Journal of the Operations Research Society of Japan
Vol. , No. , pp.

A CUTTING PLANE ALGORITHM FOR MODULARITY MAXIMIZATION PROBLEM

Yoichi Izunaga
The Institute of Behavioral Sciences

Yoshitsugu Yamamoto
Shizuoka University

(Received October 8, 2014; Revised October 26, 2016)

Abstract Modularity proposed by Newman and Girvan is the most commonly used measure when the nodes of a graph are grouped into communities consisting of tightly connected nodes. We formulate the modularity maximization problem as a set partitioning problem, and propose an algorithm for the problem based on the linear programming relaxation. We solve the dual of the linear programming relaxation by using a cutting plane method. To mediate the slow convergence that cutting plane methods usually suffer, we propose a method for finding and simultaneously adding multiple cutting planes.

Keywords: Combinatorial optimization, community detection, modularity maximization, set partitioning, cutting planes

1. Introduction

One of the most important issues in the network analysis is to find a meaningful structure, which often addresses identifying or detecting community structure. Here communities are the sets of nodes such that each set consists of tightly connected nodes, but loosely connected each other. A variety of approaches to detect communities has been proposed. The cut size defined as the number of edges connecting communities is one of the most commonly used quality measures. Minimizing the cut size, known as the minimum cut method, however often results in forming a number of very small communities. Several variants to overcome this drawback have been introduced such as the ratio cut [32], the normalized cut [30], and min-max cut [11].

A novel quality measure, called modularity, has been proposed by Newman and Girvan [27]. The modularity was originally used as a stopping criterion of the hierarchical divisive algorithm [27], and Newman [25] suggested an approach of maximizing the modularity due to the observation that a high value of the modularity leads to a good community structure. Then the modularity maximization became one of the central subjects of research. The *NP*-hardness of the modularity maximization problem shown by Brandes *et al.* [4] turned researchers' attention to heuristic algorithms, which resulted in several efficient heuristic algorithms such as the linear programming with rounding procedure by Agarwal and Kempe [1], the hierarchical agglomerative method by Clauset *et al.* [8], the simulated annealing by Guimerá and Amaral [18], and the spectral divisive method by Newman [26].

On the other hand, among exact algorithms three approaches should be mentioned. The first approach is based on the formulation of the problem as a clique partitioning problem proposed by Grötschel and Wakabayashi [17]. In this formulation, a binary variable corresponding to each pair of nodes represents whether the two nodes belong to the same community. Then the numbers of variables and constraints amount to $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$,

respectively, both of which grow rapidly with the number n of nodes. Based on this formulation, Aloise *et al.* [2] solved instances up to 115 nodes by using the cutting plane algorithm proposed by Grötschel and Wakabayashi [17].

The second approach is based on the set partitioning problem. Since this formulation has to take into account all nonempty subsets of the node set, it has $\mathcal{O}(2^n)$ variables. We can hardly secure the computational resource to hold the problem when n is large, say more than 200. Column generation technique is a common trick to deal with such problems, but the auxiliary problem of determining the entering column ends up as a quadratic programming in binary variables, which is hard to solve exactly. Moreover, the column generation is known to suffer slow convergence. To overcome this defect, Du Merle *et al.* [12] have proposed an acceleration method, called stabilized column generation method. Some computational results are reported also in Aloise *et al.* [2].

The third approach is based on the quadratic programming formulation. This formulation suffers from symmetry, that is, there exists a large number of equivalent solutions for each community structure. As a consequence the computational time increases due to the large search space of solutions. Xu *et al.* [33] solved instances up to 104 nodes by introducing some symmetry breaking constraints.

In this paper, based on the second formulation, set partitioning, we propose cutting plane algorithms for solving the LP relaxation of the set partitioning problem. Our cutting plane algorithms, which share a basic framework with the column generation proposed by Aloise *et al.* [2], have a merit of being able to provide the upper bounds on the optimal modularity at every iteration by solving a small relaxation problem. We incorporate several techniques into the algorithms: multiple cutting planes, rounding heuristics and pegging test. We report some computational results and demonstrate that our algorithm outperforms some of existing heuristics.

This paper is organized as follows. We give the definition of the modularity in Section 2 and formulate the modularity maximization problem as a set partitioning problem and a quadratic programming in Section 3. In Section 4, we introduce several relaxation problems and their dual problems. In Section 5, after reviewing cutting plane algorithms, we propose two versions of the cutting plane algorithm and explain a method to calculate an upper bound. In Section 6, we report the computational experiments of the proposed algorithm. Finally we give some conclusions in Section 7.

2. Modularity Maximization Problem

Let $G = (V, E)$ be an undirected graph with the set $V = \{1, 2, \ldots, n\}$ of n nodes and the set E of m edges. We assume that the graph G is simple, that is, G has neither loops nor parallel edges. We say that $\Pi = \{C_1, C_2, \ldots, C_k\}$ is a *partition* of V if $V = \bigcup_{p=1}^{k} C_p$, $C_p \cap C_q = \emptyset$ for any distinct p and q in $\{1, 2, \ldots, k\}$, and $C_p \neq \emptyset$ for any $p \in \{1, 2, \ldots, k\}$. Each member C_p of a partition is called a *cluster* or a *community*. We denote the set of edges that have one end-node in C and the other end-node in C' by $E(C, C')$. When $C = C'$, we abbreviate $E(C, C')$ to $E(C)$ for the sake of simplicity. *Modularity*, denoted by $\mu(\Pi)$, of a partition Π is defined as

$$\mu(\Pi) = \sum_{C \in \Pi} \left(\frac{|E(C)|}{m} - \left(\frac{2|E(C)| + \sum_{C' \in \Pi \setminus \{C\}} |E(C, C')|}{2m} \right)^2 \right), \tag{2.1}$$

where $|\cdot|$ denotes the cardinality of the corresponding set. Roughly speaking, the first term of (2.1) represents the fraction of the number of edges in a community, whereas the

second term of (2.1) represents the fraction of the number of edges connecting nodes in different communities. For more correct explanation of the modularity, we refer the reader to Newman [26]. It is known that the modularity $\mu(\Pi)$ falls between $-1/2$ and 1 for any partition of V. See Brandes *et al.* [4] for details. In addition, when Π consists of a single community, i.e., $\Pi = \{V\}$, then we have $\mu(\{V\}) = 0$. We refer to 0 and 1 as trivial lower and upper bounds on the modularity, respectively.

For $i, j \in V$, let e_{ij} be

$$e_{ij} = \begin{cases} 1 & \text{when } \{i,j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

i.e., the (i,j) element of the adjacency matrix of graph G, and d_i be the degree of node i, i.e., $d_i = |\{\, j \in V \mid \{i,j\} \in E \,\}|$, and $\pi(i)$ be the index of community which node i belongs to, i.e., $\pi(i) = p$ means $i \in C_p$. Then $\mu(\Pi)$ is rewritten as

$$\mu(\Pi) = \frac{1}{2m} \sum_{i \in V} \sum_{j \in V} \left(e_{ij} - \frac{d_i d_j}{2m} \right) \delta(\pi(i), \pi(j)),$$

where δ is the Kronecker delta, i.e.,

$$\delta(p,q) = \begin{cases} 1 & \text{when } p = q \\ 0 & \text{otherwise.} \end{cases}$$

Modularity maximization problem, (MM) for short, is the problem of finding a partition of V that maximizes the modularity $\mu(\Pi)$. Denoting $(e_{ij} - d_i d_j/2m)$ by w_{ij}, then the problem is formulated as

$$\text{(MM)}: \quad \left| \begin{array}{ll} \text{maximize} & \dfrac{1}{2m} \displaystyle\sum_{i \in V} \sum_{j \in V} w_{ij} \delta(\pi(i), \pi(j)) \\ \text{subject to} & \Pi \text{ is a partition of } V. \end{array} \right.$$

3. Formulations

In this section, we introduce two different formulations of the modularity maximization problem. The first one is based on the integer programming formulation, and the second one is based on the quadratic programming formulation.

