A CUTTING PLANE ALGORITHM FOR MODULARITY MAXIMIZATION WITH HEURISTICS FOR SEPARATION PROBLEM

YOICHI IZUNAGA AND YOSHITSUGU YAMAMOTO

Abstract. The 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 which may complement well each other.

1. Introduction

As social network services grow, clustering on graphs has been attracting more attention, and since Newman and Girvan [11] proposed the modularity as a graph clustering measure, modularity maximization problem became one of the central subjects of research. The *NP*-hardness of the modularity maximization problem shown by Brandes *et al.* [3] turned researchers' attention to heuristic algorithms, which resulted in several efficient heuristic algorithms such as the linear programming with rounding procedure [1] and the variable neighborhood search [10].

On the other hand, among exact algorithms two 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 [9]. In this formulation a binary variable corresponding to each pair of nodes represents whether the two nodes belong to the same cluster. 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 [9].

The second formulation is 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 non-convex 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.* [8] have proposed an acceleration method, called stabilized column generation method. Some computational results are reported also in Aloise *et al.* [2].

In this paper, based on the set partitioning formulation, we propose a cutting plane algorithm. Finding a "best" cutting plane in a sense is *NP*-hard, so that we propose to use a heuristic

Date: July 20, 2013.

Key words and phrases. Modularity maximization, Set partitioning, Clique partitioning, Cutting planes, Integer programming.

The second author is grateful to the Okawa Foundation for Information and Telecommunication for their financial support.

Department of Social Systems and Management Discussion Paper Series No.1309.

algorithm to find a cutting plane to add. We also propose a method to find multiple cutting planes to accelerate the convergence, and report on some computational experiments.

This paper is organized as follows. In Section 2, we give the definition of modularity and formulate the modularity maximization problem as a set partitioning problem. In Section 3, we introduce the relaxation problem and its linear programming dual problem. In Section 4, after reviewing cutting plane algorithms, we propose two versions of the cutting plane algorithm with heuristics for generating a cutting plane. In Section 5, we explain a local search algorithm which is used in our proposed algorithm. In Section 6, we report the computational experiments of the proposed algorithm. Finally in Section 7 we give some conclusions.

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 undirected m 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_p and the other end-node in C_q by $E(C_p, C_q)$. When $C_p = C_q$ we write $E(C_p, C_q)$ simply as $E(C_p)$.

Modularity, denoted by $\mu(\Pi)$, of a partition Π is defined as

$$\mu(\Pi) = \sum_{p=1}^{k} \left(\frac{|E(C_p)|}{m} - \left(\frac{|E(C_p)| + \sum_{q=1}^{k} |E(C_p, C_q)|}{2m} \right)^2 \right),$$

where $|\cdot|$ denotes the cardinality of the corresponding set. See Newman and Girvan [11] and Brandes *et al.* [3]. 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.,

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

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

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

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

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

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

$$f_C = \frac{1}{2m} \sum_{i \in C} \sum_{j \in C} w_{ij}, \tag{2.1}$$

which is rewritten as

$$\begin{aligned} &= \frac{1}{2m} \sum_{i \in V} \sum_{j \in V} w_{ij} a_{iC} a_{jC} \\ &= \frac{1}{m} \left(\sum_{i \in V} \sum_{j \in V : i < j} w_{ij} a_{iC} a_{jC} + \sum_{i \in V} w_{ii} a_{iC} \right). \end{aligned}$$

The last equality is due to the symmetry of w_{ij}. 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:

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

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

3. Relaxation problem and its dual problem

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 [4, 5, 12, 13]. 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

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

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

$$RD: \quad \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i \in V} \lambda_i + \frac{1}{m} \sum_{i \in V} w_{ii} & \\ \text{subject to} & \displaystyle\sum_{i \in V} a_{iC} \lambda_i \geq f_C & (\forall C \in \mathcal{P}) \\ & \lambda_i \in \mathbb{R} & (\forall i \in V). \end{array} \right.$$

