & Arenas, 2005; Newman, 2006b), agglomerative hierarchical clustering (merge clusters repeatedly: Blondel, Guillaume, Lambiotte, & Lefebvre, 2008; Clauset, Newman, & Moore, 2004; Newman, 2004; Pons & Latapy, 2004; Wakita & Tsurumi, 2007), and k-way partitioning (Emprise & Dit-Yan, 2011; Guimera & Amaral, 2005; Lehmann & Hansen, 2007; Newman, 2006a; Boccaletti, Ivanchenko, Latora, Pluchino, & Rapisarda, 2007; Evans & Lambiotte, 2009; Mariá & Leonidas, 2013; Nadakuditi & Newman, 2012). On the other hand, White and Smyth (2005) have shown that modularity-based clustering can be understood as a special instance of spectral clustering. Methods based on spectral clustering have been developed by Newman (2006a), White and Smyth (2005), Richardson, Mucha, and Porter (2009), and Sun, Danila, Josic, and Bassler (2009).

In terms of optimization, one distinguishes three main approaches for modularity maximization: heuristics, metaheuristics, and mathematical programming approaches. Numerous heuristics have been developed by, among others, Blondel et al. (2008), Clauset et al. (2004), Duch and Arenas (2005), Emprise and Dit-Yan (2011), Newman (2004, 2006a, 2006b), Pons and Latapy (2004), and Cafieri, Hansen, and Liberti (2011). For metaheuristics, one can cite simulated annealing (Massen and Doye, 2005; Guimera & Amaral, 2005; Medus, Acuna, & Dorso, 2005), genetic search (Tasgin, Herdagdelen, & Bingol, 2007), greedy algorithms (Wakita & Tsurumi, 2007; Schuetz & Caflisch, 2008).

The mathematical programming approaches are more recent. In these approaches, modularity maximization is formulated as either an integer (or mixed integer) linear programming problem or an integer/mixed integer quadratic program. Due to the NP-hardness of these problems, most of proposed methods are approximate. Agarwal and Kempe (2008) considered an integer linear programming formulation and introduced two approximate

algorithms. The first is linear programming followed by randomized rounding, and the second is based on vector programming relaxation of a quadratic program that recursively splits one partition into two smaller partitions while a better modularity can be obtained. Emprise and Dit-Yan (2011) considered an integer quadratic formulation and proposed an iterative algorithm that solves a quadratic program at each iteration. There are few exact algorithms; the first was introduced by Xu, Tsoka, & Papageorgiou (2007), and the most recent is based on column generation approaches for mixed integer linear/quadratic programming (Aloise et al., 2010). The largest problem solved to date by exact methods has 512 entities (Aloise et al., 2010).

Recently, new modularity measures based on the modularity Q and the modularity-density D-measure (Li, Zhang, Wang, Zhang, & Chen 2008) and then new formulations of modularity maximization via mixed integer nonlinear programming were investigated in Jonathan and Lisa (2012) and Laura, Songsong, Lazaros, and Sophia (2012). These approaches aim to identify overlapping communities or small communities that cannot be revealed by the original modularity maximization problem.

In this letter, we focus on maximizing the original modularity Q by a mathematical programming approach via an integer quadratic programming problem. Our method is based on DC (difference of convex functions) programming and DCA (DC algorithms), an innovative continuous approach in a nonconvex programming framework. DC programming and DCA were introduced by Pham Dinh Tao in their preliminary form in 1985 and have been extensively developed since 1994 by Le Thi Hoai An and Pham Dinh Tao. They have become classic and are increasingly popular (Collobert, Sinz, Weston, & Bottou, 2006; Le Thi, Belghiti, & Pham Dinh, 2006; Le Thi, Le, & Pham Dinh, 2006, 2007; Le Thi, Le, Nguyen, & Pham Dinh, 2008; Le Thi, Nguyen, & Ouchani, 2008; Liu, Shen, & Doss, 2005; Liu & Shen, 2006; Neumann, Schnörr, & Steidl, 2004; Ong & Le Thi, 2013; Pham Dinh & Le Thi, 1997, 1998; Shen, Tseng, Zhang, & Wong, 2003; Weber, Schüle, & Schnörr, 2005; and the list of references in Le Thi, 2013). They address the DC programs of the form

$$\alpha = \inf\{f(x) := g(x) - h(x) : x \in \mathbb{R}^p\} \quad (P_{dc}) \tag{1.1}$$

where $g, h : \mathbb{R}^p \to \mathbb{R} \cup \{+\infty\}$ are lower semicontinuous proper convex functions on $\mathbb{R}^p$. Such a function f is called DC function, and $g-h$, DC decomposition of f while g and h are DC components of f. The construction of DCA involves DC components g and h but not the function f itself. Hence, for a DC program, each DC decomposition corresponds to a different version of DCA. Since a DC function f has an infinite number of DC decompositions, which have crucial impacts on the qualities (e.g., speed of convergence, robustness, efficiency, globality of computed solutions) of DCA, the search

for a good DC decomposition is important from an algorithmic point of view. Moreover, despite its local character, DCA with a good initial point can converge to global solutions. Finding a good initial point is then also an important stage of DCA. How to develop an efficient algorithm based on the generic DCA scheme for a practical problem is thus a judicious question to be studied, and the answer depends on the specific structure of the problem being considered.

Our work is motivated by the fact that DCA has been successfully applied to many (smooth or nonsmooth) large-scale nonconvex programs in various domains of applied sciences, in particular, in machine learning (Collobert et al., 2006; Le Thi, Belghiti et al., 2006; Le Thi, Le et al., 2006; Le Thi et al., 2007; Le Thi, Le et al., 2008; Le Thi, Nguyen et al., 2008; Liu et al., 2005; Liu & Shen, 2006; Neumann et al., 2004; Ong & Le Thi, 2013; Shen et al., 2003; Yuille & Rangarajan, 2002; Weber et al., 2005) for which they quite often provided a global solution and proved to be more robust and efficient than standard methods. Working on the matrix space, we first formulate the modularity maximization problem as maximizing a quadratic function under the Cartesian product of unit simplices with binary variables. This problem is then equivalently reformulated as maximizing a quadratic function under the same set but now with continuous variables on which DCA can be applied. We propose an appropriate DC decomposition that gives rise, after a suitable computational strategy, to a very simple DCA scheme (DCAM) in which all computations are explicit. The advantages of our algorithm are multiple:

- Thanks to the DC decomposition technique, the initial combinatorial optimization problem is transformed equivalently, in an elegant way, to a continuous problem. Surprisingly, due to this customized choice of DC decomposition, although our algorithm works on a continuous domain, it constructs a sequence in the discrete feasible set of the initial problem. Such an original property is important for large-scale settings: in ultra-large networks, if we stop the algorithm before its convergence, we get always an integer solution, that is, the algorithm furnishes an approximate solution without rounding procedures.
- Again thanks to DC decomposition, DCAM enjoys interesting convergence properties. It converges, after a finitely many iterations, to a local solution in almost cases. In particular, DCAM is very inexpensive in terms of CPU time since all computations are explicit. In fact, each iteration requires only one matrix-vector product and maximizes a linear function on a simplex whose solutions are explicitly computed (no solver is required).
- Although the number of clusters is an input parameter of the algorithm, it can be modified by DCAM itself during its iterations. Starting with a very large number of clusters (even with n, the number of nodes of the network), DCAM can detect empty clusters and

provide an optimal partition together with the optimal number of communities. That feature constitutes a great advantage of DCAM: finding the number of clusters is a difficult and still relevant issue for researchers in clustering analysis.

These benefits are confirmed by our numerical results on real-world networks: DCA outperforms standard approaches on solution quality, computation cost, and scalability.

The rest of the letter is organized as follows. In section 2, we present the integer quadratic formulation of the modularity maximization problem. Section 3 is devoted to DC programming and DCA for solving this quadratic program. First, we give a brief presentation of DC programming and DCA and then present reformulation techniques as well as a DC formulation of the considered problem. Afterward we show how to determine the resulting DCA scheme and provide some discussions about the algorithm. Computational experiments are reported in section 4; section 5 concludes the letter.

2 An Integer Quadratic Formulation of the Modularity Maximization Problem

Assume that each vertex (entity) belongs to exactly one community (i.e., the case of overlapping communities is not considered here).