3.1. Integer programming formulation

Let $\mathcal{P}$ denote the family of all nonempty subsets of V. Note that $\mathcal{P}$ is composed of $2^n - 1$ subsets of V. Introducing a binary variable z_C for each $C \in \mathcal{P}$, a partition Π is represented by the $(2^n - 1)$-dimensional binary vector $\boldsymbol{z} = (z_C)_{C \in \mathcal{P}}$ defined as

$$z_C = \begin{cases} 1 & \text{when } C \in \Pi \\ 0 & \text{otherwise.} \end{cases}$$

This enables us to formulate problem (MM) as an integer programming problem. For each $i \in V$ and $C \in \mathcal{P}$, let a_{iC} be defined by

$$a_{iC} = \begin{cases} 1 & \text{when } i \in C \\ 0 & \text{otherwise.} \end{cases}$$

Note that the column $\boldsymbol{a}_C = (a_{1C}, \ldots, a_{nC})^\top$ is the incidence vector of community C, i.e., $C = \{\, i \in V \mid a_{iC} = 1 \,\}$. For each $C \in \mathcal{P}$, let f_C be

$$f_C = \frac{1}{2m} \sum_{i \in C} \sum_{j \in C} w_{ij}, \tag{3.1}$$

which is rewritten as

$$= \frac{1}{2m} \sum_{i \in V} \sum_{j \in V} w_{ij} a_{iC} a_{jC}.$$

The constant f_C represents the contribution of community C to the objective function $\mu(\Pi)$ when community C is selected as a member of the partition Π. Thus (MM) is formulated as the integer programming (P):

$$(P): \quad \left| \begin{array}{lll} \text{maximize} & \displaystyle\sum_{C \in \mathcal{P}} f_C z_C & \\ \text{subject to} & \displaystyle\sum_{C \in \mathcal{P}} a_{iC} z_C = 1 & (i \in V) \\ & z_C \in \{0,1\} & (C \in \mathcal{P}). \end{array} \right.$$

Since the first set of constraints states that the communities adopted form a partition of V, this problem is called a *set partitioning problem*. From now on, we will call the first set of constraints *set partitioning constraints*. Due to its huge number of variables this problem easily becomes computationally intractable as the number of nodes grows.

3.2. Quadratic programming formulation

The quadratic programming formulation for the modularity maximization problem has been proposed by Xu *et al.* [33]. Though the optimal number of communities is a priori unknown, we suppose an upper bound on the optimal number of communities is known, and denote it by t. Let T be an index set $\{1, 2, \ldots, t\}$. For each edge $r = \{i, j\} \in E$ and each $p \in T$, let x_{rp} be the binary variable such that

$$x_{rp} = \begin{cases} 1 & \text{when } r \in E(C_p) \\ 0 & \text{otherwise.} \end{cases}$$

For each node $i \in V$ and each $p \in T$, let y_{ip} be defined by

$$y_{ip} = \begin{cases} 1 & \text{when } i \in C_p \\ 0 & \text{otherwise.} \end{cases}$$

Then (MM) is formulated as the following quadratic programming:

$$(QP): \quad \left| \begin{array}{lll} \text{maximize} & \displaystyle\sum_{p \in T} \left(\frac{1}{m} \sum_{r \in E} x_{rp} - \frac{1}{4m^2} \left(\sum_{i \in V} d_i y_{ip} \right)^2 \right) & \\ \text{subject to} & \displaystyle\sum_{p \in T} y_{ip} = 1 & (i \in V) \\ & x_{rp} \le y_{ip} & (r = \{i,j\} \in E, p \in T) \\ & x_{rp} \le y_{jp} & (r = \{i,j\} \in E, p \in T) \\ & x_{rp} \in \{0,1\} & (r \in E, p \in T) \\ & y_{ip} \in \{0,1\} & (i \in V, p \in T). \end{array} \right.$$

The first set of constraints impose that each node belongs to exactly one community, and the remaining constraints express that any edge $r = \{i, j\}$ belongs to a set of edges $E(C_p)$ if both end-nodes i, j belong to the community C_p. Note that the binary constraint of the variable x_{rp} is redundant owing to the objective function of maximizing with respect to x_{rp}, hence it could be deleted or relaxed to $0 \leq x_{rp} \leq 1$.

This formulation suffers from *symmetry*, that is, re-indexing some communities yields alternative equivalent solutions. Such solutions are called symmetric solutions. As a consequence, branch-and-bound algorithms tend not to work well. See, Chapter 9 of Conforti *et al.* [9] for instance. Xu *et al.* [33] have proposed some valid inequalities to get rid of the symmetric solutions from the feasible region. In this paper we will not solve the problem (QP), but make use of this problem to obtain an upper bound on the optimal value of (P) in Subsection 5.2.

4. Relaxation Problems and Related Dual Problems

Not only the number of variables but also their integrality makes problem (P) a highly intractable problem. Then it would be a natural and clever strategy to consider relaxation problems for the useful information about the solution of (P). See, for example [6, 7, 29, 31]. The first choice to consider would be the *Linear Programming relaxation*, *LP relaxation* for short, where the binary constraint $z_C \in \{0, 1\}$ is replaced by $0 \leq z_C \leq 1$. It is given as

$$(RP): \quad \left| \begin{array}{lll} \text{maximize} & \displaystyle\sum_{C \in \mathcal{P}} f_C z_C & \\ \text{subject to} & \displaystyle\sum_{C \in \mathcal{P}} a_{iC} z_C = 1 & (i \in V) \\ & z_C \geq 0 & (C \in \mathcal{P}). \end{array} \right.$$

The upper bound constraints $z_C \leq 1$ are redundant owing to the set partitioning constraints, hence omitted. The linear programming dual problem of (RP) is given as the following (RD):

$$(RD): \quad \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i \in V} \lambda_i & \\ \text{subject to} & \displaystyle\sum_{i \in V} a_{iC} \lambda_i \geq f_C & (C \in \mathcal{P}) \\ & \lambda_i \in \mathbb{R} & (i \in V). \end{array} \right.$$

Now let us denote the feasible region and the optimal value of an optimization problem, say Q, by $\mathcal{F}(Q)$ and $\omega(Q)$, respectively. Since (RP) is a relaxation problem of (P), it holds that $\mathcal{F}(P) \subseteq \mathcal{F}(RP)$, hence $\omega(P) \leq \omega(RP)$. Applying the linear programming duality theorem to the primal dual pair (RP) and (RD), we see $\omega(P) \leq \omega(RD)$. Namely, solving (RD) we will obtain an upper bound on $\omega(P)$. The optimal solution of (RP) often provides a clue as to possibly a good feasible solution of (P) with aid of the information collected in solving its dual problem (RD). Although (RD) has only n variables, its exponentially large number of constraints makes it intractable.