Note that $f_C = 0$ when C is a singleton from (2.1). Thus n of inequality constraints imply the non-negativity of λ_i's. Therefore the problem RD is equivalent to the problem with non-negativity constraints of variables:

$$RD: \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i\in V}\lambda_i + \frac{1}{m}\sum_{i\in V} w_{ii} & \\ \text{subject to} & \displaystyle\sum_{i\in V} a_{iC}\lambda_i \geq f_C & (\forall C \in \mathcal{P}) \\ & \lambda_i \geq 0 & (\forall i \in V). \end{array} \right.$$

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 of $\omega(P)$. 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 RD. Although RD has only n variables, its exponentially large number of constraints makes it intractable.

4. Cutting plane algorithm with heuristics

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.

4.1. 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:

$$RD(\mathcal{C}): \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i\in V}\lambda_i + \frac{1}{m}\sum_{i\in V} w_{ii} & \\ \text{subject to} & \displaystyle\sum_{i\in V} a_{iC}\lambda_i \geq f_C & (\forall C \in \mathcal{C}) \\ & \lambda_i \geq 0 & (\forall i \in V). \end{array} \right.$$

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 $\gamma_C(\mathcal{C})$ of the constraint corresponding to C as

$$\gamma_C(\mathcal{C}) = \sum_{i\in V} a_{iC}\lambda_i^*(\mathcal{C}) - f_C. \tag{4.1}$$

Note that $\gamma_C(\mathcal{C}) \geq 0$ for all $C \in \mathcal{C}$. When $\gamma_C(\mathcal{C}) \geq 0$ 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

$$\gamma_C(\mathcal{C}) < 0$$

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., $\omega(RD(\mathcal{C} \cup \{C\})) > \omega(RD(\mathcal{C}))$. Substituting (2.1) for f_C of (4.1) yields

$$\gamma_C(\mathcal{C}) = \sum_{i\in V} a_{iC}\lambda_i^*(\mathcal{C}) - \frac{1}{m}\sum_{i\in V}\sum_{j\in V;\, i<j} w_{ij}a_{iC}a_{jC},$$

hence the problem of minimizing $\gamma_C(\mathcal{C})$ over $\mathcal{P}$ is formulated as the problem $AP(\mathcal{C})$ with a quadratic objective function in binary variables:

$$AP(\mathcal{C}): \quad \left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i\in V} \lambda_i^*(\mathcal{C}) y_i - \frac{1}{m} \sum_{i\in V} \sum_{j\in V;\, i<j} w_{ij} y_i y_j \\ \text{subject to} & y_i \in \{0,1\} \qquad (\forall i \in V). \end{array} \right.$$

An optimal solution $\boldsymbol{y}^*$ of this problem provides the incidence vector of a community that minimizes $\gamma_C(\mathcal{C})$ over $\mathcal{P}$. Since $\boldsymbol{y} = \boldsymbol{0}$ is a feasible solution of this problem, the optimal value is non-positive. Finding $\boldsymbol{y}^*$ with negative 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 can be 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 $AP(\mathcal{C})$ and set $\boldsymbol{y}^*$ be an optimal solution.
`Step 3`
`If` $\omega(AP(\mathcal{C})) \geq 0$ `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`

The initial subfamily $\mathcal{C}$ could be empty, but a clever choice may enhance the efficiency of the algorithm. In our experiments, as reported in Section 6, 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 $C \in \mathcal{C}^\star$.

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

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, Xpress. Lacking variables z_C with C not in $\mathcal{C}^\star$, $P(\mathcal{C}^\star)$ provides a lower bound of $\omega(P)$. 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.

4.2. **Proposed algorithm.** Since problem $AP(\mathcal{C})$ to be solved in Step 2 of the prototype cutting plane algorithm is an *NP*-hard non-convex quadratic programming with binary variables, we propose to apply a heuristic algorithm. We put off the description of the heuristic algorithm until Section 5, and we first discuss how the stopping criterion should be modified accordingly.