Let $\mathcal{P}$ be a partition of $\mathcal{V}$ and c be the number of communities in $\mathcal{P}$. Define the binary assignment matrix $U = (U_{ik})_{i=1,\ldots,n}^{k=1,\ldots,c}$ in $\mathcal{P}$, say, $U_{ik} = 1$, if the vertex i belongs to community k and 0 otherwise. Then the modularity can also be expressed according to U as follows:

$$Q(U) = \frac{1}{2m} \sum_{i,j=1}^{n} B_{ij} \sum_{k=1}^{c} U_{ik} U_{jk},$$

where $B := A - \frac{1}{2m}\, \omega\omega^T$ is a constant matrix, called the modularity matrix. It depends on only the network (independent on $\mathcal{P}$). Hence, the modularity maximization problem can be written as

$$\max_{U} \quad Q(U) := \frac{1}{2m} \sum_{i,j=1}^{n} B_{ij} \sum_{k=1}^{c} U_{ik} U_{jk} \tag{2.1}$$

$$s.t \quad \sum_{k=1}^{c} U_{ik} = 1 \; \forall i = 1, \ldots, n, \; U_{ik} \in \{0,1\} \; \forall i = 1, \ldots, n; \forall k = 1, \ldots, c. \tag{2.2}$$

The constraint $\sum_{k=1}^{c} U_{ik} = 1$ ensures that each entity belongs to exactly one community. Observe that the maximizing modularity gives an optimal partition together with the optimal number of communities c.

For a matrix $U \in \mathbb{R}^{n.c}$, $U_{i.}$ and $U_{.j}$ denote the ith row and the jth column of U, respectively. The transpose of U is denoted by U^T, and $(U_i)^T$ will be written as U_i^T for simplicity.

Let $\mathcal{M}_{n,c}(\mathbb{R})$ denote the space of real matrices of order $n \times c$. We can identify by rows (resp. columns) each matrix $U \in \mathcal{M}_{n,c}(\mathbb{R})$ with a row vector (resp. column vector) in $(\mathbb{R}^c)^n$ (resp. $(\mathbb{R}^n)^c$) by writing, respectively,

$$U \longleftrightarrow \mathfrak{U} = (U_{1.}, \ldots, U_{n.}), U_{i.}^T \in \mathbb{R}^c,\ \mathfrak{U}^{\mathrm{T}} \in (\mathbb{R}^{\mathrm{c}})^{\mathrm{n}} \tag{2.3}$$

and

$$U \longleftrightarrow \mathcal{U} = \begin{pmatrix} U_{.1} \\ \vdots \\ U_{.c} \end{pmatrix},\ U_{.j} \in \mathbb{R}^n, \mathcal{U} \in (\mathbb{R}^n)^c. \tag{2.4}$$

The inner product in $\mathcal{M}_{n,c}(\mathbb{R})$ is defined as the inner product in $(\mathbb{R}^c)^n$ or $(\mathbb{R}^n)^c$, that is,

$$\langle X, Y \rangle_{\mathcal{M}_{n,c}(\mathbb{R})} = \langle \mathfrak{X}^T, \mathfrak{Y}^T \rangle_{(\mathbb{R}^c)^n} = \langle \mathcal{X}, \mathcal{Y} \rangle_{(\mathbb{R}^n)^c} = Tr(X^T Y),$$

where $Tr(X^T Y)$ denotes the trace of the matrix $X^T Y$. Hence the modularity measure can be computed as

$$Q(U) = \frac{1}{2m} Tr(U^T B U) = \frac{1}{2m} \sum_{j=1}^{c} (U_{.j})^T B U_{.j} = \frac{1}{2m} \sum_{i=1}^{n} U_{i.} ([(BU)]_{i.})^T. \tag{2.5}$$

For simplifying computations, we choose either representation, matrix or vector, of U in a convenient way. For instance, with the column identification of U, we can express the modularity measure using vector representation as $Q(\mathcal{U}) = \frac{1}{2m} \mathcal{U}^T B_b \mathcal{U}$, where B_b is the diagonal block matrix of order $n.c \times n.c$ whose blocks are all equal to B.

Let Δ_c be the $(c-1)-$simplex defined as

$$\Delta_c = \left\{ x \in [0,1]^c : \sum_{k=1}^{c} x_k = 1 \right\},$$

and let $\Delta \subset [0,1]^{n\times c}$ be the Cartesian product of n simplices Δ_c, say,

$$\Delta := \Delta_c \times, \ldots, \times \Delta_c.$$

Denote by $V(\Delta_c)$ (resp. $V(\Delta)$) the vertex set of Δ_c (resp. Δ). Obviously

$$V(\Delta) = V(\Delta_c) \times, \ldots, \times V(\Delta_c).$$

With the matrix representation, problem 2.3 is written as

$$\max\left\{\frac{1}{2m} Tr(U^T B U) : U_{i.} \in V(\Delta_c), \forall i = 1, \ldots, n\right\},$$

and with the vector column representation, it is expressed as

$$\max\left\{Q(\mathcal{U}) := \frac{1}{2m}\mathcal{U}^T B_b \mathcal{U} : \mathcal{U} \in V(\Delta)\right\}. \tag{2.6}$$

In the next section, we develop DC programming and DCA for solving the modularity maximization problem of the form 2.6.

3 Solving the Modularity Maximization Problem by DC Programming and DCA

3.1 A Brief Introduction of DC Programming and DCA. As indicated in section 1, DC programming and DCA address the problem of minimizing a DC function on the entire space $\mathbb{R}^p$ or on a closed convex set $C \in \mathbb{R}^p$. Generally, a DC program takes the form

$$\alpha = \inf\{f(x) := g(x) - h(x) : x \in \mathbb{R}^p\}, \quad (P_{dc}), \tag{3.1}$$

where $g, h : \mathbb{R}^p \to \mathbb{R} \cup \{+\infty\}$ are lower semicontinuous proper convex functions on the Euclidean space $\boldsymbol{X} := \mathbb{R}^p$. When either g or h is a polyhedral convex function (say, a pointwise supremum of a finite collection of affine functions) (P_{dc}) is called a polyhedral DC program.

A convex constraint $x \in C$ can be incorporated in the objective function f by using the indicator function on C, denoted χ_C, which is defined by $\chi_C(x) = 0$ if $x \in C$, $+\infty$ otherwise. (Any convex constrained DC program can be written in the standard form (P_{dc}) by adding the indicator function of the convex constraint set to the first DC component g.)

Let $y \in \boldsymbol{Y}$, where $\boldsymbol{Y}$ is the dual space of $\boldsymbol{X} = \mathbb{R}^p$ that can be identified with $\mathbb{R}^p$ itself. Let $g^*(y)$ defined by

$$g^*(y) = \sup\{\langle x, y\rangle - g(x) \ : \ x \in \mathbb{R}^p\}$$

be the conjugate function of g. Then the following program is called the dual program of (P_{dc}):

$$\alpha_D = \inf\{h^*(y) - g^*(y) : y \in \mathbf{Y}\}. \quad (D_{dc}).$$

One can prove that $\alpha = \alpha_D$ (Le Thi & Pham Dinh, 2005; Pham Dinh & Le Thi, 1997) and there is a perfect symmetry between primal and dual DC programs: the dual of (D_{dc}) is exactly (P_{dc}).

For a convex function h, the subdifferential of h at x_0, denoted by $\partial h(x_0)$, is defined by

$$\partial h(x_0) \equiv \{y \in \mathbb{R}^p : h(x) \geq h(x_0) + \langle x - x_0, y\rangle, \forall x \in \mathbb{R}^p\}.$$

The subdifferential $\partial h(x_0)$ is a closed convex set that generalizes the derivative in the sense that h is differentiable at x_0 if and only if $\partial h(x_0) \equiv \{\nabla_x h(x_0)\}$.

DCA is based on the local optimality conditions for (P_{dc}), namely,

$$\partial h(x^*) \cap \partial g(x^*) \neq \emptyset \tag{3.2}$$

(such a point x^* is called the critical point of $g - h$ or generalized KKT point for (P_{dc})), and

$$\emptyset \neq \partial h(x^*) \subset \partial g(x^*). \tag{3.3}$$

Condition 3.3 is a necessary local optimality for (P_{dc}). It is also sufficient for many classes of DC programs that are quite often encountered in practice. In particular it is sufficient for polyhedral DC programs.

The transportation of global solutions between (P_{dc}) and (D_{dc}) is expressed by the following properties:

Property 1.

$$[\cup_{y^* \in \mathcal{D}} \partial g^*(y^*)] \subset \mathcal{P}, \ [\cup_{x^* \in \mathcal{P}} \partial h(x^*)] \subset \mathcal{D}, \tag{3.4}$$

where $\mathcal{P}$ and $\mathcal{D}$ denote the solution sets of (P_{dc}) and (D_{dc}), respectively.