Another interesting way is *Lagrangian relaxation*, see, e.g., Conforti *et al.* [9], Fisher [13] and Geoffrion [14]. We relax the set partitioning constraints by adding them to the objective function as a penalty with the Lagrangian multiplier vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^n$, then we obtain the following Lagrangian relaxation problem $(LR(\boldsymbol{\lambda}))$ having only binary variable

constraint:

$$(LR(\boldsymbol{\lambda})): \quad \left| \begin{array}{ll} \text{maximize} & \displaystyle\sum_{C\in\mathcal{P}} f_C z_C + \sum_{i\in V}\lambda_i\left(1-\sum_{C\in\mathcal{P}} a_{iC}z_C\right) \\ & \displaystyle= \sum_{C\in\mathcal{P}}\gamma_C(\boldsymbol{\lambda})z_C + \sum_{i\in V}\lambda_i \\ \text{subject to} & z_C\in\{0,1\} \qquad\qquad (C\in\mathcal{P}) \end{array} \right.$$

where $\gamma_C(\boldsymbol{\lambda}) = f_C - \sum_{i\in V} a_{iC}\lambda_i$. For a given multiplier vector $\boldsymbol{\lambda}$, we can obtain an optimal solution $\boldsymbol{z}(\boldsymbol{\lambda}) = (z_C(\boldsymbol{\lambda}))_{C\in\mathcal{P}}$ by simply setting $z_C(\boldsymbol{\lambda}) = 1$ if $\gamma_C(\boldsymbol{\lambda}) > 0$, and $z_C(\boldsymbol{\lambda}) = 0$ otherwise. Then the optimal value $\omega(LR(\boldsymbol{\lambda}))$ provides an upper bound on $\omega(P)$ for any $\boldsymbol{\lambda}$, and the upper bound given by the problem $(LR(\boldsymbol{\lambda}))$ depends on a choice of the multiplier vector. The problem of finding the best upper bound on $\omega(P)$ is called the *Lagrangian dual problem*, which is given as

$$(LD): \quad \left| \begin{array}{ll} \text{minimize} & \omega(LR(\boldsymbol{\lambda})) \\ \text{subject to} & \boldsymbol{\lambda}\in\mathbb{R}^n. \end{array} \right.$$

The objective function of (LD) is a piecewise linear convex function with respect to $\boldsymbol{\lambda}$, hence sub-differentiable. One of the most commonly used method for this problem is the *subgradient method* (See e.g., [9]).

Now, we consider continuous relaxation problem of $(LR(\boldsymbol{\lambda}))$, i.e., replacing the binary constraint by $0 \le z_C \le 1$, and denote it by $(\overline{LR(\boldsymbol{\lambda})})$. Clearly, any optimal solution of $(\overline{LR(\boldsymbol{\lambda})})$ is also optimal for $(LR(\boldsymbol{\lambda}))$, which is called *integrality property* [14]. Under this property, the optimal value of (LD) coincides with that of the problem (RD), hence the problems (RD) and (LD) yield upper bounds of the same quality, and the subgradient method for (LD) is more attractive than the simplex method as well as barrier method for (RD) in terms of low computational burden and low memory consumption. However, the Lagrangian multiplier $\boldsymbol{\lambda}$ obtained by the subgradient method does not necessarily satisfy the dual feasibility condition $\gamma_C(\boldsymbol{\lambda}) \le 0$ unlike the case of applying the simplex method or barrier method to (RD), which may causes undesirable issue in cutting plane algorithm. Specifically, the algorithm may repeatedly generate a cut which is already added.

Boschetti *et al.* [3] have proposed a parametric relaxation for the set partitioning problem, and the dual problem to their relaxation problem also provides upper bounds on the same level with those provided by LP relaxation even if the parameter in the relaxation problem is appropriately set on.

As mentioned above, LP relaxation provides a tight upper bound for the set partitioning problem and produces a solution with a favorable feature, dual feasibility, so that we consider the LP relaxation problem in this paper.

5. Cutting Plane Algorithm

The constraints of problem (RD) far outnumber the variables, hence most of them should not be binding at an optimal solution. The cutting plane algorithm is one of commonly used methods for LP problems of this kind. We will give a brief review of cutting plane algorithms, and then propose an algorithm for (RD).

5.1. Prototype of cutting plane algorithm

The key idea of the cutting plane algorithm is to deal with a small subfamily $\mathcal{C}$ of $\mathcal{P}$, and instead of (RD), to solve the following problem with fewer constraints:

$$(RD(\mathcal{C})): \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i\in V} \lambda_i & \\ \text{subject to} & \displaystyle\sum_{i\in V} a_{iC}\lambda_i \geq f_C & (C \in \mathcal{C}) \\ & \lambda_i \in \mathbb{R} & (i \in V). \end{array} \right.$$

Let $\boldsymbol{\lambda}(\mathcal{C})$ denote an optimal solution of $(RD(\mathcal{C}))$. Since the constraints $\sum_{i\in V} a_{iC}\lambda_i \geq f_C$ for $C \in \mathcal{P} \setminus \mathcal{C}$ are not considered, it is not necessarily a feasible solution of (RD). To check the feasibility of $\boldsymbol{\lambda}(\mathcal{C})$, we define a measure of violation $\gamma_C(\boldsymbol{\lambda}(\mathcal{C}))$ of the constraint corresponding to C as

$$\gamma_C(\boldsymbol{\lambda}(\mathcal{C})) = f_C - \sum_{i\in V} a_{iC}\lambda_i(\mathcal{C}). \tag{5.1}$$

Note that $\gamma_C(\boldsymbol{\lambda}(\mathcal{C})) \leq 0$ for all $C \in \mathcal{C}$. When $\gamma_C(\boldsymbol{\lambda}(\mathcal{C})) \leq 0$ for all $C \in \mathcal{P} \setminus \mathcal{C}$, $\boldsymbol{\lambda}(\mathcal{C})$ is a feasible solution of problem (RD), hence an optimal solution of problem (RD). When

$$\gamma_C(\boldsymbol{\lambda}(\mathcal{C})) > 0$$

holds for some $C \in \mathcal{P} \setminus \mathcal{C}$, adding this C to $\mathcal{C}$ can lead to an improvement of the optimal value of problem $(RD(\mathcal{C}))$, i.e., $\omega(RD(\mathcal{C} \cup \{C\})) \geq \omega(RD(\mathcal{C}))$. Substituting (3.1) for f_C of (5.1) yields

$$\gamma_C(\boldsymbol{\lambda}(\mathcal{C})) = \frac{1}{2m}\sum_{i\in V}\sum_{j\in V} w_{ij}a_{iC}a_{jC} - \sum_{i\in V} a_{iC}\lambda_i(\mathcal{C}),$$

hence the problem of maximizing $\gamma_C(\boldsymbol{\lambda}(\mathcal{C}))$ over $\mathcal{P}$ is formulated as the problem $(\overline{SP(\boldsymbol{\lambda}(\mathcal{C}))})$ with a quadratic objective function in binary variables:

$$(\overline{SP(\boldsymbol{\lambda}(\mathcal{C}))}): \left| \begin{array}{lll} \text{maximize} & \displaystyle\frac{1}{2m}\sum_{i\in V}\sum_{j\in V} w_{ij}y_iy_j - \sum_{i\in V}\lambda_i(\mathcal{C})y_i & \\ \text{subject to} & y_i \in \{0,1\} & (i \in V). \end{array} \right.$$

An optimal solution $\boldsymbol{y}^*$ of this problem provides the incidence vector of the community that maximizes $\gamma_C(\boldsymbol{\lambda}(\mathcal{C}))$ over $\mathcal{P}$. Since $\boldsymbol{y} = \boldsymbol{0}$ is a feasible solution of this problem, the optimal value is non-negative. According to the quadratic formulation presented in Subsection 3.2, $(\overline{SP(\boldsymbol{\lambda}(\mathcal{C}))})$ is equivalently formulated as the following problem with a quadratic concave objective function:

$$(SP(\boldsymbol{\lambda}(\mathcal{C}))): \left| \begin{array}{lll} \text{maximize} & \displaystyle\frac{1}{m}\sum_{r\in E} x_r - \frac{1}{4m^2}\left(\sum_{i\in V} d_iy_i\right)^2 - \sum_{i\in V}\lambda_i(\mathcal{C})y_i & \\ \text{subject to} & x_r \leq y_i & (r = \{i,j\} \in E) \\ & x_r \leq y_j & (r = \{i,j\} \in E) \\ & 0 \leq x_r \leq 1 & (r \in E) \\ & y_i \in \{0,1\} & (i \in V). \end{array} \right.$$

For each edge $r = \{i,j\} \in E$, a binary variable x_r is equal to 1 when both end-nodes i, j of edge r belong to a community that maximizes $\gamma_C(\boldsymbol{\lambda}(\mathcal{C}))$, and a variable y_i, for each $i \in V$,

is equal to 1 when node i belongs to the community and 0 otherwise. Having found $\boldsymbol{y}^*$ with positive optimal value, we have only to add the constraint

$$\sum_{i \in V} y_i^* \lambda_i \geq f^*$$

to $(RD(\mathcal{C}))$, where

$$f^* = \frac{1}{2m} \sum_{i \in V} \sum_{j \in V} w_{ij} y_i^* y_j^*.$$

From the above discussion, a prototype of the cutting plane algorithm is given as follows.

