An Efficient Module Detection Algorithm for Large-Scale Complex Networks

Chuangchuang Sun and Ran Dai

Abstract—The module detection problem for large-scale complex networks has attracted attention to understand the interactions of nodes as well as the underlying network property. Since the concept of modularity is created to quantify the module property in a network, numerous efforts have been focused on solving the module detection problem via maximizing the modularity of a network. However, due to the NP-complete nature of the modularity maximization problem, existing algorithms, ranging from heuristic methods to greedy algorithms, are computationally demanding. Accordingly, we first reformulate the network module detection problem as Mixed Integer Quadratic Programming (MIQP) problem. Subsequently, an efficient and memory-saving algorithm based on inexact Augmented Lagrangian Method (ALM) is developed to solve MIQPs. By exploiting the explicit structure and sparsity of the problem settings, simplified and closed form solutions are obtained for subproblems of the inexact ALM. Simulation results from the proposed algorithm and the greedy algorithm using real-world databases are presented.

Index Terms—Network Module Detection; Community Detection; Mixed Integer Quadratic Programming; Augmented Lagrangian Method

I. INTRODUCTION

Many real world complex systems with multiple subsystems in various disciplines, ranging from sociology, biology, engineering, information technology, just to name a few, can be abstracted as networks. In those systems, instead of manipulating the system as a whole, e.g., through decentralized control, people wish to find the modules of a network to suppress the complexity. Besides, it is beneficial to better understand the interactions of nodes, as well as, the underlying property of the complex networks. Module detection is to group the nodes into modules such that the intra-module connectivity is obviously denser that the inter-module one. Consequently, module detection has attracted great attention in the study of complex networks.

Traditional approaches for solving network module detection problems typically include graph partitioning [1] and hierarchical clustering methods [2]. The main disadvantage for the former is the unreliable iterative bisectioning to split the graph and for the latter is the uncertainties to choose a partition among many options to better represent the module structure. Moreover, the divisive algorithms [3] has been developed to select edges according to their contribution importance to a specific performance index. In addition, methodologies based on partitional clustering and spectral clustering have been proposed [4], [5].

One challenge in network module detection problem is to quantify the module structure in a network. Newman in [3] proposed the notion modularity, which is subsequently accepted as the benchmark performance index for the module networks despite the resolution limit [6]. After introducing the concept of modularity, efforts on solving the module detection problems have been focused on maximizing the modularity of a network via optimization approaches. Furthermore, work in [7] demonstrates the equivalence between the method of modularity maximization and the method of maximum likelihood. Following the definition of modularity and justification of the model, Newman developed a series of algorithms to detect the number of modules (prerequisite of many existing algorithms) in a network [8] and to find these modules, including the matrix eigenvector method [9] and the greedy algorithm [10]. Variants of greedy algorithms have also been developed: see [11]. Other modularity optimization based algorithms or techniques include genetic algorithms [12], simulated annealing [13], extremal optimization [14], and etc. More details are addressed in the comprehensive survey [15]. On the other hands, the network module detection problem is also formulated as a Mixed Integer Quadratic Programming (MIQP) [16] problem and solved via commercial optimization solver, such as CPLEX. However, existing algorithms are subject to low efficiency when solving large-scale module detection problems. It has been recognized that the heuristic algorithms, such as genetic algorithm and simulated annealing, are computationally demanding. Furthermore, due to the NP-hardness of MIQP, the corresponding algorithms in the worst-case are exponential complexity and most are inefficient [17]–[19].

Based on the NP-hard nature of the modularity maximization [20], it is unrealistic to develop an efficient optimization algorithm to search for its global optimal solution within polynomial computation time. As a tradeoff or compromise, we aim to develop a local optimization algorithm with reasonable computation time. We first reformulate the module detection problem as a MIQP, which is equivalent to the one formulated in [16]. The new formulation is more compact and concise, i.e., with fewer number of variables and constraints, which facilitates developing a time and memory efficient algorithm that does not require employing commercial solvers. Algorithms for MIQPs generally include two categories. One is global optimization technique, such as branch and bound [18]. The other is convex relaxations, such as semidefinite relaxation [21]. Though relaxation techniques, in most cases, can only find a lower bound and not even a feasible one, there are some theoretical guarantee.
for special cases like the max-cut problems [22]. While the former is not scalable, solving a semidefinite programming for the latter is also computationally prohibitive in medium and large-scale problems.