Under technical conditions, this transportation also holds for local solutions of (P_{dc}) and (D_{dc}) (see Le Thi & Pham Dinh, 2005; Pham Dinh & Le Thi, 1997 for more details).

Property 2. Let x^* be a local solution to (P_{dc}) and let $y^* \in \partial h(x^*)$. If g^* is differentiable at y^*, then y^* is a local solution to (D_{dc}). Similarly, let y^* be a local solution to (D_{dc}) and let $x^* \in \partial g^*(y^*)$. If h is differentiable at x^*, then x^* is a local solution to (P_{dc}).

The main idea of DCA is simple: each iteration of DCA approximates the convex function h by its affine minorant defined by $y^k \in \partial h(x^k)$ and solves the resulting convex program.

DCA: General Scheme

Initializations: let $x^0 \in \mathbb{R}^p$ be a guess, set $k := 0$.
Repeat
1. Calculate $y^k \in \partial h(x^k)$.
2. Calculate $x^{k+1} \in \arg\min\{g(x) - \langle x, y^k \rangle : x \in \mathbb{R}^p\} \quad (P_k)$.
3. $K = k + 1$.

Until convergence of $\{x^k\}$.

Convergence properties of DCA and its theoretical basis can be found in Le Thi and Pham Dinh (2005) and Pham Dinh and Le Thi (1997, 1998). For instance, it is important to mention that (for simplicity, we omit here the dual part of these properties):

- DCA is a descent method without line search: the sequence $\{g(x^k) - h(x^k)$ is decreasing. If $g(x^{k+1}) - h(x^{k+1}) = g(x^k) - h(x^k)$, then x^k is a critical point of $g - h$.
- If the optimal value $\alpha$of problem (P_{dc}) is finite and the infinite sequence $\{x^k\}$is bounded, then every limit point x^* of the sequence $\{x^k\}$ is a critical point of $g - h$. In such a case, DCA terminates at the kth iteration.
- DCA has a linear convergence for DC programs. Especially for polyhedral DC programs, the sequence $\{x^k\}$ contains finitely many elements and the algorithm converges after a finite number of iterations.

For a complete study of DC programming and DCA, see Le Thi and Pham Dinh (2005) and Pham Dinh and Le Thi (1997, 1998) and the references therein.

The solution of a nonconvex program (P_{dc}) by DCA must be composed of two stages: the search for an appropriate DC decomposition of f and that of a good initial point. Research has been very active on the use of DC programming and DCA for machine learning and data mining.

It should be noted that:

- The convex concave procedure (CCCP) for constructing discrete time dynamical systems mentioned in Yuille and Rangarajan (2002) is a special case of DCA applied to smooth optimization.
- The SLA (successive linear approximation) algorithm developed by Bradley and Mangasarian (1998) is a version of DCA for concave minimization;
- The EM algorithm that Dempster, Laird, and Rubin (1977) applied to the log-linear model is a special case of DCA.

We show below how to use DCA for solving the equivalent modularity maximization problem, equation 2.6.

3.2 DC Programming and DCA for Solving Problem 2.6. Both column and row identifications displayed in section 2 will be used here in order to provide an explicit DCA that could handle large-size problems.

3.2.1 A Continuous Reformulation of Problem 2.6. Using a DC decomposition technique, we first reformulate problem 2.6 as a continuous optimization problem.

For any real number μ, the modularity measure can be rewritten as

$$Q(\mathcal{U}) = \frac{1}{2m}\mathcal{U}^T(B_b + \mu I_b)\mathcal{U} - \frac{1}{2m}\mu\mathcal{U}^T\mathcal{U},$$

where I_b is the identity matrix of order $n.c$.

Let h be the quadratic function defined by $h(\mathcal{U}) := \frac{1}{2}\mathcal{U}^T(B_b + \mu I_b)\mathcal{U}$. As $\mathcal{U} \in V(\Delta)$, we have

$$\mathcal{U}^T\mathcal{U} = Tr(U^TU) = \sum_{i=1}^{n} U_{i.}(U_{i.})^T = n.$$

Therefore, problem 2.6 is equivalent to

$$\max\left\{h(\mathcal{U}) : =\frac{1}{2}\mathcal{U}^T(B_b + \mu I_b)\mathcal{U} : \mathcal{U} \in V(\Delta)\right\}. \tag{3.5}$$

Let μ be a scalar such that $\mu > -\lambda_1(B)$, where $\lambda_1(B)$ is the smallest eigenvalue of the modularity matrix B. Hence, the quadratic function h is strongly convex and we call μ the regularization parameter. As h is strongly convex, problem 3.5 and the convex maximization problem,

$$\max\left\{h(\mathcal{U}) : \mathcal{U} \in \Delta\right\}, \tag{3.6}$$

are equivalent in the sense that they have the same optimal value and the same solution set.

3.2.2 A DC Formulation of Problem 3.6. In the sequel, we consider problem 3.6 with $\mu > -\lambda_1(B)$. This convex maximization problem can be written as

$$\min\left\{-h(\mathcal{U}) : \mathcal{U} \in \Delta\right\},$$

which is equivalent to

$$\min\left\{f(\mathcal{U}) := \chi_\Delta(\mathcal{U}) - h(\mathcal{U}) : \mathcal{U} \in \mathbb{R}^{n.c}\right\}, \tag{3.7}$$

where χ_Δ is the indicator function on Δ. Clearly, the function χ_Δ is convex (because that Δ is a convex set), and then problem 3.7 is a DC program with the following natural DC decomposition:

$$f(\mathcal{U}) := g(\mathcal{U}) - h(\mathcal{U}),\ g(\mathcal{U}) = \chi_\Delta(\mathcal{U}).$$

Since χ_Δ is a polyhedral function, problem 3.7 is a polyhedral DC program. Moreover, h is differentiable, and its gradient is given by $\nabla h(\mathcal{U}) = (B_b + \mu I_b)\mathcal{U}$.

3.2.3 The DCA Scheme Corresponding to Problem 3.7. According to the generic DCA scheme, applying DCA to problem 3.7 consists of computing, at each iteration k, $\mathcal{Y}^k = \nabla h(\mathcal{U}^k) = (B_b + \mu I_b)\mathcal{U}^k$ and then solving the next problem to obtain $\mathcal{U}^{k+1}$:

$$\min\{\chi_\Delta(\mathcal{U}) - \langle \mathcal{U}, \mathcal{Y}^k\rangle : \mathcal{U} \in \mathbb{R}^{n.c}\}). \tag{3.8}$$

Denote by $Y^k = (B + \mu I)U^k$ (I denotes the identity matrix of order n). Taking into account the useful properties concerning matrix representation of $\mathcal{U}$ and $\mathcal{Y}^k$, we can compute explicitly an optimal solution of problem 3.8 as follows:

$$\begin{aligned}\min\{\chi_\Delta(\mathcal{U}) - \langle \mathcal{U}, \mathcal{Y}^k\rangle : \mathcal{U} \in \mathbb{R}^{n.c}\} &\Longleftrightarrow \max\{\langle \mathcal{U}, \mathcal{Y}^k\rangle : \mathcal{U} \in \Delta\} \\ = \max_{U_{i.} \in \Delta_c, \forall i=1,\dots n} \left\{\sum_{i=1}^{n} \langle U_{i.}, Y_{i.}^k\rangle\right\} &= \max_{U_{i.} \in V(\Delta_c), \forall i=1,\dots n} \left\{\sum_{i=1}^{n} \langle U_{i.}, Y_{i.}^k\rangle\right\}.\end{aligned} \tag{3.9}$$

The last problem is separable, and solving it amounts to solving n problems of the form

$$\max\{\langle U_{i.}, Y_{i.}^k\rangle : U_{i.} \in V(\Delta_c)\}$$

whose optimal solution is given by

$$U_{i.}^{k+1} = e_{\arg\max_{j=1\dots c} Y_{ij}^k}, \tag{3.10}$$

where $\{e_j : j = 1, \dots, c\}$ is the canonical basis of $\mathbb{R}^c$.

The sequence $\{\mathcal{U}^k\}$ computed by DCA has the following interesting property which shows the efficiency of DCA in the search of the optimal number of clusters.