Prototype of the Cutting Plane Algorithm

`Step 0`

Let $\mathcal{C}$ be an initial family of nonempty subsets of V.

`Step 1`

Solve $(RD(\mathcal{C}))$ to obtain an optimal solution $\boldsymbol{\lambda}(\mathcal{C})$ and the optimal value $\omega(RD(\mathcal{C}))$.

`Step 2`

Solve $(SP(\boldsymbol{\lambda}(\mathcal{C})))$ and set $\boldsymbol{y}^*$ be an optimal solution.

`Step 3`

`If` $\omega(SP(\boldsymbol{\lambda}(\mathcal{C}))) \leq 0$ `then`

Set $\mathcal{C}^\star \leftarrow \mathcal{C}$ and $\omega^\star \leftarrow \omega(RD(\mathcal{C}))$. Output $\mathcal{C}^\star$ and $\omega^\star$, and terminate.

`else`

Set $C \leftarrow \{\, i \in V \mid y_i^* = 1 \,\}$ and increment $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$. Return to `Step 1`.

`end if`

Note that one can regard the cutting plane algorithm for a dual LP problem as column generation algorithm for a primal problem corresponding to the above dual problem. We refer the reader to Desaulniers *et al.* [10].

The initial subfamily $\mathcal{C}$ could be empty, but a clever choice may enhance the efficiency of the algorithm. In our experiments we collected all singletons of V to make the initial family $\mathcal{C}$. When the algorithm terminates, we have solved (RD), hence also (RP), which usually admits a fractional optimal solution. The final family $\mathcal{C}^\star$ however would yield a set of primal variables that are likely to be positive at an optimal solution of problem (P). Then we propose to solve the following problem $(P(\mathcal{C}^\star))$ of variables z_C with $C \in \mathcal{C}^\star$.

$$(P(\mathcal{C}^\star)): \quad \left| \begin{array}{lll} \text{maximize} & \displaystyle\sum_{C \in \mathcal{C}^\star} f_C z_C & \\ \text{subject to} & \displaystyle\sum_{C \in \mathcal{C}^\star} a_{iC} z_C = 1 & (i \in V) \\ & z_C \in \{0,1\} & (C \in \mathcal{C}^\star). \end{array} \right.$$

This problem is expected to have much fewer variables than problem (P) does, so that it could be solved within a reasonable time by an IP solver, e.g., CPLEX, Gurobi and Xpress. Furthermore, we can fix some variables corresponding to subsets in $\mathcal{C}^\star$ without loss of the optimality of the solution for the problem $(P(\mathcal{C}^\star))$. This preprocessing technique for $(P(\mathcal{C}^\star))$ is called *pegging test* [31]. In order to explain the pegging test, we first consider the following

Lagrangian relaxation problem $(LR(\mathcal{C}^\star, \boldsymbol{\lambda}))$ of $(P(\mathcal{C}^\star))$.

$$(LR(\mathcal{C}^\star, \boldsymbol{\lambda})): \quad \left| \begin{array}{ll} \text{maximize} & \displaystyle\sum_{C \in \mathcal{C}^\star} f_C z_C + \sum_{i \in V} \lambda_i \left(1 - \sum_{C \in \mathcal{C}^\star} a_{iC} z_C\right) \\ \text{subject to} & z_C \in \{0,1\} \quad (C \in \mathcal{C}^\star). \end{array} \right.$$

From (5.1), the objective function of $(LR(\mathcal{C}^\star, \boldsymbol{\lambda}))$ is written as

$$\sum_{C \in \mathcal{C}^\star} \gamma_C(\boldsymbol{\lambda}) z_C + \sum_{i \in V} \lambda_i.$$

For a given multiplier vector $\boldsymbol{\lambda} \in \mathbb{R}^n$, we can obtain an optimal solution $\boldsymbol{z}(\boldsymbol{\lambda})$ of $(LR(\mathcal{C}^\star, \boldsymbol{\lambda}))$ by simply setting $z_C(\boldsymbol{\lambda}) = 1$ if $\gamma_C(\boldsymbol{\lambda}) > 0$, and $z_C(\boldsymbol{\lambda}) = 0$ otherwise.

Proposition 5.1. *Let LB be a lower bound on (P) and $\boldsymbol{\lambda}$ be an optimal solution of $(RD(\mathcal{C}^\star))$, respectively. If $\sum_{i \in V} \lambda_i + \gamma_C(\boldsymbol{\lambda}) < LB$, then $z_C = 0$ for any optimal solution of the problem $(P(\mathcal{C}^\star))$.*

Proof. Since the problem $(LR(\mathcal{C}^\star, \boldsymbol{\lambda}))$ is a relaxation problem of $(P(\mathcal{C}^\star))$, we have

$$\omega(P(\mathcal{C}^\star \mid z_C = 1)) \leq \omega(LR(\mathcal{C}^\star, \boldsymbol{\lambda} \mid z_C = 1)). \tag{5.2}$$

For any $C \in \mathcal{C}^\star$, $\gamma_C(\boldsymbol{\lambda})$ is non-positive due to the feasibility of $\boldsymbol{\lambda}$ for the problem $(RD(\mathcal{C}^\star))$, hence $\omega(LR(\mathcal{C}^\star, \boldsymbol{\lambda})) = \sum_{i \in V} \lambda_i$ holds. Moreover we obtain

$$\omega(LR(\mathcal{C}^\star, \boldsymbol{\lambda} \mid z_C = 1)) = \sum_{i \in V} \lambda_i + \gamma_C(\boldsymbol{\lambda}). \tag{5.3}$$

From (5.2), (5.3) and the assumption, we have the following inequality

$$\omega(P(\mathcal{C}^\star \mid z_C = 1)) \leq \sum_{i \in V} \lambda_i + \gamma_C(\boldsymbol{\lambda}) < \mathrm{LB}.$$

The above inequality $\omega(P(\mathcal{C}^\star \mid z_C = 1)) < \mathrm{LB}$ implies the non-existence of the optimal solution that satisfies $z_C = 1$. Therefore $z_C = 0$ for any optimal solution of the problem $(P(\mathcal{C}^\star))$. □

We put off the description of the heuristics to obtain a lower bound LB on $\omega(P)$ until Subsection 5.2. Lacking variables z_C with C not in $\mathcal{C}^\star$, $(P(\mathcal{C}^\star))$ provides a lower bound on $\omega(P)$. Then

$$\omega(P(\mathcal{C}^\star)) \leq \omega(P) \leq \omega(RD(\mathcal{C}^\star)).$$

The value $\omega(RD(\mathcal{C}^\star)) - \omega(P(\mathcal{C}^\star))$ provides an upper bound of the difference between $\omega(P(\mathcal{C}^\star))$ and $\omega(P)$, hence the quality of the solution of $(P(\mathcal{C}^\star))$ given by an IP solver.

5.2. Proposed cutting plane algorithms

In this subsection we first discuss the stopping criterion of the cutting plane algorithm. Generally, this algorithm suffers from slow convergence, and, many iterations may be needed to prove the optimality of (RD) after the optimal value of (RD) has been obtained. Even so, it should be noted that we can evaluate the quality of the current optimal value $\omega(RD(\mathcal{C}))$ by means of the difference between $\omega(RD(\mathcal{C}))$ and an upper bound on $\omega(P)$.