In this paper, an algorithm based on inexact Augmented Lagrangian Method (ALM) is developed to solve the network module detection problem formulated as an MIQP. In the proposed inexact ALM, we partition the variables into two blocks, which leads to a closed form solution of each subproblem. In addition, we adequately exploit the special structure and sparsity of the problem settings to simplify the closed form solutions and save memory. The rest of the paper is organized as follows. In §II, the problem formulation of network module detection as an MIQP is described. The algorithm framework and the subproblem solutions are addressed in §III. Simulation results on various real-world databases with varying scales are presented in §IV. We conclude the paper with a few remarks in §V.

A. Preliminary

For a matrix $X \in \mathbb{R}^{m \times n}$, $X_{ij}$ denotes its entry on the $i$th row and $j$th column. Similarly, $X_{i:j,j:2}$ represent the entries in $i$th row and from $j$th column to $j$th column. $X_{i:j}$ represents the $i$th column of $X$. Moreover, we denote $x := \text{vec}(X) \in \mathbb{R}^{mn}$ as vectorization of $X$ by stacking the columns of $X$ on top of one another in order. Correspondingly, $X := \text{mat}(x)$ is the reverse operation. For two matrices $X$ and $Y$, $X \otimes Y$ represents the Kronecker matrix product. $\|X\|_F$ represents the Frobenius norm. $\text{blkdiag}(X_1, \ldots, X_n)$ represents a block diagonal matrix with square matrices $X_1, \ldots, X_n$ on the diagonal. For two matrices $X$ and $Y$ with the same dimensions, $X \odot Y$ represents the element-wise multiplication. Correspondingly, $X \otimes Y$ represents the element-wise division. For a vector $x$, $\text{Diag}(x)$ represents a diagonal matrix with $x$ as the diagonal entries.

II. PROBLEM FORMULATION

Modularity, the quantity to measure the difference between the intra-community and inter-community connection, is first defined in [3]. Its formulation is as follows

$$Q = \frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j), \quad (2.1)$$

where $A$ is the adjacency matrix, $k_i$ is the degree of node $i$, $n$ is the number of the nodes and $m$ is the number of the total edges. In this paper, we focus on undirected networks. Moreover, $C_i$ and $C_j$ are the communities to which vertex $i$ and $j$ belong to, respectively. Then $\delta(C_i, C_j)$ is defined as

$$\delta(C_i, C_j) = \begin{cases} 1, & C_i = C_j \\ 0, & C_i \neq C_j \end{cases}. \quad (2.2)$$

As we aim to detect the community in the network by maximizing the modularity function, it will be computationally favorable if $\delta(C_i, C_j)$ can be represented by a continuously differentiable function. For a two-community network, a bivalent vector, $s \in \{-1, 1\}^n$, is introduced to define $\delta(C_i, C_j) = \frac{1}{2} (s_i s_j + 1)$ with $s_i = -1$ ($s_i = 1$, resp.) representing that vertex $i$ is within the designated community (the other community, resp.). However, for a network with possible $k$ communities ($k > 2$), a single column vector like $s$ is insufficient to describe the relationship. Correspondingly, we define a binary matrix $X \in \{0, 1\}^{k \times n}$ such that

$$X_{ij} = \begin{cases} 1, & \text{if node } j \text{ is community } i \\ 0, & \text{otherwise} \end{cases}. \quad (2.3)$$

Moreover, as we exclude the cases with overlapping communities, i.e., one vertex belongs to one and only one community, it adds the following constraint on $X$,

$$\sum_{i=1}^{k} X_{ij} = 1, \forall j = 1, \ldots, n. \quad (2.3)$$

Consequently, $\delta(C_i, C_j) = X_i^T X_{i:j}$ represents its exact definition in (2.2), where $X_{i:j}$ is the $i$th column of matrix $X$. For notation ease, we define $B \in \mathbb{S}^n$ such that $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$. Accordingly, the modularity in (2.1) can be rewritten as

$$Q = \frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} \left( (A_{ij} - \frac{k_i k_j}{2m}) X_i^T X_{i:j} \right) = \frac{1}{2m} x^T (B \odot I_k)x, \quad (2.4)$$

where $x \in \mathbb{R}^{nk} := \text{vec}(X)$ is a vectorization of $X$. Furthermore, the equality constraint in (2.3) can be rewritten with regard to $x$ in the form of

$$\sum_{i=k(j-1)+1}^{kj} x_i = 1, \forall j = 1, \ldots, n \iff E x = 1, \quad (2.5)$$

with $E \in \mathbb{R}^{n \times nk}$, $E_{j,k(j-1)+1:kj} = 1$ and zeros otherwise for $j = 1, \ldots, n$.