Proposition 1. *If at iteration k one has $U_{il}^k = 0\ \forall i = 1, \dots, n$, then $U_{il}^{k+1} = 0\ \forall i = 1, \dots, n$. Consequently, if a cluster is empty at an iteration k, then it can be deleted in DCA. The sequence $\{c^k\}$ (c^k stands for the number of nonempty*

clusters at the iteration k) is decreasing during DCA's iterations and reduced to c^* at iteration k_*, say $c^0 \geq c^1 \geq \ldots \geq c^k \geq c^{k+1} \geq \ldots \geq c^{k_*} = c^*$ and $c^k = c^*$ for all $k > k_*$.

Proof. If there exists $l \in \{1, \ldots, c\}$ such that $U_{il}^k = 0 \, \forall i = 1, \ldots, n$, then $Y_{il}^k = (B + \mu I)_{i.} U_{.l}^k = 0 \, \forall i = 1, \ldots, n$. Now let $j_* := \arg\max_j Y_{ij}^k$. We will prove that $j_* \neq l$. Indeed, we have (e is the vector of ones in $\mathbb{R}^n$)

$$\sum_{j=1}^{c} Y_{ij}^k = \sum_{j=1}^{c} (B + \mu I)_{i.} U_{.j}^k = (B + \mu I)_{i.} \sum_{j=1}^{c} U_{.j}^k = (B + \mu I)_{i.} e^T$$

$$= \left(\sum_{t=1}^{n} B_{it} \right) + \mu = \mu.$$

Hence $Y_{i.}^k$ contains at least one positive element and therefore $Y_{ij_*}^k := \max_{j=1\ldots c} Y_{ij}^k > 0$. That means $j_* \neq l$. From equation 3.10, it follows that $U_{il}^{k+1} = 0$.

Proposition 1 allows us to update the number of clusters during DCA's iterations. Starting with c^0, a very large number of clusters (the number of nodes in the network), DCA automatically detects empty clusters. Then by removing all columns l such that $U_{.l}^k = 0$, we reduce considerably (during some first iterations) the dimension of matrices Y^k and U^k.

The proposed DCA including the update of clusters number can be described as follows:

DCAM

Initialization: Let U^0 be a $n \times c$ matrix with binary values, let c^0 be an integer number sufficiently large.
Let ε and ϵ be small positive values. Calculate $\mu = -\lambda_{\min}(B) + \varepsilon$. $k \leftarrow 0$. Set $c \leftarrow c^0$.
Repeat

1. Compute $Y^k = (B + \mu I)U^k$.
2. Set $U_{i.}^{k+1} = e_{\arg\max_{j=1,\ldots,c^k} Y_{ij}^k}$, $\forall i \in \{1, \ldots, n\}$.
3. For $l = 1, \ldots, c$
   if $U_{.l}^{k+1} = 0$ then update $c \leftarrow c - 1$, remove the column $U_{.l}^{k+1}$ from U^{k+1}
   end for
   $k \leftarrow k + 1$,

until $\frac{\|U^{k+1} - U^k\|}{\|U^k\|} < \epsilon$.

Theorem 1 *(convergence properties of DCAM).*

i. *DCAM generates the sequence* $\{\mathcal{U}^k\}$ *contained in* $V(\Delta)$ *such that the sequence* $\{-Q(\mathcal{U}^k)\}$ *is decreasing (or, equivalently, the sequence* $\{Q(\mathcal{U}^k)\}$ *is increasing).*

ii. *Sequence* $\{\mathcal{U}^k\}$ *converges to* $\mathcal{U}^* \in V(\Delta)$ *after a finite number of iterations.*

iii. *Point* $\mathcal{U}^*$ *is a KKT point of problem 3.7. Moreover, if*

$$\arg\max_j (B + \mu I)U^*_{ij} \quad \text{is a singleton } \forall i \in \{1, \ldots, n\}, \tag{3.11}$$

then $\mathcal{U}^*$ *is a local solution to problem 3.7.*

Proof. i and ii are direct consequences of the convergence properties of DCA for a polyhedral DC program. Moreover, since h is differentiable everywhere, condition $\partial h(\mathcal{U}^*) \cap \partial g(\mathcal{U}^*) \neq \emptyset$ becomes $\partial h(\mathcal{U}^*) \subset \partial g(\mathcal{U}^*)$ which is the KKT condition for problem 3.7. Hence, the first part of iii is straightforward. Only the second part of iii needs a proof.

We first note that $g := \chi_\Delta$ is a polyhedral convex function; so is its conjugate $g^* := \chi^*_\Delta$. Hence, the dual DC program of problem 3.7 is a polyhedral DC program and the relation

$$\emptyset \neq \partial g^*(\mathcal{Y}^*) \subset \partial h^*(\mathcal{Y}^*) \tag{3.12}$$

is a necessary and sufficient local optimality condition (Le Thi & Pham Dinh, 2005; Pham Dinh & Le Thi, 1997, 1998). Clearly condition 3.11 is verified if and only if the problem

$$\min\{\chi_\Delta(\mathcal{U}) - \langle \mathcal{U}, \mathcal{Y}^* \rangle : \mathcal{U} \in \mathbb{R}^{n.c}\} \tag{3.13}$$

admits a unique optimal solution, that is, g^* is differentiable at $\mathcal{Y}^* = (B_b + \mu I_b)\mathcal{U}^*$. In other words, by returning to matrix representation with $Y^* = (B + \mu I)U^*$, problem 3.13 can be expressed in the form of problem 3.8, and the uniqueness of solutions to problem 3.13 holds iff, for $i = 1, \ldots, n$, the row $[(B + \mu I)U^*]_{i.}$ has a unique greatest entry. In this case, relation 3.12 holds, and therefore, $\mathcal{Y}^*$ is a local solution to the dual DC program of equation 3.7. Using the transportation of local minimizers (see the two properties mentioned in section 3.1) between primal and dual DC programs, we conclude that $\mathcal{U}^*$ is a local solution to problem 3.7. Thus ends the proof.

3.3 Discussion. The main advantages of our algorithms were noted in section 1. Here we provide a more detailed discussion of the complexity of the algorithm and the choice of c^0.

DCAM algorithm is simple and easy to implement. One iteration of the algorithm relies only on very few basic operations, which leads to a very low computation cost (all computations are explicit and then no solver is used for the convex subproblem in DCA). Moreover, there is no need to

evaluate the objective function numerous times as in standard gradient descent schemes, for instance.

An important point contributing to the efficiency of DCAM comes from the fact that even if we reformulate the initial combinatorial problem as a continuous problem, 3.6, DCAM works only on the vertex set of the simplex Δ, which is finite. Therefore, contrary to other relaxation approaches in which cluster assignments are represented as a full matrix (see, e.g., the deterministic annealing method of Lehmann and Hansen, 2007), in the DCAM algorithm, the assignment matrix is a binary matrix all along the iterations. Matrix operations can therefore be performed efficiently. The total number of operations to solve this main linear program is nc. Hence, the key operations in DCAM consist of computing the gradient matrix $Y^k := (B + \mu I)U^k = (A + \mu I)U^k - \frac{1}{2m}\omega(\omega^T U^k)$, which requires $O(m + nc)$ operations ($O(m)$ and $O(nc)$ for computing, respectively, the first and the second term). It should be noticed that Y^k can be computed incrementally as $Y^{k+1} = (B + \mu I)U^{k+1} = Y^k + (B + \mu I)(U^{k+1} - U^k)$. Observe that each row of $(U^{k+1} - U^k)$ corresponds to a change in the assignment that contains only two non-null components: -1 for the previous community and $+1$ for the new community; the other rows are null. Then the incremental computation of Y^{k+1} requires roughly $2\min(m, nq) + 2nc$ sums and products in the worst case, where q is the number of changes in the assignments when passing from k to $k + 1$. So if $q < \frac{m}{2n}$, the incremental approach is more efficient than the full calculation. If m takes a high value, the incremental approach will be used often and the running time will be shorter. In practice, the incremental approach can reduce running time on large networks by 15%.

The computation of $\lambda_1(B)$ can be done by the shifted power method in $O((n + m)n)$, assuming that the power method requires n iterations to converge (see Newman, 2006b). However DCA does not require an exact computation of $\lambda_1(B)$, only an approximation of this eigenvalue. Therefore, in practical situations, the number of iterations is much smaller than n.