To obtain an upper bound on $\omega(P)$, we recall the problem (QP) presented in Subsection 3.2. We relax the first set of constraints and add them to the objective function as

a penalty with Lagrangian multiplier vector $\boldsymbol{\lambda} \in \mathbb{R}^n$, and obtain the following Lagrangian relaxation problem:

$$(LRQP(\boldsymbol{\lambda})): \left| \begin{array}{lll} \text{maximize} & \displaystyle\sum_{p\in T}\left(\frac{1}{m}\sum_{r\in E} x_{rp} - \frac{1}{4m^2}\left(\sum_{i\in V} d_i y_{ip}\right)^2\right) + \sum_{i\in V}\lambda_i\left(1 - \sum_{p\in T} y_{ip}\right) & \\ \text{subject to} & x_{rp} \le y_{ip} & (r=\{i,j\}\in E, p\in T) \\ & x_{rp} \le y_{jp} & (r=\{i,j\}\in E, p\in T) \\ & 0 \le x_{rp} \le 1 & (r\in E, p\in T) \\ & y_{ip} \in \{0,1\} & (i\in V, p\in T). \end{array} \right.$$

Since the problem $(LRQP(\boldsymbol{\lambda}))$ is a relaxation problem of (P), the optimal value of $(LRQP(\boldsymbol{\lambda}))$ provides an upper bound on $\omega(P)$ for any $\boldsymbol{\lambda} \in \mathbb{R}^n$. Owing to the absence of the first set of constraints of the problem (QP), the problem $(LRQP(\boldsymbol{\lambda}))$ is decomposable into t subproblems. Let us denote the subproblem corresponding to $p \in T$ by $(LRQP(\boldsymbol{\lambda}, p))$, then the subproblem $(LRQP(\boldsymbol{\lambda}, p))$ is as follows:

$$(LRQP(\boldsymbol{\lambda}, p)): \left| \begin{array}{lll} \text{maximize} & \displaystyle\frac{1}{m}\sum_{r\in E} x_{rp} - \frac{1}{4m^2}\left(\sum_{i\in V} d_i y_{ip}\right)^2 - \sum_{i\in V}\lambda_i y_{ip} & \\ \text{subject to} & x_{rp} \le y_{ip} & (r=\{i,j\}\in E) \\ & x_{rp} \le y_{jp} & (r=\{i,j\}\in E) \\ & 0 \le x_{rp} \le 1 & (r\in E) \\ & y_{ip} \in \{0,1\} & (i\in V). \end{array} \right.$$

The optimal value of $(LRQP(\boldsymbol{\lambda}))$ is given as follows via the optimal value $\omega(LRQP(\boldsymbol{\lambda}, p))$.

$$\begin{aligned} \omega(LRQP(\boldsymbol{\lambda})) &= \sum_{p\in T} \max_{\boldsymbol{x},\boldsymbol{y}\in\mathcal{F}(LRQP(\boldsymbol{\lambda},p))} \left\{\frac{1}{m}\sum_{r\in E} x_{rp} - \frac{1}{4m^2}\left(\sum_{i\in V} d_i y_{ip}\right)^2 - \sum_{i\in V}\lambda_i y_{ip}\right\} + \sum_{i\in V}\lambda_i \\ &= \sum_{p\in T}\omega(LRQP(\boldsymbol{\lambda}, p)) + \sum_{i\in V}\lambda_i. \end{aligned}$$

Note that each subproblem $(LRQP(\boldsymbol{\lambda}, p))$ has the same optimal value regardless of index $p \in T$ and that $(LRQP(\boldsymbol{\lambda}, p))$ is equivalent to $(SP(\boldsymbol{\lambda}))$. Therefore we have the following equation:

$$\omega(LRQP(\boldsymbol{\lambda})) = t \cdot \omega(SP(\boldsymbol{\lambda})) + \sum_{i\in V}\lambda_i. \tag{5.4}$$

Proposition 5.2. *Let t be an upper bound on the number of communities at an optimal solution of (P). Then (5.4) is an upper bound on $\omega(P)$ for any $\boldsymbol{\lambda} \in \mathbb{R}^n$.*

Proof. Clear from the above discussion. □

Due to arbitrariness of $\boldsymbol{\lambda} \in \mathbb{R}^n$ in Proposition 5.2, the proposition holds for an optimal solution $\boldsymbol{\lambda}(\mathcal{C})$ obtained at each iteration of the algorithm. Thus we can obtain an upper bound on $\omega(P)$ at every iteration without additional computation. If the difference between the upper bound and $\omega(RD(\mathcal{C}))$ is small, we can stop the algorithm even if $\omega(SP(\mathcal{C})) \le 0$ does not hold. For a predetermined parameter $\varepsilon \ge 0$, we use the following condition as one of the stopping criterion of our algorithm

$$\frac{\mathrm{UB} - \omega(RD(\mathcal{C}))}{\mathrm{UB}} \le \varepsilon,$$

where UB is the smallest upper bound obtained so far. Note that the above condition implies $(\omega(P) - \omega(RD(\mathcal{C})))/\omega(P) \leq \varepsilon$ due to $\omega(P) \leq \mathrm{UB}$, that is, $\omega(RD(\mathcal{C}))$ is sufficiently close to the optimal value $\omega(P)$.

Here we describe the heuristic algorithm to obtain a feasible solution of the problem (P), which is based on a simple rounding procedure. First, we derive an optimal solution $\boldsymbol{z}(\mathcal{C})$ of $(RP(\mathcal{C}))$ from the optimal dual solution $\boldsymbol{\lambda}(\mathcal{C})$ obtained at Step 1 of the cutting plane algorithm. Since this primal solution is usually a fractional solution, we construct an integer solution $\bar{\boldsymbol{z}} = (\bar{z}_C)_{C \in \mathcal{C}}$ by rounding $\bar{z}_C = 1$ if $z_C(\mathcal{C}) > 0$, and $\bar{z}_C = 0$ otherwise. We say that $\{C_1, C_2, \ldots, C_k\}$ is a *cover* of V if $V = \bigcup_{p=1}^{k} C_p$ and $C_p \neq \emptyset$ for any $p \in \{1, 2, \ldots, k\}$, and let $\mathcal{S}$ denote a sub-family represented by the solution $\bar{\boldsymbol{z}}$ for succinct notation. The sub-family $\mathcal{S}$ is not necessarily a partition of V, but a cover of V due to the feasibility of $\boldsymbol{z}(\mathcal{C})$ for the problem $(RP(\mathcal{C}))$. Thus some communities may overlap with each other. For a given $\mathcal{S}$, we define the following set

$$M(\mathcal{S}, i) = \{\, C \in \mathcal{S} \mid i \in C \,\}$$

in order to check whether $\mathcal{S}$ is a partition of V. When the cardinality of $M(\mathcal{S}, i)$ is equal to 1 for any $i \in V$, the sub-family $\mathcal{S}$ is a partition of V, hence we can obtain a lower bound on $\omega(P)$. If $|M(\mathcal{S}, i)| > 1$ holds for some $i \in V$, then we compute the variation $\Delta(i, C)$ of the contribution f_C when node i is removed from a community C,

$$\Delta(i, C) = \frac{1}{m} \sum_{j \in C} w_{ij}$$

for each $C \in M(\mathcal{S}, i)$. Let C^* be the community that minimizes $\Delta(i, C)$ over $M(\mathcal{S}, i)$, and we remove the index i from any $C \in M(\mathcal{S}, i) \setminus C^*$. The rounding heuristics is described as follows.

Rounding Heuristics

`Step 0`

Let $\boldsymbol{z}(\mathcal{C})$ be an optimal solution of $(RP(\mathcal{C}))$.

`Step 1`

Construct $\bar{\boldsymbol{z}}$ from $\boldsymbol{z}(\mathcal{C})$ by the rounding procedure.
Let $\mathcal{S}$ be a sub-family represented by $\bar{\boldsymbol{z}}$.