Maximizing modularity in network module detection means to search for the best module structure to suppress complexity that the intra-module connectivity is much denser than the inter-module one. As a summary, the network module detection problem can be formulated as

$$\min_x \quad x^T M x \quad \text{s.t.} \quad E x = 1, \quad (2.6)$$

$x \in \{0, 1\}^{nk}$,

with $M = -\frac{1}{2m} (B \odot I_k)$. Problem (2.6) is a MIQP problem and generally NP-hard. Although the network module detection problem formulated in (2.6) is equivalent to the one in [16], the former is in a more compact form. The new formulation facilitates generating the proposed algorithm for solving (2.6), which is described in the following sections.

III. NETWORK MODULE DETECTION ALGORITHMS

A. Classical Alternating Direction Method of Multipliers

Given binary variables are involved in (2.6), we start with decoupling the binary constraints on $x$ from it linear equality
constraint, $Ex = 1$. An auxiliary variable $y$ is introduced and problem in (2.6) is reformulated as

$$
\begin{align*}
\min_{x, y} & \quad x^T M y \\
\text{s.t.} & \quad Ex = 1 \\
& \quad x^T H_i y - e_i^T x = 0, \forall i = 1, \ldots, n
\end{align*}
$$

(3.7)

In this way, $y$ is used to carry the binary constraints so that it can be decoupled from the other set of constraints, i.e., the linear equality constraint. It is intuitive to introduce the classical Alternating Direction Method of Multipliers (ADMM) framework to solve the above problem as ADMM is well known for partitioning variables into desired subsets to simplify computation [23]. In the ADMM framework, the variables in (3.7) are partitioned into two blocks, $x$ and $y$, and solved alternatively in sequence. When fixing one of the blocks and solving the other, the subproblems in each sequence of ADMM are straightforward. To be more specific, one subproblem is linearly constrained quadratic programming while the other is a projection onto the bivalent constraint. However, though ADMM does work well for some nonconvex problems, the general convergence is not theoretically guaranteed. For this specific problem, the classical ADMM has poor convergence when solving (3.7).

Accordingly, a time and storage efficient algorithm, based on the inexact augmented Lagrangian method, is proposed to solve the MIQP problem in (2.6).

### B. The Inexact ALM Algorithm

We first reformulate the binary constraint using a continuous quadratic function, expressed as

$$
\begin{align*}
x_i \in \{0, 1\} & \iff x_i^T H_i x_i - e_i^T x = 0, \forall i = 1, \ldots, n
\end{align*}
$$

(3.8)

where $(H_i)_{ii}$ is 1 and 0 for other entries and $e_i$ is a unit vector with the $i$th entry being 1 and 0 otherwise. Subsequently, by introducing an auxiliary variable $y$, (2.6) can be rewritten as

$$
\begin{align*}
\min_{x, y} & \quad x^T M y \\
\text{s.t.} & \quad Ex = 1 \\
& \quad x^T H_i y - e_i^T x = 0, \forall i = 1, \ldots, n \\
& \quad x = y.
\end{align*}
$$

(3.9)

Similar to the classical ADMM, introducing $y$ in the above formulation is to simplify the subproblem in each iteration, which will be explained later. In this way, the binary constraint is a coupling one getting both partitions of variables involved, which is different from that in (3.7). To solve (3.9) via the augmented Lagrangian method, the augmented Lagrangian function of (3.7) is constructed first, expressed as

$$
\mathcal{L}(x, y; \nu, \lambda, \mu) = x^T M y + \nu (Ex - 1) + \lambda (x^T H y - e^T x) + \langle \mu, x - y \rangle
$$