Let $\alpha(AP(\mathcal{C}))$ denote the objective function value of a solution provided by a heuristic algorithm for $AP(\mathcal{C})$ in Step 2. The non-negativity of $\alpha(AP(\mathcal{C}))$ does not imply the non-negativity of $\omega(AP(\mathcal{C}))$, hence we cannot claim that we have reached a solution of RD even if $\alpha(AP(\mathcal{C})) \geq 0$ holds. Therefore, we propose to stop the algorithm when $\alpha(AP(\mathcal{C}))$ continues to be non-negative for a predetermined number of successive iterations. Our proposed algorithm is described as follows, where k_{max} is the predetermined number of iterations. 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.
Determine a natural number k_{max}, and set $k \leftarrow 0$.
`Step 1`
Solve $RD(\mathcal{C})$ to obtain an optimal solution $\boldsymbol{\lambda}^*(\mathcal{C})$ and the optimal value $\omega(RD(\mathcal{C}))$.
`Step 2`
Apply the heuristic algorithm to $AP(\mathcal{C})$ (see Section 5 for the algorithm).
Set $\boldsymbol{y}^*$ and $\alpha(AP(\mathcal{C}))$ be a solution obtained and its objective function value, respectively.
`Step 3`
`if` $\alpha(AP(\mathcal{C})) \geq 0$ `then`
$k \leftarrow k + 1$.
`else`
Set $C \leftarrow \{\, i \in V \mid y_i^* = 1 \,\}$, increment $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$, and set $k \leftarrow 0$. Go to `Step 1`.
`end if`
`Step 4`
`if` $k > k_{max}$ `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`
Go to `Step 2`.
`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 how slowly $\omega(RD(\mathcal{C}))$ increases as the algorithm *SCP* progresses. Concerning "Dolphins" and "Jazz" the optimal value $\omega(RD(\mathcal{C}))$ did not change at all after 500th iteration. Here "Dolphins", "Football" and "Jazz" are the benchmark instances available from the site:

`http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`

The size as well as the known optimal value $\omega(P)$ of each instance is given in Table 2.

The slow convergence we observed may arise from a particular structure of $RD(\mathcal{C})$ 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

TABLE 1. Plateau situation of *SCP*

iteration	$\omega(RD(\mathcal{C}))$		
	Dolphins	Football	Jazz
1	-0.0213	-0.0087	-0.0070
500	0.4241	0.4430	0.3276
1000	0.4241	0.4470	0.3276
2000	0.4241	0.4549	0.3276
3000	0.4241	0.4587	0.3276
4000	0.4241	0.4605	0.3276
5000	0.4241	0.4605	0.3276
6000	0.4241	0.4609	0.3276

TABLE 2. Solved instances

name	n	m	$\omega(P)$
Dolphins	62	159	0.5285
Football	115	613	0.6045
Jazz	198	2742	0.4448

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 $AP(\mathcal{C})$ in (4.1). We then fix the variables y_i to zero for all i with $y_i^* = 1$, and consider $AP(\mathcal{C})$. More precisely, we let $V^{(1)} = \{\, i \in V \mid y_i^* = 1 \,\}$ and approximately solve the problem

$$AP(\mathcal{C}, V^{(1)}): \quad \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i\in V} \lambda_i^*(\mathcal{C}) y_i - \frac{1}{m} \sum_{i\in V} \sum_{j\in V; i<j} w_{ij} y_i y_j & \\ \text{subject to} & y_i \in \{0,1\} & (\forall i \in V \setminus V^{(1)}) \\ & y_i = 0 & (\forall i \in V^{(1)}) \end{array} \right.$$