`Step 2`

`if` $|M(\mathcal{S}, i)| = 1$ for any $i \in V$ `then`
Calculate f_C for any $C \in \mathcal{S}$, and set $\ell(P) \leftarrow \sum_{C \in \mathcal{S}} f_C$.
Output $\ell(P)$ and terminate.
`else`
Go to `Step 3`.
`end if`

`Step 3`

Select a minimum index $i \in V$ such that $|M(\mathcal{S}, i)| > 1$.
Compute $\Delta(i, C)$ for any $C \in M(\mathcal{S}, i)$, and set $C^* \leftarrow \mathrm{argmin}\{\, \Delta(i, C) \mid C \in M(\mathcal{S}, i) \,\}$.
`for` $C \in M(\mathcal{S}, i) \setminus C^*$ `do`
Set $C \leftarrow C \setminus \{i\}$.

`Step 4`

Update the sub-family $\mathcal{S}$ according to the modification at `Step 3`,
and return to `Step 2`.

Our proposed algorithm is described as follows. Note that $\mathcal{C}$ is incremented by a single set C determined by $\boldsymbol{y}^*$ found in Step 2. We will call this algorithm the *Single-Cutting-Plane-at-a-Time Algorithm*, *SCP* for short.

Single-Cutting-Plane-at-a-Time Algorithm (SCP)

`Step 0`

Let $\mathcal{C}$ be an initial family of nonempty subsets of V and ε be a parameter.

Initialize an upper bound UB and a lower bound LB by setting UB $\leftarrow 1$ and LB $\leftarrow 0$.

`Step 1`

Solve $(RD(\mathcal{C}))$ to obtain an optimal solution $\boldsymbol{\lambda}(\mathcal{C})$ and the optimal value $\omega(RD(\mathcal{C}))$.

Compute a lower bound $\ell(P)$ by the rounding heuristics.

`if` LB $< \ell(P)$ `then` LB $\leftarrow \ell(P)$.

`Step 2`

Solve $(SP(\boldsymbol{\lambda}(\mathcal{C})))$ and set $\boldsymbol{y}^*$ be an optimal solution.

Compute an upper bound $u(P)$ according to Proposition 5.2.

`if` UB $> u(P)$ `then` UB $\leftarrow u(P)$.

`Step 3`

`if` $\omega(SP(\boldsymbol{\lambda}(\mathcal{C}))) \leq 0$ `or` $(\mathrm{UB} - \omega(RD(\mathcal{C})))/\mathrm{UB} \leq \varepsilon$ `then`

Set $\mathcal{C}^\star \leftarrow \mathcal{C}$, $\omega^\star \leftarrow \omega(RD(\mathcal{C}))$, $\mathrm{LB}^\star \leftarrow$ LB and $\mathrm{UB}^\star \leftarrow$ UB.

Output $\mathcal{C}^\star$, $\omega^\star$, $\mathrm{LB}^\star$, $\mathrm{UB}^\star$, and terminate.

`else`

Set $C \leftarrow \{\, i \in V \mid y^*_i = 1 \,\}$, increment $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$. Go to `Step 1`.

`end if`

Carrying out some preliminary experiments by SCP, we frequently observed that the optimal value $\omega(RD(\mathcal{C}))$ stays constant for many iterations even when $\mathcal{C}$ is repeatedly incremented. We show in Table 1 and Figure 1 how slowly $\omega(RD(\mathcal{C}))$ increases. Here "Karate" is Zachary's karate dataset [34] representing friendship relation between members of a karate club. The slow convergence we observed may arise from a particular structure of $(RD(\mathcal{C}))$

Table 1: Plateau situation of SCP

Karate	
iteration	$\omega(RD(\mathcal{C}))$
0	0.00000
10	0.37179
20	0.37179
30	0.37179
40	0.38047
50	0.41979

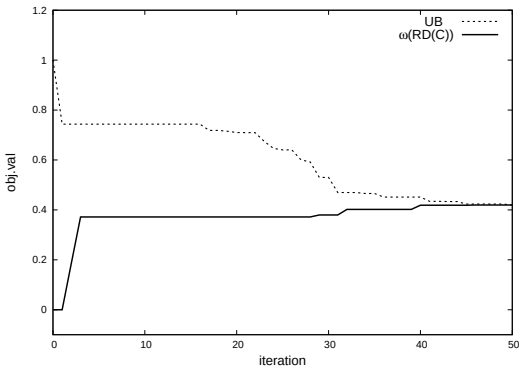

Figure 1: Behavior of SCP for Karate

that all coefficients of the objective function are one and all coefficients of the constraints are either zero or one. This makes the contour of the objective function and some face of

$\mathcal{F}(RD(\mathcal{C}))$ be parallel, and the whole face be optimal. As a consequence, the optimal value $\omega(RD(\mathcal{C}))$ stays constant although lots of cutting planes are added. To cut off such a face entirely, we propose to simultaneously add multiple cutting planes which may complement well each other. The first cutting plane is the same as the one defined by $\boldsymbol{y}^*$ and f^* obtained from problem $(SP(\boldsymbol{\lambda}(\mathcal{C})))$. We then fix the variables y_i to zero for all i with $y_i^* = 1$, and consider $(SP(\boldsymbol{\lambda}(\mathcal{C})))$. More precisely, we let $V^{(1)} = \{\, i \in V \mid y_i^* = 1 \,\}$ and approximately solve the problem

$$(SP(\boldsymbol{\lambda}(\mathcal{C}), V^{(1)})): \left| \begin{array}{lll} \text{maximize} & \dfrac{1}{m}\displaystyle\sum_{r\in E} x_r - \dfrac{1}{4m^2}\left(\sum_{i\in V} d_i y_i\right)^2 - \sum_{i\in V}\lambda_i(\mathcal{C}) y_i & \\ \text{subject to} & x_r \le y_i & (r = \{i,j\} \in E) \\ & x_r \le y_j & (r = \{i,j\} \in E) \\ & 0 \le x_r \le 1 & (r \in E) \\ & y_i \in \{0,1\} & (i \in V \setminus V^{(1)}) \\ & y_i = 0 & (i \in V^{(1)}) \end{array} \right.$$

to obtain $\boldsymbol{y}^{(1)}$ and $f^{(1)}$, i.e., the second cutting plane. In a general step, with $V^{(h)} = \{\, i \in V \mid y_i^{(l)} = 1 \text{ for some } l < h \,\}$ we approximately solve

$$(SP(\boldsymbol{\lambda}(\mathcal{C}), V^{(h)})): \left| \begin{array}{lll} \text{maximize} & \dfrac{1}{m}\displaystyle\sum_{r\in E} x_r - \dfrac{1}{4m^2}\left(\sum_{i\in V} d_i y_i\right)^2 - \sum_{i\in V}\lambda_i(\mathcal{C}) y_i & \\ \text{subject to} & x_r \le y_i & (r = \{i,j\} \in E) \\ & x_r \le y_j & (r = \{i,j\} \in E) \\ & 0 \le x_r \le 1 & (r \in E) \\ & y_i \in \{0,1\} & (i \in V \setminus V^{(h)}) \\ & y_i = 0 & (i \in V^{(h)}), \end{array} \right.$$

where $\boldsymbol{y}^{(0)} = \boldsymbol{y}^*$ and $V^{(0)} = \emptyset$. As long as $\omega(SP(\boldsymbol{\lambda}(\mathcal{C}), V^{(h)}))$ is positive, we keep on generating cutting planes. When $\omega(SP(\boldsymbol{\lambda}(\mathcal{C}), V^{(h)}))$ becomes non-positive, we add all the cutting planes obtained so far to $\mathcal{C}$. The Step 3 of the algorithm SCP should be modified as follows. We will call the algorithm with this modification the *Multiple-Cutting-Planes-at-a-Time Algorithm*, *MCP* for short.