Finally, it is worth noting once again the great advantage of DCAM in finding automatically the optimal number of clusters: starting with a very large value c^0 (the number of communities), DCAM furnishes as output results an optimal partition, together with the optimal number of communities c^*. As for the input value c^0, to avoid an underestimation of c^*, we start with a very large value c^0—for example, $c^0 = n$. Obviously, for large networks, the smaller c^0 is, the shorter CPU time would be, and the choice of $c^0 = n$ in these networks is not reasonable. Hence, a suitable value of c^0 should be chosen to avoid too many unnecessary calculations during the first iterations. It is well known in practice that the number of clusters of a network having n nodes is around $\sqrt{\frac{n}{2}}$ (Mardia, Kent, & Bibby, 1979). Hence we can take $c^0 \in [\sqrt{\frac{n}{2}}, n]$, for example, $c^0 = n$, in small and medium-size networks ($n \leq 500\,000$) and the larger n is, c^0 should be closer to $\sqrt{\frac{n}{2}}$.

In our experiments, we observe that the sequence $\{c^k\}$ is decreasing (quickly during some first iterations of DCAM) and reduced to c^* after the first half of DCA's iterations; during the remaining iterations, DCAM continues to optimize the modularity with this value c^*. So by removing empty clusters and updating the value of c during DCAM's iterations, we reduce considerably the complexity of DCAM.

3.4 Finding Good Starting Points for DCAM. Finding good starting points is one of important tasks in the design of DCA-based algorithms. Especially due to some issues of the modularity measure, this task should be carefully studied for modularity maximization problems. In fact, modularity clustering is known to have some issues, as has been discussed in Fortunato (2010) and (Good, de Montjoye, and Clauset, 2010):

1. The optimal partition may not coincide with the most intuitive partition (the resolution limit problem). Thus, merging clusters to get modular structures with higher modularity (as the trends of several heuristics, for instance, multilevel methods) may not produce an intuitive partition. That is why, in these approaches, small communities are often hidden.
2. There are typically an exponential number of structurally diverse alternative partitions with modularities very close to the optimum (the degeneracy problem). This implies that the results of deterministic algorithms that return a unique partition for a given network should be treated with particular caution, since this behavior tends to obscure the magnitude of the degeneracy problem and the wide range of alternative solutions.
3. The maximum modularity score depends on the size n of the network and the number of modules c it contains.

DCAM seems to be suitable for issue 3 in the sense that it finds simultaneously the optimal number c^* of modules and the optimal modularity. Issue 2 motivates us to investigate two efficient techniques to initialize DCAM in order to escape local minima. The first technique is inspired by the label propagation algorithm (LPA) proposed in Raghavan, Albert, and Kumara (2007). Furthermore, taking into account issue 1, we propose a DCAM-like algorithm that slightly modifies DCAM in the step of computing U^k (assign nodes to clusters) so that the clusters having a low density of intraconnections (among them small clusters) are favorable in the assignment. These two procedures to compute starting points are described below.

3.4.1 A Two-Step Procedure. In the first step, each node in the network is assigned a random numeric label that belongs to $\{1, 2, \ldots, c\}$. In the second step, the label of every node is updated by the new label corresponding to the most frequent in its neighborhood. This step is performed iteratively with a small number of iterations (two iterations in our experiment).

3.4.2 A DCAM-like Algorithm. Perform DCAM in which step 2 is replaced by

$$U_{i.}^{k+1} = e_{\arg\max_{j=1,\ldots,c^k} \frac{Y_{ij}^k}{deg(j)}}, \forall i \in \{1, \ldots, n\}, \tag{3.14}$$

where $deg(j)$ denotes the number of edges connecting two vertices u, v such that both u and v belong to cluster j.

Clearly this procedure gives priority to clusters having a low density of intraconnections in the assignment step. It seems to be able to circumvent the most important limit of modularity measure—small communities cannot be detected by modularity maximization. Hence performing some iterations of a DCAM-like the algorithm may give a good solution for starting the main DCAM algorithm.

4 Numerical Experiments

We tested DCAM on several real networks and compared it with six existing approaches. The experiments have been conducted on an Intel Core i7-2720QM CPU 2 × 2.20 Ghz with 4 Gb of memory.

4.1 Data Sets and Reference Algorithms. To evaluate the performance of the algorithms, we conduct experiments on 15 well-known networks that have been tested in previously (e.g., Newman, 2006b). The data sets are presented in Table 1 (note that the optimization problems have nc variables and $nc + n$ constrains). The data sets come from different sources:

- Zachary's Karate Club, Dolphin Social Network, Les Miserables, Books about US politics, and American College football are from Mark Newman's network data repository at http://www-personal.umich.edu/∼mejn/netdata/
- Jazz Musicians Network, Email communication network, PF2177, GearBox Tmt/sym, Hook/1498, Af/shell10, Rgg_n_2_21_s0, and Rgg_n_2_22_s0 are downloaded from the University of Florida Sparse Matrix Collection at http://www.cise.ufl.edu/research/sparse/matrices/
- Email Enron is at http://snap.stanford.edu/data/email-Enron.html;

The DCAM is compared to eight reference algorithms. The first six are heuristic: CNM developed in Clauset et al. (2004); FG, the improvement implementation version of CNM (Wakita & Tsurumi, 2007); WT, the Walktrap algorithm based on a new distance between nodes using random walks (Pons & Latapy, 2004); SP, the Spectral Bisection algorithm Newman (2006b); FU, the fast unfolding algorithm Blondel et al. (2008); (the fastest heuristic algorithm); and ML, multi level clustering (Noack & Rotta, 2009). The last two algorithms (Agarwal & Kempe, 2008) are based on mathematical

Table 1: Description of Data Sets.

Name	Note	Number of Vertices	Number of Edges
Zachary's Karate Club	KAR	34	78
Dolphin Social Network	DOL	62	159
Les Miserables	MIS	77	254
Books about US Politics	BOK	105	441
American College Football	BAL	115	613
Jazz Musicians Network	JAZ	198	2,742
Email Communication Network	EMA	1,133	5,451
Pf2177	PF2	9,728	367,436
Email_Enron	EER	36,691	183,830
GearBox	GEB	153,746	4,617,075
Tmt/sym	TMT	726,713	2,903,837
Hook/1498	HOK	1,498,023	31,207,734
Af/shell10	AFS	1,508,065	27,090,195
Rgg_n_2_21_s0	RG1	2,097,152	14,487,995
Rgg_n_2_22_s0	RG2	4,194,304	30,359,198

programming: LP (linear programming followed by randomized rounding) and VP (a vector programming relaxation of a quadratic program that recursively splits one partition into two smaller partitions while a better modularity can be obtained). Following are the sources of these algorithms:

- SP algorithm: http://deim.urv.cat/~sgomez/radatools.php)
- CNM: http://cs.unm.edu/ aaron/research/fastmodularity.htm
- FG and WT (Igraph library on C^{++}): http://igraph.sourceforge.net/
- ML: http://studiy.tu-cottbus.de/~rrotta/news:2010:06_02_new_release_of_the_multi-level_clustering_software
- FU: https://sites.google.com/site/findcommunities/
- VP and LP (Agarwal's web page): http://www-scf.usc.edu/~gaurava/

ML and FU are the two state-of-the-art methods for modularity maximization. In the ML algorithm, the communities detection has two phases: the coarsening phase with a merge prioritizer and the refinement phase. We use the single-step greedy coarsening method and multilevel fast greedy refinement (with reduction factors of 50%) at the suggestion of the authors (Noack & Rotta, 2009). For the merge prioritizer, the criteria for merging cluster pairs are based on modularity increasing ΔQ.

The fast unfolding algorithm has two phases. The first phase provides a local maxima of the modularity by performing a coarsening method whose merge prioritizer is the gain of modularity ΔQ. The second phase constructs a new network whose nodes are now the communities found during the first phase. The two phases are performed iteratively until there are no more changes (see Blondel et al., 2008, for details).

4.2 Experiment Setting. Our experiments have three parts. In the first experiment, we compare the behavior of DCAM with different starting points. In the second, we study the influence of the value c^0 on the performance of DCAM. In the third, we compare DCAM with reference algorithms.

For the first experiment, we consider three versions of DCAM according to the choice of initial points. The first, DCAM1, starts with a random initial point with values in $\{0, 1\}$. The second, DCAM2, uses the two-step procedure to initialize DCAM. The third, denoted DCAM3, starts with the two-step procedure and then performs T iterations of the DCAM-like algorithm ($T = 15$ in our experiment) to generate a starting point for DCAM.