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

$$AP(\mathcal{C}, V^{(h)}): \quad \left| \begin{array}{lll} \text{minimize} & \displaystyle\sum_{i\in V} \lambda_i^*(\mathcal{C}) y_i - \frac{1}{m} \sum_{i\in V} \sum_{j\in V; i<j} w_{ij} y_i y_j & \\ \text{subject to} & y_i \in \{0,1\} & (\forall i \in V \setminus V^{(h)}) \\ & y_i = 0 & (\forall i \in V^{(h)}), \end{array} \right.$$

where $\boldsymbol{y}^{(0)} = \boldsymbol{y}^*$ and $V^{(0)} = \emptyset$. We call these problems *Restricted Auxiliary problems*. As long as $\alpha(AP(\mathcal{C}, V^{(h)}))$ is negative, we keep on generating cutting planes. When $\alpha(AP(\mathcal{C}, V^{(h)}))$ becomes non-negative, 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 α(AP(C)) ≥ 0 then
        k ← k + 1.
    else
```

Generate cutting planes until $\alpha(AP(\mathcal{C}, V^{(h)}))$ becomes non-negative.
Add all the cutting planes generated to $\mathcal{C}$. Set $k \leftarrow 0$ and go to `Step 1`.
`end if`

5. Heuristic algorithm based on the noising method

Now we discuss the heuristic algorithm for $AP(\mathcal{C})$ used in Step 2, which is based on the Noising method originally proposed by Charon and Hudry [6, 7].

We start with the explanation of the conventional local search. Given a feasible solution $\boldsymbol{y}$ of $AP(\mathcal{C})$, we define the neighborhood $N(\boldsymbol{y})$ of $\boldsymbol{y}$ as

$$N(\boldsymbol{y}) = \{\, \boldsymbol{y}' \mid \|\boldsymbol{y}' - \boldsymbol{y}\|_1 \leq 1 \,\}.$$

Then each vector in $N(\boldsymbol{y})$ is obtained by replacing a component, say y_q of $\boldsymbol{y}$ by its complement $1 - y_q$. We denote the thus obtained vector by $\boldsymbol{y}^q$. The variation, denoted by $v(\boldsymbol{y}, \boldsymbol{y}^q)$, of the objective function of $AP(\mathcal{C})$ when $\boldsymbol{y}$ changes to $\boldsymbol{y}^q$ is given as

$$v(\boldsymbol{y}, \boldsymbol{y}^q) = (1 - 2y_q) \left(\lambda_q^*(\mathcal{C}) - \frac{1}{m} \sum_{i \in V \setminus \{q\}} w_{iq} y_i \right).$$

The above variation (5) is computed for each solution in $N(\boldsymbol{y})$. If they are all non-negative, then the local search stops and return $\boldsymbol{y}$ as a local optimal solution. Otherwise, we move $\boldsymbol{y}$ to the solution which minimizes $v(\boldsymbol{y}, \boldsymbol{y}^q)$ over $N(\boldsymbol{y})$. The algorithm is described as follows.

Conventional Local Search

`Step 0`
Let $\boldsymbol{y}$ be an initial feasible solution of $AP(\mathcal{C})$.
`Step 1`
`for` $q = 1$ to n `do`
Compute $v(\boldsymbol{y}, \boldsymbol{y}^q)$.
`Step 2`