(3.10)

$$
+ \frac{\rho_1}{2} \|Ex - 1\|_F^2 + \frac{\rho_2}{2} \|x^T H y - e^T x\|_F^2 + \frac{\rho_3}{2} \|x - y\|_F^2,
$$

where $\nu \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^{nk}$ and $\mu \in \mathbb{R}^{nk}$ are the dual variables of the corresponding constraints and $\rho_1 > 0$, $\rho_2 > 0$, $\rho_3 > 0$ are the penalty coefficients for the augmented terms can be adjusted along the iterations. Unlike the augmented Lagrangian function in ADMM, here each constraint, including the local one $Ex = 1$ is also contained. Moreover, for notation simplicity, $x^T H y - e^T x : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ is defined as $x^T H y - e^T x := [x^T H_1 y - e_1^T x, \ldots, x^T H_n y - e_n^T x]^T$.

In the traditional augmented Lagrangian method, the dual variable sets $x$ and $y$ need be updated simultaneously. However, due to the fact that $\mathcal{L}(x, y; \nu, \lambda, \mu)$ (with the dual variables fixed) is a nonconvex forth order polynomial function, seeking for the global optimum is computationally complicated. However, $\mathcal{L}(x, y; \nu, \lambda, \mu)$ is bi-convex with respect to $x$ and $y$. Then instead of updating $x$ and $y$ simultaneously, an inexact ALM is proposed that also partitions the variables into two sets, $x$ and $y$ and update one with the other fixed. To solve (3.9), the algorithm alternates between these two variable sets followed by the update of the dual variables $(\nu, \lambda, \mu)$. With $(x^h, y^h; \nu^h, \lambda^h, \mu^h)$ obtained at step $h$, the subproblems of the inexact ALM at step $h + 1$ for (3.9) include

$$
\begin{align*}
x^{h+1} & := \arg \min_x \mathcal{L}(x, y^h; \nu^h, \lambda^h, \mu^h) \\
y^{h+1} & := \arg \min_y \mathcal{L}(x^{h+1}, y; \nu^h, \lambda^h, \mu^h)
\end{align*}
$$

(3.11a)

(3.11b)

$$
\begin{align*}
\nu^{h+1} & := \nu^h + \rho_1 (Ex^{h+1} - 1) \\
\lambda^{h+1} & := \lambda^h + \rho_2 (x^{h+1}^T H y^{h+1} - e^T x^{h+1}) \\
\mu^{h+1} & := \mu^h + \rho_3 (x^{h+1} - y^{h+1}).
\end{align*}
$$

(3.11c)

(3.11d)

(3.11e)

### C. Subproblem Solutions

In (3.11), updating $x^{h+1}$ using (3.11a) is to solve an unconstrained strongly convex optimization problem. As a result, the first order optimality conditions listed below will lead to its global optimal solution,

$$
\begin{align*}
\frac{\partial \mathcal{L}}{\partial x} = Cy + E^T \nu + \sum_{i=1}^n \lambda_i (H_i y - e_i) + \mu
\end{align*}
$$

(3.12)

+ $\rho_1 E^T (Ex - 1) + \rho_2 \sum_{i=1}^n (H_i y - e_i) (H_i y - e_i)^T x

+ \rho_3 (x - y) = 0.$

Note that in the above equation, the algorithm step index is ignored for simplicity. Furthermore, the computation cost in (3.12) can be significantly reduced by exploiting the sparsity of $H_i$ as well as $e_i$. From the quadratic function definition in (3.8), some terms in (3.12) can be simplified as

$$
\begin{align*}
\sum_{i=1}^n \lambda_i (H_i y - e_i) = \lambda \circ (y - 1)
\end{align*}
$$

(3.13)

$$
\begin{align*}
\sum_{i=1}^n (H_i y - e_i) (H_i y - e_i)^T = \text{Diag}((y - 1) \circ (y - 1)).
\end{align*}
$$

(3.14)

Now the focus is to solve the linear equality derived from (3.12). For conciseness, we rewrite it as $G_x = b_x$, where
Moreover, replacing the original objective

\[ y \rightarrow y - 1 \]

Since the Hessian of (3.11b) is a nonsingular diagonal matrix,

Analogously, we have

For large-scale network module detection problems, the complexity (at least \( O((nk)^2) \)) so far. Similarly, the backslash (left division) is also time consuming due to the same reason (complexity of \( O((nk)^2) \)). Subsequently, this motivates us to fully exploit the problem data structure and sparsity. As \( G_k \) is the sum of three components, it shows that the last two are diagonal and the first satisfies \( E^T E = I_n \otimes 1_{kk,k} = \text{blkdiag}(1_{kk,k}, \ldots, 1_{kk,k}, 1_{kk,k}) \in \mathbb{R}^{nk \times nk} \).