Since the initial point is randomly computed (even if DCAM2 and DCAM3 use a particular procedure, in the first step, the labels of points are randomly chosen), we run each DCAM five times and take the best solution (the one corresponding to the largest modularity value). CPU time is computed as the total CPU of five runs, including the time for finding the initial point. As for the input value c^0, at the first run we take $c^0 = n$ except for the TMT, HOK, and AFS (resp. RG1, RG2) data sets and set $c^0 = 5.\sqrt{\frac{n}{2}}$ (resp. $c^0 = \sqrt{\frac{n}{2}}$). Let c^{*1} be the number of optimal clusters given by the first run of DCAM. Since the solution given by DCAM is suboptimal, c^{*1} is close to the real optimal number of clusters. Hence in the remaining four runs, we start DCAM with a larger value c^0, say $c^0 := 2c^{*1}$.

In the stop condition of DCAM, $\varepsilon = 10^{-6}$ for the first 10 data sets and $\varepsilon = 10^{-9}$ for the last 5 data sets.

FU is a stochastic/randomized algorithm; thus, we run it 5 times with a random order of nodes in the coarsening phase and take the best solution. As in DCAM, the FU CPU time is the total CPU of five runs.

4.3 Numerical Results. For experiment 1, We report in Table 2 the modularity (Q), the number of communities (c^*), and the CPU time of three versions of DCAM. For Experiment 2, we run DCAM3 five times for each of three values of c^0: n, $\frac{n}{2}$, and $5.\sqrt{\frac{n}{2}}$. The results are reported in Table 3, where c^* and Q are the best value given by five runs and CPU time is the total CPU time of five runs. For experiment 3, we compare DCAM3 with eight reference algorithms. The modularity values (Q) given by each algorithm, as well as the corresponding number of communities (c^*), are reported in Table 4. More clearly, Figure 1 presents the comparative modularity results of the algorithms. We set the limit CPU time equal to 1 hour for each run; the symbol - means that the algorithms do not converge after 1 hour.

The CPU times in seconds of all algorithms are given in Table 5.

4.3.1 Comparison of the Three Versions of DCAM. Thanks to the initial procedure, DCAM2 and DCAM3 are better than DCAM1 (with random

Table 2: Comparison Between the Tree Versions of DCAM.

	DCAM1			DCAM2			DCAM3		
Data Set	c*	Q	CPU	c*	Q	CPU	c*	Q	CPU
KAR	4	0.420	0.000	4	0.420	0.000	4	0.420	0
DOL	5	**0.529**	0.000	4	0.528	0.000	4	0.528	0.000
MIS	6	0.549	0.000	6	**0.560**	0.000	6	**0.560**	0.000
BOK	5	0.526	0.000	5	**0.527**	0.000	5	**0.527**	0.000
BAL	10	0.605	0.000	10	0.605	0.000	10	0.605	0.000
JAZ	4	0.445	0.000	4	0.445	0.000	4	0.445	0.000
EMA	39	0.535	0.172	12	**0.551**	0.000	12	**0.551**	0.000
PF2	19	0.546	2.532	10	**0.558**	0.782	10	**0.558**	2.028
EER	1282	0.521	194.05	1072	**0.579**	36.72	1072	**0.579**	37.551
GEB	431	0.735	215.37	142	0.841	80.11	75	**0.898**	42.718
TMT	3409	0.618	705.43	136	**0.829**	167.15	152	0.827	175.972
HOK	2433	0.594	5597.21	16	0.854	102.43	18	**0.895**	113.47
AFS	1736	0.627	3765.23	8	0.738	81.79	22	**0.891**	125.63
RG1	1954	0.758	4895.21	277	0.839	880.96	314	**0.957**	985.56
RG2	1448	0.755	5800.30	350	0.827	1966.2	525	**0.934**	1981.42

Note: Numbers in bold correspond to the best results.

Table 3: Results of DCAM3 with Different Values of c^0.

	$c^0 = n$			$c^0 = \frac{n}{2}$			$c^0 = 5\sqrt{\frac{n}{2}}$		
Data Set	c*	Q	CPU	c*	Q	CPU	c*	Q	CPU
KAR	4	0.420	0.000	4	0.420	0.000	4	0.420	0.000
DOL	4	0.528	0.000	5	0.529	0.000	4	0.527	0.000
MIS	6	0.560	0.000	6	0.555	0.000	6	0.555	0.000
BOK	5	0.527	0.000	5	0.527	0.000	5	0.527	0.000
BAL	10	0.605	0.000	10	0.605	0.000	9	0.598	0.000
JAZ	4	0.445	0.000	4	0.445	0.000	4	0.445	0.000
EMA	12	0.551	0.000	12	0.551	0.000	10	0.538	0.000
PF2	10	0.558	0.485	10	0.558	0.325	10	0.558	0.314
EER	1072	0.579	37.551	1059	0.567	35.60	734	0.561	26.05
GEB	75	0.898	42.718	103	0.897	35.56	102	0.889	28.28

initial points) on both quality and rapidity, in particular for large networks (networks with more than 1 million nodes). The gain ratio on running time of DCAM2 (resp. DCAM3) is up to 55 times (resp. 49 times) (HOK data). Meanwhile, even with a random starting point, DCAM1 is an efficient algorithm: it can handle large-scale problems in a reasonable time.

Comparing DCAM2 and DCAM3, we observe that DCAM3 is better in terms of modularity, especially for large networks. For the first 11 data sets (small and medium-size networks), the modularity values given by DCAM2

Table 4: Comparative Results (Number of Communities and Modularity Value) of Nine Algorithms.

	DCAM3		CNM		WT		SP		FG		ML		FU		VP		LP		
Data Set	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	c*	Q	UB
KAR	4	**0.420**	3	0.381	3	0.394	4	0.393	3	0.381	4	0.395	4	0.419	3	**0.420**	4	**0.420**	0.420
DOL	4	0.528	4	0.495	5	0.501	5	0.491	4	0.496	4	0.495	5	0.527	3	0.526	5	**0.529**	0.531
MIS	6	**0.560**	5	0.501	8	0.521	8	0.521	5	0.501	5	0.501	6	0.558	4	**0.560**	6	**0.560**	0.561
BOK	5	**0.527**	4	0.502	4	0.515	4	0.467	4	0.502	4	0.502	5	**0.527**	3	**0.527**	5	**0.527**	0.528
BAL	10	**0.605**	7	0.577	10	0.603	8	0.477	6	0.550	7	0.567	10	0.605	5	**0.605**	10	**0.605**	0.606
JAZ	4	**0.445**	4	0.439	10	0.433	3	0.394	4	0.439	4	0.439	7	0.444	3	**0.445**	8	**0.445**	0.446
EMA	12	0.551	13	0.513	44	0.525	22	0.493	17	0.501	13	0.511	12	0.548	4	**0.579**	5	0.526	-
PF2	10	**0.558**	10	0.545	20	0.454	17	0.431	10	0.545	10	0.545	10	0.556	–	–	–	–	–
EER	1072	0.579	1637	0.510	–	–	2631	0.409	1605	0.517	1045	0.525	983	**0.611**	–	–	–	–	–
GEB	75	0.898	17	0.831	–	–	–	–	19	0.831	17	0.831	29	**0.908**	–	–	–	–	–
TMT	152	0.827	9	0.797	–	–	–	–	10	0.798	9	0.797	120	**0.972**	–	–	–	–	–
HOK	18	**0.895**	–	–	–	–	–	–	–	–	–	–	27	0.892	–	–	–	–	–
AFS	22	**0.891**	–	–	–	–	–	–	–	–	–	–	33	0.886	–	–	–	–	–
RG1	314	0.957	–	–	–	–	–	–	–	–	–	–	21	**0.983**	–	–	–	–	–
RG2	525	0.947	–	–	–	–	–	–	–	–	–	–	230	**0.989**	–	–	–	–	–

Note: Numbers in bold correspond to the best results.

Table 5: CPU Time in Seconds of Each Algorithm.