$v^* \leftarrow \min\{v(\boldsymbol{y}, \boldsymbol{y}^q)\}$ and $\boldsymbol{y}^* \leftarrow \operatorname{argmin}\{v(\boldsymbol{y}, \boldsymbol{y}^q)\}$.
`if` $v^* \geq 0$ `then`
Output $\boldsymbol{y}$, and terminate.
`else`
Set $\boldsymbol{y} \leftarrow \boldsymbol{y}^*$, and go to `Step 1`.
`end if`

The above naive local search is notorious for that it is liable to get trapped in a local optimal solution. To overcome this drawback Charon and Hudry [6, 7] proposed the *Noising method*, which is based on the local search but differs in that the objective function is perturbed by adding a random value, which they call *noise*.

First, we give an initial feasible solution $\boldsymbol{y}$, and find the best solution $\boldsymbol{y}^*$ in $N(\boldsymbol{y})$ by the local search. Then we calculate the following perturbed variation $\tilde{v}(\boldsymbol{y}, \boldsymbol{y}^*)$ by adding a number ρ randomly chosen from the interval $[-r, r]$:

$$\tilde{v}(\boldsymbol{y}, \boldsymbol{y}^*) = v(\boldsymbol{y}, \boldsymbol{y}^*) + \rho.$$

If $\tilde{v}(\boldsymbol{y}, \boldsymbol{y}^*)$ is negative, we move $\boldsymbol{y}$ to $\boldsymbol{y}^*$. The interval $[-r, r]$ is reduced to $[-r + d, r - d]$ after a predetermined number of iterations, and the Noising method terminates when the interval shrinks to $\{0\}$ or vanishes. Noising method is described as follows.

Noising Method

`Step 0`
Let $\boldsymbol{y}$ be an initial feasible solution.
Set $r \leftarrow 100$, $d \leftarrow 1$, FixedIter $\leftarrow 10$, and $t \leftarrow 0$.
`Step 1`
Set $t \leftarrow t + 1$ and call Local Search to obtain a local optimal solution $\boldsymbol{y}^*$ in $N(\boldsymbol{y})$.
`Step 2`
Draw the noise ρ randomly from the interval $[-r, r]$ and compute $\tilde{v}(\boldsymbol{y}, \boldsymbol{y}^*)$.
`if` $\tilde{v}(\boldsymbol{y}, \boldsymbol{y}^*) < 0$ `then` set $\boldsymbol{y} \leftarrow \boldsymbol{y}^*$.
`Step 3`
`if` $t \equiv 0$ (mod FixedIter) `then` set $r \leftarrow r - d$ and go to `Step 1`.
`Step 4`
`if` $r = 0$ `then`
Output $\boldsymbol{y}$ and terminate.
`else`
Go to `Step 1`.
`end if`

6. Computational experiments

We report the computational experiment with *MCP*. The experiment was performed on a PC with an Intel Core2 Duo, 3.06GHz processor and 4.0GB of memory. We implemented the algorithm in Java, and used CPLEX 12.3 as the LP solver, and solved the three benchmark instances introduced in Subsection 4.2.