In other words, \( G_x \) is a block diagonal matrix with \( n \) square blocks of \( k \). As \( k \ll n \) in realistic problem settings, \( x \) can be solved via block-wise back slash. In one word, (3.13) can be solved block-wisely as,

\[
x_{I_k} = \left( \rho_1 1_{kk,k} + \rho_2 \text{Diag}((y_{I_k} - 1) \circ (y_{I_k} - 1)) + \rho_3 I_k \right) \backslash (b_x)_{I_k}, \forall i = 1, \ldots, n
\]  

(3.14)

where \( I_k = (i-1)k + 1 : ik \) is the index set. There are two advantages in solving (3.13) both computation-wise and memory-wise. For the former, with complexity \( O(nk^2) \), it avoids large-scale matrix inversion (\( \geq O((nk)^2) \)) or left division (\( O((nk)^2) \)), and full size matrix multiplication by taking advantage of the block diagonal structure. Recall \( k \ll n \), this reduction is significant. As for the latter, instead of caching \( G_x \in \mathbb{R}^{nk^k} \), it suffices to just store \( y \in \mathbb{R}^{nk^k} \), which substantially decreases the memory burden.

In a parallel scheme, the optimum of (3.11b) can be achieved by solving

\[
\frac{\partial L}{\partial y} = Cx + \sum_{i=1}^{n} \lambda_i H_i x - \mu + \rho_2 \sum_{i=1}^{n} (H_i x) \left( (H_i x)^T y - e_i^T x \right) + \rho_3 (y - x) = 0.
\]  

(3.15)

Analogously, we have

\[
\sum_{i=1}^{n} \lambda_i H_i x = \lambda \circ x
\]

\[
\sum_{i=1}^{n} (H_i x) \left( (H_i x)^T y - e_i^T x \right) = \text{Diag}(x \circ y) y - (x \circ y).
\]

Since the Hessian of (3.11b) is a nonsingular diagonal matrix, its inverse operation is straightforward. So far, the ease of solving the subproblems validates the advantages of introducing the auxiliary variable \( y \), the constraint \( x = y \), and the quadratic constraints. Though (3.9) is less compact compared to (3.6), it transforms the quadratic constraint into bilinear and binary variables into continuous ones. Subsequently, the augmented term in the augmented Lagrangian function is quadratic and convex instead of quartic and concave, which leads to closed form solution of two subproblems of (3.9). Moreover, replacing the original objective \( x^T M x \) by \( x^T M y \) is to avoid the presence of the dense \( M \) in the Hessian matrix in the \( x \)-update procedure. Otherwise, \( G_x \) will not be a block diagonal matrix and the aforementioned efficient method for \( x \)-update is no longer applicable. These are the key strategies to enhance the algorithm efficiency and make it promising for large-scale problems.

Recall the definition of \( M = -\frac{1}{2m} (B \otimes I_k) \in \mathbb{S}^{nk} \), which can cause another memory issue. Note that in each iteration, \( M \) appears in both subproblems, (3.11a) and (3.11b). However, for any two matrices \( A_1, B_1 \) and a vector \( v \) in appropriate dimension, we have the following property \( (B_1 \otimes A_1)v = vec(A_1 \text{mat}(v)B_1^T) \). As a result, instead of storing \( M \) and execute the full size matrix multiplication, the implementation below in \( x \)-update and analogously for \( y \)-update is executed

\[
Mx = -\frac{1}{2m} (B \otimes I_k)x = -\frac{1}{2m} \text{vec}(\text{mat}(x)B^T).
\]

In an analogous way, due to its special structure, the computation related to \( E \) can also be simplified such that it is not necessary to cache \( E \) either. Updating the Lagrange multipliers in (3.11c)-(3.11e) is straightforward. So far, we end solving the subproblems involved in the inexact ALM algorithm and it is summarized below. For conciseness, we keep \( M \) and \( E \) therein.