Data	DCAM3	CNM	WT	SP	FG	ML	FU	VP	LP
KAR	**0.000**	0.116	**0.000**	0.015	**0.000**	**0.000**	0.07	18.51	0.69
DOL	**0.000**	0.223	**0.000**	0.028	**0.000**	**0.000**	0.04	19.29	4.53
MIS	**0.000**	0.551	0.015	0.038	**0.000**	0.04	0.07	26.13	3.98
BOK	**0.000**	0.367	0.048	0.066	0.015	0.21	0.10	21.14	18.45
BAL	**0.000**	0.394	0.000	0.096	0.000	0.27	0.10	34.95	12.79
JAZ	**0.000**	0.747	0.031	0.228	0.016	0.24	0.05	27.14	2.52
EMA	**0.000**	4.371	0.280	3.148	0.078	1.26	0.22	386.82	37.82
PF2	**2.028**	76.349	42.619	451.11	59.748	155.64	10.37	–	–
EER	37.551	231.355	–	2686.1	50.779	153.52	**4.21**	–	–
GEB	**42.718**	2372.0	–	–	179.93	1329.16	72.40	–	–
TMT	175.972	4560.37	–	–	188.57	1210.88	**113.35**	–	–
HOK	**113.47**	–	–	–	–	–	411.46	–	–
AFS	**125.63**	–	–	–	–	–	131.85	–	–
RG1	985.56	–	–	–	–	–	**421.93**	–	–
RG2	1981.42	–	–	–	–	–	**482.05**	–	–

Note: Numbers in bold correspond to the best results.

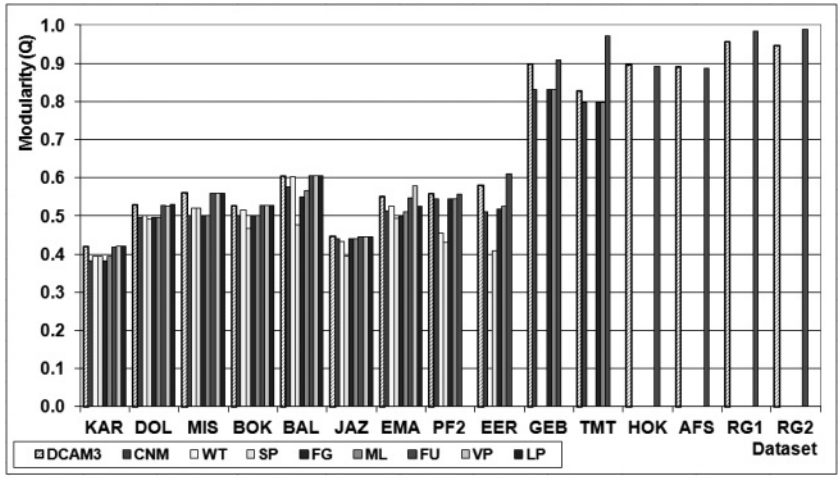

Figure 1: Comparative Modularity Values of nine Algorithms.

and DCAM3 are comparable: they are equal on 9 of 11 data sets and quite close on the 2 remaining data sets. For the last 4 data sets (large networks), the modularity values furnished by DCAM3 are much higher. These results confirm that the concept of the DCAM-like algorithm is interesting: it may be able to circumvent the limit of the modularity measure.

In terms of rapidity, DCAM3 needs more CPU time than DCAM2 in large networks. Meanwhile the CPU times of the two algorithms are quite close.

4.3.2 The Influence of c^0 on DCAM. DCAM is robust: c^* is quite stable when c^0 varies (except for EER data), and the difference of modularity values varies from 0.0 to 0.018. The results of the EER data show that DCAM does not have the same trend as several heuristics, for instance, multilevel methods—namely, merge clusters to get modular structures with higher modularity, and so the smaller the number of communities is, the larger the modularity is. As for CPU time, not surprisingly, the smaller c^0 is, the shorter the running time will be.

4.3.3 Comparison Between DCAM3 and the Six Heuristic Algorithms. In comparing the first four algorithms, CNM, SP, WT, and FG, DCAM3 always gives the largest modularity value and the shortest running time. The total CPU time of DCAM3 is much smaller than that of the four reference algorithms. On the four medium-size networks—FP2, EER, GEB, TMT ($9000 < n < 1,000,000$)—the gain ratio of DCAM varies, respectively, from 6.2 to 157.4 and from 1.1 to 123.2 in comparison with CNM and FG; WP (resp. SP) can solve only one (resp. two) problem(s) and the gain ratio of DCAM is 87.9 (resp. 71.5 and 930.1) times;

None of these first four algorithms can handle large networks, while DCAM3 is scalable: it works well on all large networks.

In comparing the two state-of-the-art methods ML and FU, the modularity values given by DCAM3 are better than that of ML (resp. FU) on 15 of 15 (resp. 8 of 15) data sets. FU gives higher (resp. the same) modularity values than DCAM3 on 5 (resp. 2) datasets. Note that ML does not converge after 1 hour on 4 large networks.

In terms of rapidity, DCAM3 is always faster than ML (the gain ratio is up to 76.7 times). On the seven medium and large data sets, DCAM3 is faster than FU on 4 data sets, while FU is faster on 3 data sets.

4.3.4 Comparison Between DCAM and the Two Mathematical Programming-based Algorithms VP and LP. VP and LP work on only seven small networks (no more than 1133 nodes). In terms of modularity, the results of DCAM3, VP, and LP are comparable: they are equal on five of seven data sets, and everyone wins on one data set.

Note that the VP and LP algorithms include a refinement procedure and perform well on only the seven small data sets. Also, for these small data sets, the modularity values provided by DCAM are very close to the upper bounds given in Agarwal and Kempe (2008) (the gap is zero on one data set, 0.001 on five of seven data sets, and 0.002 on one data set).

As for running time, on the seven data sets that can be handled by VP and LP, the CPU times of VP and LP vary from 0.69 to 386.82 seconds while those of DCAM3 are always smaller than 0.001 second.

5 Conclusion

We have studied the problem of detecting community structure in networks via the maximization of the modularity. By exploiting the special structure of the problem considered here, we have developed a fast and scalable DCA scheme (DCAM) that requires only a matrix-vector product at each iteration. The implementation is performed by an incremental approach that takes into account the sparsity of the modularity matrix. Our algorithm is neither a hierarchical partition nor a k-partition approach and does not require any refinement. Starting with a very large number of communities, DCAM furnishes, as output results, an optimal partition, together with the optimal number of communities c^*; the number of communities is discovered automatically during DCA's iterations. The number of clusters is updated at each iteration when empty clusters are found and computational efforts are considerably reduced. Moreover, the DCAM-like algorithm combined with an LPA procedure has been proved to be efficient for initializing DCAM.

Our algorithm can handle large networks with up to 4,194,304 nodes and 30,359,198 edges; say, the initial optimization problem has 6,073,352,192 variables and 6,077,546,496 constraints (as we start with $c^0 = 1448$ and the number of variables and of constraints are, respectively, nc and $nc + n$).

The DCAM has been compared to eight reference algorithms on a variety of real-world network data sets. Experimental results show that it outperforms reference algorithms not only on quality and rapidity but also on scalability, and it realizes a very good trade-off between the quality of solutions and running time.

In future work, we plan to investigate DCA for optimizing new modularity measures to identify overlapping communities or small communities that cannot be revealed by the original modularity maximization problem.

Acknowledgments

We are very grateful to the anonymous referees and the associate editor for their helpful and constructive comments that helped us to improve our letter. This research has been supported by Fonds Européens de Développement Régional Lorraine via the project Innovations techniques d'optimisation pour le traitement Massif de Données.

References

Agarwal, G., & Kempe, D. (2008). Modularity-maximizing graph communities via mathematical programming. *European Physical Journal B*, *66*(3), 409–418. doi: 10.1140/epjb/e2008-00425-1

Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Perron, S., & Liberti, L. (2010). Column generation algorithms for exact modularity maximisation in networks. *Phys. Rev. E*, *82*, 046112.

Arenas, A., Fernandez, A., & Gomez, S. (2008). Analysis of the structure of complex networks at different resolution levels. *New J. Phys. 10*, 053039. doi:10.1088/1367-2630/10/5/053039

Blondel, V. D., Guillaume, J., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *J. Stat. Mech.* doi: 10.1088/1742-5468/2008/10/P10008

Boccaletti, S., Ivanchenko, M., Latora, V., Pluchino, A., & Rapisarda, A. (2007). Detecting complex network modularity by dynamical clustering. *Physical Review E*, *75*, 045102.

Bradley, B. S., & Mangasarian, O. L. (1998). Feature selection via concave minimization and support vector machines. In J. Shavlik (Ed.), *Machine Learning Proceedings of the Fifteenth International Conferences* (pp. 82–90). San Francisco: Morgan Kaufmann.

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., & Wagner, D. (2008). On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, *20*, 172–188.

Cafieri, S., Hansen, P., & Liberti, L. (2011). A locally optimal heuristic for modularity maximization of networks. *Phys. Rev. E*, *83*, 056105.