Since the result may change due to the random noise ρ used in the algorithm, we executed the algorithm five times for each instance. We set $\mathcal{C}$ initially to the family of all singletons, i.e., $\mathcal{C} = \{\{1\}, \{2\}, \ldots, \{n\}\}$, and the parameter k_{max} to 30. The statistics collected are given in Table 3, and Table 4 shows the results of 15 executions.

Table 3. Statistics

$\lvert\mathcal{C}^\star\rvert$	cardinality of the final family of subsets $\mathcal{C}^\star$
$\omega(RD(\mathcal{C}^\star))$	optimal value of $RD(\mathcal{C}^\star)$ obtained at the end of the algorithm
$\omega(P(\mathcal{C}^\star))$	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	computation time in seconds

From Table 4, we see that *MCP* solved the instances of Dolphins and Football to optimality, although it does not provide a proof of optimality. At four out of five executions *MCP* failed to solve Jass, but the gap was less than 0.5%. Aloise *et al.* [2] reported that the stabilized column generation method solved Dolphins in approximately seven seconds on a PC with Intel Pentium, 3.20GHz processor and 3.0GB of memory, and CPLEX 10.110 as the LP solver. However they could not solve Football after the computation of more than 100,000 seconds.

TABLE 4. Computational results of *MCP*

instance	exec.	$\lvert\mathcal{C}^\star\rvert$	$\omega(RD(\mathcal{C}^\star))$	$\omega(P(\mathcal{C}^\star))$	gap (%)	time (s)
Dolphins	1	797	0.5285	0.5285	0.000	11
	2	982	0.5285	0.5285	0.000	23
	3	676	0.5285	0.5285	0.000	9
	4	1,081	0.5285	0.5285	0.000	23
	5	998	0.5285	0.5285	0.000	16
Football	1	5,617	0.6045	0.6045	0.000	326
	2	5,046	0.6045	0.6045	0.000	291
	3	5,248	0.6045	0.6045	0.000	312
	4	4,982	0.6045	0.6045	0.000	265
	5	5,029	0.6045	0.6045	0.000	294
Jazz	1	26,804	0.4439	0.4436	0.269	25,002
	2	26,890	0.4448	0.4448	0.000	19,780
	3	26,872	0.4445	0.4445	0.067	19,188
	4	26,509	0.4445	0.4445	0.067	21,238
	5	27,103	0.4434	0.4429	0.427	25,001

We also observe that the number of generated constraints $|\mathcal{C}^\star|$ is much smaller than that of the original problem. Take Dolphins with 62 nodes for instance, the generated constraints are less than $1/10^{15}$ of the original constraints totalling 4.6×10^{18}.

Figures 1, 2 and 3 provide a comparison of *SCP* with *MCP*. We stopped *SCP* when the number of cutting planes generated amounts to the same number of those generated by *MCP*. The figures show $\omega(RD(\mathcal{C}))$ vs. the number of iterations. For both algorithms $\omega(RD(\mathcal{C}))$ rapidly increases at an early stage, and then increases slowly or stays constant as the algorithm progresses.

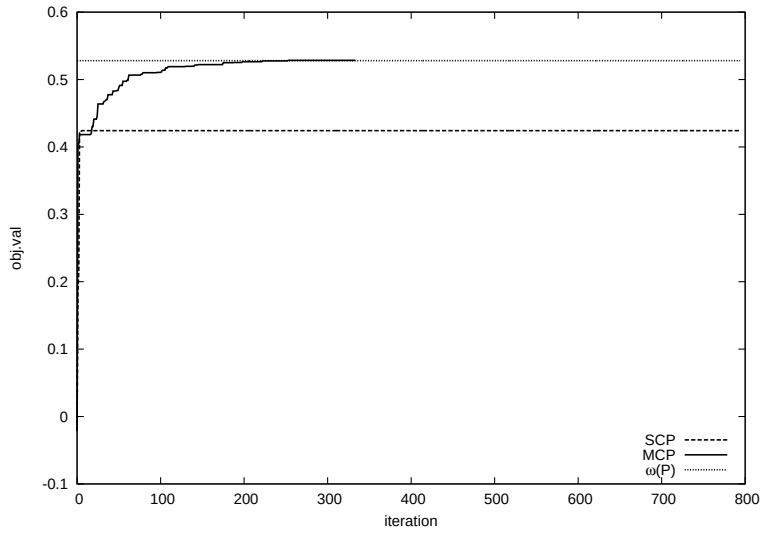

FIGURE 1. $\omega(RD(\mathcal{C}))$ vs. iterations for Dolphins

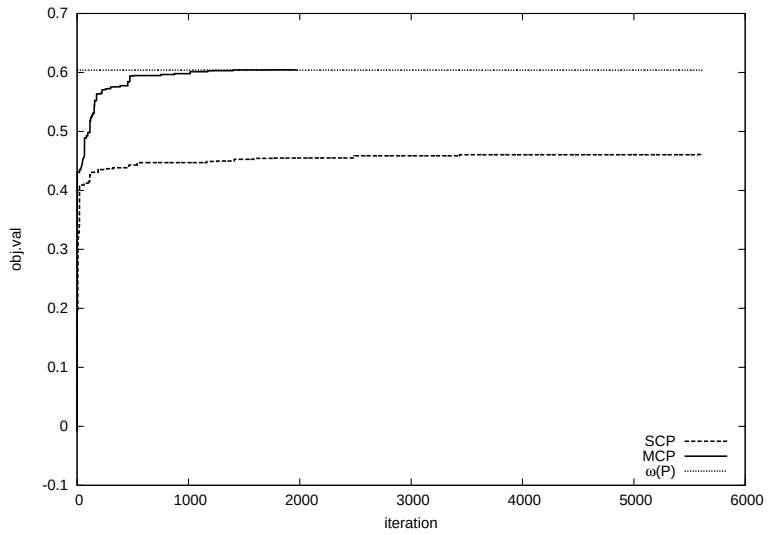