Step 3 of the Multiple-Cutting-Planes-at-a-Time Algorithm (MCP)

`Step 3`
`if` $\omega(SP(\boldsymbol{\lambda}(\mathcal{C}))) \le 0$ `or` $(\mathrm{UB} - \omega(RD(\mathcal{C})))/\mathrm{U}B \le \varepsilon$ `then`
  Set $\mathcal{C}^\star \leftarrow \mathcal{C}$, $\omega^\star \leftarrow \omega(RD(\mathcal{C}))$, $\mathrm{LB}^\star \leftarrow \mathrm{LB}$ and $\mathrm{UB}^\star \leftarrow \mathrm{UB}$.
  Output $\mathcal{C}^\star$, $\omega^\star$, $\mathrm{LB}^\star$, $\mathrm{UB}^\star$, and terminate.
`else`
  Generate cutting planes until $\omega(SP(\boldsymbol{\lambda}(\mathcal{C}), V^{(h)}))$ becomes non-positive.
  Add all the cutting planes generated to $\mathcal{C}$. Go to `Step 1`.
`end if`

6. Computational Experiments

In this section, we report the computational experiments with the algorithms SCP and MCP. The experiments were performed on a computer with an Intel Core i7, 3.70 GHz

processor and 32.0 GB of memory. We implemented the algorithm in Python 2.7, and used the barrier method in Gurobi 6.0.0 as the LP solver. In the experiments, we used ten instances; Michael's Strike dataset [22], Zachary's Karate dataset [34], Gil-Mendieta and Schmidt's Mexico dataset [15], Michael and Massey's Sawmill dataset [23], Lusseau's Dolphins dataset [21], Hugo's Les Misérables dataset [19], Krebs' Books dataset [20], Girvan and Newman's Football dataset [16], U.S.airport (USAir97) dataset [35], and Electronic Circuit (s838) dataset [24]. The size, the known optimal value $\omega(P)$ and the optimal number of communities t^* of each instance are given in Table 2. We set $\mathcal{C}$ initially to the family

Table 2: Solved instances

ID	name	n	m	$\omega(P)$	t^*
1	Strike	24	38	0.56198	4
2	Karate	34	78	0.41979	4
3	Mexico	35	117	0.35952	4
4	Sawmill	36	62	0.55007	4
5	Dolphins	62	159	0.52852	5
6	Les Misérables	77	254	0.56001	6
7	Books	105	441	0.52724	4
8	Football	115	613	0.60457	10
9	USAir97	332	2126	0.36820	6
10	s838	512	819	0.81940	12

of all singleton, i.e., $\mathcal{C} = \{\{1\}, \{2\}, \ldots, \{n\}\}$, and the tolerance parameter ε to 0.03. We set the parameter t to the optimal number of communities t^*, which is usually unknown in advance. This setting may provide a favorable condition for our algorithm. The statistics collected are given in Table 3, and Table 4 shows the results of both algorithms SCP and MCP for each instance. The symbol "*" in the columns "peg" and "time 2" represents that the phase of solving the problem $(P(\mathcal{C}^\star))$ is not executed since the solution obtained by the cutting plane algorithm has already satisfied the integrality. The symbol "OT" in the column "time 1" means that the cutting plane algorithm does not terminate after more than $604,800$ seconds, i.e., seven days.

Table 3: Statistics

iter.	the number of solving the problem $(RD(\mathcal{C}))$
$\lvert\mathcal{C}^\star\rvert$	cardinality of the final family of subsets $\mathcal{C}^\star$
$\mathrm{LB}^\star$	lower bound obtained at the end of algorithm
$\omega(RD(\mathcal{C}^\star))$	optimal value of $(RD(\mathcal{C}^\star))$ obtained at the end of the algorithm
$\mathrm{UB}^\star$	upper bound obtained at the end of algorithm
peg	the number of pegged variables
$\omega(P(\mathcal{C}^\star))$	optimal value of $(P(\mathcal{C}^\star))$
gap	relative gap defined by gap $= \left(\frac{\omega(P)-\omega(P(\mathcal{C}^\star))}{\omega(P)}\right) \times 100$
time 1	computation time of the cutting plane algorithm in seconds
time 2	computation time of solving the problem $(P(\mathcal{C}^\star))$ in seconds

From Table 4, we observe that the number of generated constraints is much smaller than that of the original problem in both algorithms. Take instance Karate (ID=2) with 34

Table 4: Computational results of SCP and MCP

ID	iter.		$\lvert\mathcal{C}^\star\rvert$		LB*		$\omega(RD(\mathcal{C}^\star))$		UB*	
	SCP	MCP	SCP	MCP	SCP	MCP	SCP	MCP	SCP	MCP
1	29	14	53	60	0.55574	0.56198	0.55574	0.56198	0.56872	0.56758
2	45	20	79	77	0.41880	0.41979	0.41880	0.41979	0.42351	0.42639
3	50	22	85	87	0.34062	0.35952	0.35621	0.35952	0.36637	0.36454
4	34	16	70	81	0.55007	0.55007	0.55007	0.55007	0.56293	0.55737
5	86	32	148	159	0.52632	0.52760	0.52679	0.52760	0.54308	0.53869
6	112	28	189	161	0.54822	0.55908	0.55607	0.55908	0.57105	0.57243
7	107	52	212	242	0.52037	0.51797	0.52183	0.52344	0.53766	0.53831
8	471	47	586	238	0.60440	0.60457	0.60440	0.60457	0.62113	0.60457
9	NA	518	NA	2247	NA	0.35080	NA	0.36680	NA	0.37814
10	NA	1169	NA	9456	NA	0.81676	NA	0.81844	NA	0.84031
ID	peg		$\omega(P(\mathcal{C}^\star))$		gap (%)		time 1 (s)		time 2 (s)	
	SCP	MCP	SCP	MCP	SCP	MCP	SCP	MCP	SCP	MCP
1	*	*	0.55574	0.56198	1.10912	0.00000	3.94	1.82	*	*
2	*	*	0.41880	0.41979	0.23501	0.00000	6.25	3.43	*	*
3	27	*	0.35207	0.35952	2.07247	0.00000	28.19	13.56	0.22	*
4	*	*	0.55007	0.55007	0.00000	0.00000	4.99	2.88	*	*
5	136	*	0.52632	0.52760	0.41547	0.17224	37.84	16.60	0.07	*
6	105	*	0.55242	0.55908	1.35375	0.16497	75.56	22.51	0.68	*
7	185	62	0.52036	0.52262	1.30455	0.87544	144.79	73.07	0.32	2.20
8	*	*	0.60440	0.60457	0.02692	0.00000	3802.60	218.20	*	*
9	NA	157	NA	0.36477	NA	0.93048	OT	398900.90	NA	152.77
10	NA	7127	NA	0.81722	NA	0.26537	OT	37473.13	NA	312.15

nodes for example, the generated constraints are less than $1/10^8$ of the original constraints totaling 1.7×10^{10}. To compare SCP with MCP, we see that the number of iterations and the computation time of MCP are much smaller than those of SCP. Especially, for the instance Football (ID=8), the number of iterations (resp., the computation time) is reduced by a factor of approximately 10.02 (resp., 17.44). We also see that MCP solves the instances of Strike (ID=1), Karate (ID=2), Mexico (ID=3), Sawmill (ID=4) and Football (ID=8) to optimality, whereas SCP only solves the instance Sawmill (ID=4). Regarding the accuracy of the obtained solution, for the instances that SCP (resp., MCP) failed to solve, the remaining gap was less than approximately 2% (resp., 1%). From these observations, the algorithm MCP is superior to SCP in terms of both computation time and accuracy of the solution.