Clauset, A., Newman, M. E. J., & Moore, C. (2004). Finding community structure in very large networks. *Phys. Rev. E*, *70*(6), 066111. doi: 10.1103/PhysRevE.70.066111

Collobert, R., Sinz, F., Weston, J., & Bottou, L. (2006). Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine Learning*. San Francisco: Morgan Kaufmann.

Danon, L., Duch, J., Diaz-Guilera, A., & Arenas, A. (2005). Comparing community structure identification. *Journal of Statistical Mechanics*, 2005, P09008. doi:10.1088/1742-5468/2005/09/P09008

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, *39*(1), 1–38.

Djidjev, H. (2008). A scalable multilevel algorithm for graph clustering and community structure detection. In *Algorithms and models for the web-graph* (pp. 117–128). Berlin: Springer-Verlag.

Duch, J., & Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Phys. Rev. E*, *72*(2), 027104. doi:10.1103/PhysRevE.72.027104

Emprise, Y. K. C., & Dit-Yan, Y. (2011). A convex formulation of modularity maximization for community detection. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*.

Evans, T. S., & Lambiotte, R. (2009). Line graphs, link partitions, and overlapping communities. *Phys. Rev. E*, *80*(1), 016105. doi: 10.1103/PhysRevE.80.016105

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, *486*, 75–174.

Fortunato, S., & Barthelemy, M. (2007). Resolution limit in community detection. In *PNAS*, *104*, 36–41. doi:10.1073/pnas.0605965104

Good, B. H., de Montjoye, Y. A., & Clauset, A. (2010). The performance of modularity maximization in practical contexts. *Phys. Rev. E*, *81*, 046106.

Guimera, R., & Amaral, L. (2005). Functional cartography of complex metabolic networks. *Nature*, *433*, 895–900.

Jonathan, Q. J., & Lisa, J. M. (2012). Modularity functions maximization with nonnegative relaxation facilitates community detection in networks. *Physica A: Statistical Mechanics and Its Applications*, *391*, 854–865.

Laura, B., Songsong, L., Lazaros, G. P., & Sophia, T. (2012). A mathematical programming approach to community structure detection in complex networks. In *Proceedings of the 22nd European Symposium on Computer Aided Process Engineering* (pp. 1387–1391). Amsterdam: Elsevier.

Lehmann, S., & Hansen, L. (2007). Deterministic modularity optimization. *Eur. Phys. J. B*, *60*(1), 83–88. doi:10.1140/epjb/e2007-00313-2

Le Thi, H. A. (2013). *DC programming and DCA*. http://lita.sciences.univ-metz.fr/~lethi/DCA.html

Le Thi, H. A., Belghiti, T., & Pham Dinh, T. (2006). A new efficient algorithm based on DC programming and DCA for clustering. *Journal of Global Optimization*, *37*, 593–608.

Le Thi, H. A., Le, H. M., & Pham Dinh, T. (2006). Optimization based DC programming and DCA for hierarchical clustering. *European Journal of Operational Research*, *183*, 1067–1085.

Le Thi, H. A., Le, H. M., & Pham Dinh, T. (2007). Fuzzy clustering based on nonconvex optimisation approaches using difference of convex (DC) functions algorithms. *Journal of Advances in Data Analysis and Classification*, *2*, 1–20.

Le Thi, H. A., Le, H. M., Nguyen, V. V., & Pham Dinh, T. (2008). A DC programming approach for feature selection in support vector machines learning. *Journal of Advances in Data Analysis and Classification*, *2*(3), 259–278.

Le Thi, H. A., Nguyen, V. V., & Ouchani, S. (2008). Gene selection for cancer classification using DCA. *Adv. Dat. Min. Appl. LNCS*, *5139*, 62–72.

Le Thi, H. A., & Pham Dinh, T. (2005). The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. *Annals of Operations Research*, *133*, 23–46.

Li, Z., Zhang, S., Wang, R. S., Zhang, X. S., & Chen, L. (2008). Quantitative function for community detection. *Phys. Rev. E*, *77*, 036109.

Liu, Y., & Shen, X. (2006). Multicategory ψ-learning. *Journal of the American Statistical Association*, *101*, 500–509.

Liu, Y., Shen, X., & Doss, H. (2005). Multicategory ψ-learning and support vector machine: Computational tools. *Journal of Computational and Graphical Statistics*, *14*, 219–236.

Mardia, K. V., Kent, J. T., & Bibby, J. M. (1979). *Multivariate analysis*. Orlando, FL: Academic Press.

Mariá, C. V. N., & Leonidas, S. P. (2013). Community detection by modularity maximization using GRASP with path relinking. *Computers and Operations Research*, *40*, 3121–3131.

Massen, C., & Doye, J. (2005). Identifying communities within energy landscapes. *Physical Review E*, *71*, 046101.

Medus, A., Acuna, G., & Dorso, C. (2005). Detection of community structures in networks via global optimization. *Physica A*, *358*, 593–604.

Nadakuditi, R. R., & Newman, M. E. J. (2012). Graph Spectra and the detectability of community structure in networks. *Phys. Rev. Lett.*, *108*, 188701.

Neumann, J., Schnörr, C., & Steidl, G. (2004). SVM-based feature selection by direct objective minimisation. In *Pattern Recognition, Proc. of 26th DAGM Symposium* (pp. 212–219). Berlin: Springer.

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, *69*(6), 066133. doi:10.1103/PhysRevE.69.066133

Newman, M. E. J. (2006a). Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, *74*(3), 036104. doi: 10.1103/PhysRevE.74.036104

Newman, M. E. J. (2006b). Modularity and community structure in networks. *PNAS*, *103*(23), 8577–8582.

Newman, M. E. J. (2010). *Networks: An introduction*. New York: Oxford University Press.

Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E*, *69*(2), 026113. doi:10.1103/PhysRevE.69.026113

Noack, A., & Rotta, R. (2009). Multi-level algorithms for modularity clustering. In *Proceedings of the 8th International Symposium on Experimental Algorithms*. Dordrecht: Kluwer.

Ong, C. S., & Le Thi, H. A. (2013). Learning sparse classifiers with difference of convex functions algorithms. *Optimization Methods and Software*, *28*, 830–854.

Pham Dinh, T., & Le Thi, H. A. (1997). Convex analysis approach to D.c programming: Theory, algorithms and applications. *Acta Mathematica Vietnamica*, *22*, 289–355.

Pham Dinh, T., & Le Thi, H. A. (1998). DC optimization algorithms for solving the trust region subproblem. *SIAM J. Optimization*, *8*, 476–505.

Pons, P., & Latapy, M. (2004). Computing communities in large networks using random walks. *J. Graph Alg. and App.*, *10*, 284–293.

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, *76*, 036106.

Richardson, T., Mucha, P., & Porter, M. (2009). Spectral tripartitioning of networks. *Physical Review E*, *80*, 036111.

Schuetz, P., & Caflisch, A. (2008). Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Physical Review E*, *77*, 046112.

Shen, X., Tseng, G. C., Zhang, X., & Wong, W. H. (2003). ψ-Learning. *Journal of the American Statistical Association*, *98*, 724–734.

Sun, Y., Danila, B., Josic, K., & Bassler, K. E. (2009). Improved community structure detection using a modified fine-tuning strategy. *Europhysics Letters*, *86*, no. 28004.

Tasgin, M., Herdagdelen, A., & Bingol, H. (2007). *Community detection in complex networks using genetic algorithms*. arXiv:0711.0491

Twan, V. L., & Elena, M. (2013). Graph clustering with local search optimization: The resolution bias of the objective function matters most. *Phys. Rev. E*, *87*, 012812.

Wakita, K., & Tsurumi, T. (2007). *Finding community structure in mega-scale social networks*. arXiv e-print cs/0702048

Weber, S., Schüle, T., & Schnörr, C. (2005). Prior learning and convex-concave regularization of binary tomography. *Electr. Notes in Discr. Math.*, *20*, 313–327.

White, S., & Smyth, P. (2005). A spectral clustering approach to finding communities in graph. In H. Kargupta, J. Srivastava, C. Kamath, & A. Goodman (Eds.), *Proceedings of the 5th SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics* (pp. 274–285). Philadelphia: SIAM.

Xu, G., Tsoka, S., & Papageorgiou, L. (2007). Finding community structures in complex networks using mixed integer optimization. *Eur. Physical Journal B*, *60*, 231–239.

Yuille, A. L., & Rangarajan, A. (2002). The convex concave procedure (CCCP). In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems, 14*. Cambridge, MA: MIT Press.

Received January 1, 2014; accepted June 23, 2014.