FIGURE 2. $\omega(RD(\mathcal{C}))$ vs. iterations for Football

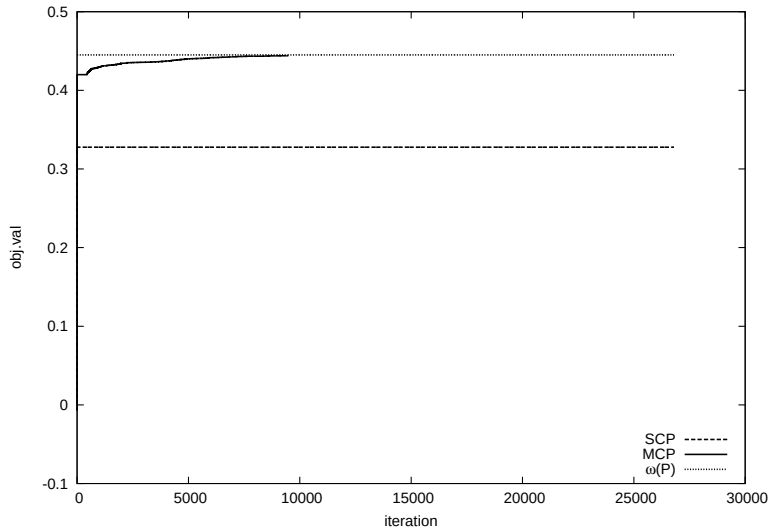

FIGURE 3. $\omega(RD(\mathcal{C}))$ vs. iterations for Jazz

7. Conclusion

One of the key points in developing a good algorithm for modularity maximization problems would be generating deep cutting planes. We proposed to solve the auxiliary problem $AP(\mathcal{C})$ by a heuristic algorithm based on the noising method in this paper, however here still remains room for further research. The method of multiple cutting planes that we proposed in this paper performed fairly well, however it should need further investigation from both theoretical and computational view points.

References

[1] G. Agarwal and D. Kempe, "Modularity-maximizing graph communities via mathematical programming," *The European Physical Journal*, B.66, pp.409-418, 2008.
[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, 2012.
[3] 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, pp.172-188, 2008.
[4] A. Caprara, M. Fischetti, and P. Toth, "Heuristic method for the set covering problem," *Operations Research*, 47, pp.730-743, 1999.
[5] A. Caprara, P. Toth, and M. Fischetti, "Algorithms for the set covering problem," *Annals of Operations Research*, 98, pp.353-371, 2000.
[6] I. Charon and O. Hudry, "Application of the noising method to the travelling salesman problem," *European Journal of Operations Research*, 125, pp.266-277, 2000.
[7] I. Charon and O. Hudry, "The noising methods: A generalization of some metaheuristics," *European Journal of Operations Research*, 135, pp.86-101, 2001.
[8] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, "Stabilized column generation," *Discrete Mathematics*, 194, pp.229-237, 1999.
[9] M. Grötschel and Y. Wakabayashi, "A cutting plane algorithm for a clustering problem," *Mathematical Programming*, 45, pp.59-96, 1989.
[10] P. Hansen and N. Mladenovic, "Variable neighborhood search: Principles and applications," *European Journl of Operational Research*, 130, pp.449-467, 2001.
[11] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review*, E.69, 2004.
[12] C. R. Reeves (ed.): *Modern heuristic techniques for combinatorial problem*. Blackwell Scientific Publications, 1993.
[13] S. Umetani and M. Yagiura, "Relaxation heuristics for the set covering problem," *Journal of the Operations Research Society of Japan*, 50, pp.350-375, 2007.

(Y. Izunaga) Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan
E-mail address: s1130131@sk.tsukuba.ac.jp

(Y. Yamamoto) Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan
E-mail address: yamamoto@sk.tsukuba.ac.jp