To compare the performance of several existing heuristics and our proposed algorithm, we give the lower bounds on the modularity obtained by several heuristics as well as $\omega(P(\mathcal{C}^\star))$ of MCP in Table 5. The columns GN, CNM, SD, CHL and NR represent the Girvan-Newman algorithm [16], the hierarchical agglomerative method by Clauset *et al.* [8], the Newman's spectral divisive method [26], the divisive method by Cafieri *et al.* [5], and the Noack and Rotta heuristics [28], respectively. Note that the largest lower bounds for each instance are bold-faced.

Table 5: Lower bounds by the existing heuristics and MCP

ID	GN	CNM	SD	CHL	NR	MCP
1	NA	NA	NA	NA	NA	**0.56198**
2	0.401	0.38067	0.39341	0.41880	**0.41979**	**0.41979**
3	NA	NA	NA	NA	NA	**0.35952**
4	NA	NA	NA	NA	NA	**0.55007**
5	0.520	0.49549	0.49120	0.52646	0.52377	**0.52760**
6	0.540	0.50060	0.51383	0.54676	**0.56001**	0.55908
7	NA	0.50197	0.46718	0.52629	**0.52694**	0.52262
8	0.601	0.57728	0.49261	0.60091	0.60028	**0.60457**
9	NA	0.32039	0.31666	0.35959	**0.36577**	0.36477
10	NA	0.80556	0.73392	0.81663	0.81624	**0.81722**

From Table 5, we confirm that the algorithm MCP attains the best result for seven instances out of whole of instances. Note that, for the three instances which have not been tested by any existing heuristics, MCP finds the optimal values. For other three instances whose best results are obtained by the algorithm NR, the lower bounds obtained by MCP are quite close to those obtained by the algorithm NR. To be specific, the difference (resp., gap) between them is at most 0.00432 (resp., 0.82%).

7. Conclusion

In this paper, we proposed the cutting plane algorithms for the modularity maximization problem. One of the advantages of the algorithms is that they are able to provide the upper bounds on the optimal modularity at every iteration by solving a small relaxation problem of the original problem. This bounding technique enables us to evaluate the quality of the objective value of the small relaxation problem at every iteration of the algorithms.

The key point in developing a good algorithm for the modularity maximization problems would be generating deep cutting planes. The method of multiple cutting planes that we

proposed in this paper performed fairly well, and this method could apply to other clustering problems of finding a partition of a given set. However it should need further investigation from both theoretical and computational view points.

A direction of further research is to incorporate a heuristics providing an initial partition into our algorithm. The exact algorithm [2] first finds a partition by Noack and Rotta's heuristics [28] before starting the column generation process, and uses members of the partition as an initial set of columns. This scheme would be expected to reduce the number of iterations of our algorithm. Moreover the instances we solved are so limited that further experiments should be carried out.

Acknowledgements

The authors thank anonymous referees for their careful reading and valuable comments on the earlier version of this paper.

References

[1] G. Agarwal and D. Kempe: Modularity-maximizing graph communities via mathematical programming. *The European Physical Journal B*, **66** (2008), 409–418.

[2] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, L. Liberti, and S. Pellon: Column generation algorithms for exact modularity maximization in networks. *Physical Review E*, **82** (2010), 046112.

[3] M.A. Boschetti, A. Mingozzi, and S. Ricciardelli: A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, **5** (2008), 735–747.

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

[5] S. Cafieri, P. Hansen, and L. Liberti: Locally optimal heuristic for modularity maximization of networks. *Physical Review E*, **83** (2011), 056105.

[6] A. Caprara, M. Fischetti, and P. Toth: Heuristic method for the set covering problem. *Operations Research*, **47** (1999), 730–743.

[7] A. Caprara, P. Toth, and M. Fischetti: Algorithms for the set covering problem. *Annals of Operations Research*, **98** (2000), 353–371.

[8] A. Clauset, M.E. Newman, and C. Moore: Finding community structure in very large networks. *Physical Review E*, **70** (2004), 066111.

[9] M. Conforti, G. Cornuejols, and G. Zambelli: *Integer Programming* (Springer, 2014).

[10] G. Desaulniers, J. Desrosiers, and M.M. Solomon: *Column Generation* (Springer, 2005).

[11] C.H.Q. Ding, X. He, H. Zha, M. Gu, and H.D. Simon: A min-max cut algorithm for graph partitioning and data clustering. In N. Cercone, T.Y. Lin, and X. Wu (eds.): *Proceedings 2001 IEEE International Conference on Data Mining*, (2001), 107–114.

[12] O. Du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen: Stabilized column generation. *Discrete Mathematics*, **194** (1999), 229–237.

[13] M.L. Fisher: The Lagrangian relaxation method for solving integer programming problems. *Management Science*, **27** (1981), 1–18.

[14] A.M. Geoffrion: Lagrangean relaxation for integer programming. *Mathematical Programming Studies*, **2** (1974), 82–114.

[15] J. Gil-Mendieta and S. Schmidt: The political network in Mexico. *Social Networks*, **18** (1996), 355–381.

[16] M. Girvan and M.E. Newman: Community structure in social and biological networks. *Proceedings of the National Academy of Sciences U.S.A.*, **99** (2002), 7821–7826.

[17] M. Grötschel and Y. Wakabayashi: A cutting plane algorithm for a clustering problem. *Mathematical Programming*, **45** (1989), 59–96.

[18] R. Guimerá and L.A.N. Amaral: Functional cartography of complex metabolic networks. *Nature*, **433** (2005), 895–900.

[19] D.E. Knuth: *The Stanford GraphBase: A Platform for Combinatorial Computing* (Addison-Wesley Reading, 1993).

[20] V. Krebs: `http://www.orgnet.com/`(last visited April 19, 2016.).

[21] D. Lusseau, K. Schneider, O.J. Boisseau, P. Haase, E. Slooten, and S.M. Dawson: The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, **54** (2003), 396–405.

[22] J.H. Michael: Labor dispute reconciliation in a forest products manufacturing facility. *Forest Products Journal*, **47** (1997), 41–45.

[23] J.H. Michael and J.G. Massey: Modeling the communication network in a sawmill. *Forest Products Journal*, **47** (1997), 25–30.

[24] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, U. Alon: Superfamilies of evolved and designed networks. *Science*, **303** (2004), 1538–1542.

[25] M.E. Newman: Fast algorithm for detecting community structure in networks. *Physical Review E*, **69** (2004), 066133.

[26] M.E. Newman: Modularity and community structure in networks. *Proceedings of the National Academy of Sciences U.S.A.*, **103** (2006), 8577–8582.

[27] M.E. Newman and M. Girvan: Finding and evaluating community structure in networks. *Physical Review E*, **69** (2004), 026113.

[28] A. Noack and R. Rotta: Multi-level algorithms for modularity clustering. In J. Vahrenhold (eds.): *Proceedings of the 8th International Symposium on Experimental Algorithms, Lecture Notes in Computer Science*, **5526** (2009), 257–268.

[29] C.R. Reeves: *Modern Heuristic Techniques for Combinatorial Problem* (Blackwell Scientific Publications, 1993).

[30] J. Shi and J. Malik: Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22** (2000), 888–905.

[31] S. Umetani and M. Yagiura: Relaxation heuristics for the set covering problem. *Journal of the Operations Research Society of Japan*, **50** (2007), 350–375.

[32] Y-C. Wei and C-K. Cheng: Ratio cut partitioning for hierarchical designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **10** (1991), 911–921.

[33] G. Xu, S. Tsoka, and L.G. Papageorgiou: Finding community structures in complex networks using mixed integer optimization. *The European Physical Journal B*, **60** (2007), 231–239.

[34] W.W. Zachary: An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, **33** (1977), 452–473.

[35] `http://vlado.fmf.uni-lj.si/pub/networks/data/` (last visited April 19, 2016.).

Yoichi Izunaga
Information Systems Research Division

The Institute of Behavioral Sciences
2-9 Ichigayahonmura-cho, Shinjuku-ku
Tokyo 162-0845, Japan
E-mail: yizunaga@ibs.or.jp