Algorithm 1: Inexact augmented Lagrangian algorithm for problem (3.9)

Input: Problem parameters \( n, k, A, B, C, E, \rho_1, \rho_2, \rho_3, \alpha_1 > 0, \alpha_2 > 0, \alpha_3 > 0 \) and initial point \( (x^0, y^0, \nu^0, \lambda^0, \mu^0) \)

Output: Local minimizer \((x, y)\) to (3.9)

begin

1) Set \( h = 0 \)
2) while not convergent
3) Update \( x \) via (3.14)
4) Update \( y \) via solving (3.15)
5) Update \( \nu, \lambda, \mu \) via (3.11c)-(3.11e)
6) Update \( \rho_1, \rho_2, \rho_3 \)

end

Again, the proposed formulation in (3.9) is equivalent to the one developed in [16], but more compact with fewer number of variables and constraints. The new formulation
enables a computationally efficient algorithm (Algorithm 1) for the network module detection problem with closed form solutions for subproblems in the inexact ALM algorithm. Rather than solving the network module detection problem as a general mixed integer nonlinear programming in [16], our proposed formulation and algorithm aim to improve scalability for large-scale problems.

IV. SIMULATION

In this section, we apply the proposed algorithm to multiple real-world databases with varying scales. For comparison, we apply the greedy algorithm from [11] to verify the improved computational efficiency of the proposed algorithm. All codes are written in MATLAB and all simulation is run on a standard desktop computer with a 3.60 GHz processor and a 32.0 RAM.

Both algorithms are applied the original network module detection problem formulated in (2.6). To test the robustness, for the small scale problems, we run both of the algorithms for three times. Note that random initials are generated for Algorithm 1. The stopping criterion is \( \frac{1}{\norm{\Delta x_k}} \leq 1 \times 10^{-3} \) and \( \alpha_i \geq 1, i = 1, 2, 3 \) are with small real values. As a result, the comparative results of two algorithms are provided in Table I. For small scale problems, the greedy algorithm outperforms Algorithm 1 in efficiency. However, in larger scale ones, Algorithm 1 conversely demonstrates the significant improvement of computational efficiency. In Table I, \( Q^* \) represents the best known modularity of the corresponding network [16]. \( Q_G, Q_{ALM}(\text{max}), Q_{ALM}(\text{median}) \) represent the modularity from greedy algorithm, the maximum and median of modularity from multiple trials of Algorithm 1, respectively. Moreover, \( T_G \) and \( T_{ALM} \) represent the running time (in seconds) of greedy algorithm and Algorithm 1, respectively, and ‘relative’ is the relative difference of the modularity from two algorithms, defined as \( \frac{Q_{ALM}(\text{max})-Q_G}{Q_G} \). Given the relative difference, it is also worth mentioning that our algorithm remains at comparable modularity value while improving the efficiency. Moreover, from the comparison of the maximum and median values from multiple trials of Algorithm 1, it verifies that Algorithm 1 is robust to random initials. Additionally, the computation time of our algorithm is less sensitive than that of the greedy algorithm, which makes it more promising for larger scale networks.

To better illustrate the comparison, Figure 1 demonstrates the time versus the number of network nodes for both algorithms. Moreover, we use the football database as an example to demonstrate the module detection result from Algorithm 1. Figure 2 shows that the coupling constraints converge to prime feasibility in a few iterations. Figure 3 provides the network after grouping the nodes into modules. It is obvious that the inter-module connectivity is more sparse than the intra-module connectivity. Similarly, such information can be validated in Fig. 4 which shows the permuted adjacency matrix. The nearly-block diagonal matrix proves the effectiveness of Algorithm 1 for solving network module detection problems from another perspective.

V. CONCLUSIONS

This paper aims to solve the module detection problem for large-scale complex networks to maximize modularity. We first formulate the network module detection problem as a Mixed Integer Quadratic Programming (MIQP) in a compact form. An time and memory efficient algorithm based on inexact Augmented Lagrangian Method (ALM) is developed to MIQPs. As the subproblems generated from the inexact ALM are of simple forms, each of the them leads to a closed form solution. Additionally, the special structure and sparsity
of the problem formulation are utilized to simplify the closed form solutions. Comparative results from the proposed algorithm and the greedy algorithm on solving real-world database examples are presented. Simulation results validate the efficiency of the proposed algorithm compared to the greedy algorithm for medium to large-scale problems while maintaining desired network modularity values.

VI. ACKNOWLEDGEMENTS

The authors would like to thank Antoine Scherrer for the MATLAB codes implementing greedy algorithm for comparison in this paper (https://perso.uclouvain.be/vincent.blondel/research/louvain.html).

REFERENCES


TABLE I

<table>
<thead>
<tr>
<th>Network</th>
<th>node</th>
<th>edge</th>
<th>$Q^*$</th>
<th>$Q_G$</th>
<th>$Q_{ALM}$ (max)</th>
<th>$Q_{ALM}$ (median)</th>
<th>$T_G/s$</th>
<th>$T_{ALM}/s$</th>
<th>relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>karate</td>
<td>34</td>
<td>78</td>
<td>0.420</td>
<td>0.4188</td>
<td>0.4118</td>
<td>0.4003</td>
<td>0.0045</td>
<td>0.0340</td>
<td>-0.0</td>
</tr>
<tr>
<td>dolphins</td>
<td>62</td>
<td>159</td>
<td>0.529</td>
<td>0.5188</td>
<td>0.5074</td>
<td>0.4902</td>
<td>0.0076</td>
<td>0.0398</td>
<td>-0.0220</td>
</tr>
<tr>
<td>lesmis</td>
<td>77</td>
<td>254</td>
<td>0.560</td>
<td>0.5556</td>
<td>0.5249</td>
<td>0.5241</td>
<td>0.0078</td>
<td>0.0516</td>
<td>-0.0551</td>
</tr>
<tr>
<td>polbooks</td>
<td>105</td>
<td>441</td>
<td>-</td>
<td>0.4986</td>
<td>0.4903</td>
<td>0.4651</td>
<td>0.0077</td>
<td>0.0768</td>
<td>-0.167</td>
</tr>
<tr>
<td>adjnoun</td>
<td>112</td>
<td>425</td>
<td>-</td>
<td>0.2906</td>
<td>0.2495</td>
<td>0.2257</td>
<td>0.0149</td>
<td>0.1278</td>
<td>-0.1416</td>
</tr>
<tr>
<td>football</td>
<td>115</td>
<td>613</td>
<td>-</td>
<td>0.6046</td>
<td>0.5979</td>
<td>0.5921</td>
<td>0.0160</td>
<td>0.0722</td>
<td>-0.0110</td>
</tr>
<tr>
<td>email</td>
<td>1,133</td>
<td>10,903</td>
<td>-</td>
<td>0.5406</td>
<td>0.5159</td>
<td>0.5016</td>
<td>0.4626</td>
<td>1.3594</td>
<td>-0.0457</td>
</tr>
<tr>
<td>power</td>
<td>4,941</td>
<td>6,394</td>
<td>-</td>
<td>0.7490</td>
<td>0.7491</td>
<td>0.7447</td>
<td>100.2699</td>
<td>11.31734</td>
<td>0.00</td>
</tr>
<tr>
<td>hep-th</td>
<td>8,361</td>
<td>15,751</td>
<td>-</td>
<td>0.6580</td>
<td>0.6531</td>
<td>0.6508</td>
<td>395.4231</td>
<td>41.1734</td>
<td>-0.195</td>
</tr>
<tr>
<td>astro-ph</td>
<td>16,706</td>
<td>121,251</td>
<td>-</td>
<td>0.6114</td>
<td>0.6114</td>
<td>0.6114</td>
<td>583.7437</td>
<td>111.3529</td>
<td>0.00</td>
</tr>
<tr>
<td>cond-mat</td>
<td>16,726</td>
<td>47,594</td>
<td>-</td>
<td>0.6201</td>
<td>0.6201</td>
<td>0.6201</td>
<td>1176.40</td>
<td>111.2393</td>
<td>-0.2321</td>
</tr>
</tbody>
</table>

Fig. 4. The (permuted) nearly-block diagonal adjacency matrix of the network. Each block corresponds to one module.