Eur. Phys. J. B (2013) 86: 302
DOI: 10.1140/epjb/e2013-40006-7

THE EUROPEAN PHYSICAL JOURNAL B

Regular Article

Computing an upper bound of modularity

Atsushi Miyauchi[1,a] and Yuichiro Miyamoto[2]

[1] Graduate School of Decision Science and Technology, Tokyo Institute of Technology, Ookayama 2-12-1, Meguro-ku, 152-8552 Tokyo, Japan
[2] Faculty of Science and Technology, Sophia University, Kioi-cho 7-1, Chiyoda-ku, 102-8554 Tokyo, Japan

Received 4 January 2013 / Received in final form 9 May 2013
Published online 1st July 2013 –

Abstract. Modularity proposed by Newman and Girvan is a quality function for community detection. Numerous heuristics for modularity maximization have been proposed because the problem is NP-hard. However, the accuracy of these heuristics has yet to be properly evaluated because computational experiments typically use large networks whose optimal modularity is unknown. In this study, we propose two powerful methods for computing a nontrivial upper bound of modularity. More precisely, our methods can obtain the optimal value of a linear programming relaxation of the standard integer linear programming for modularity maximization. The first method modifies the traditional row generation approach proposed by Grötschel and Wakabayashi to shorten the computation time. The second method is based on a row and column generation. In this method, we first solve a significantly small subproblem of the linear programming and iteratively add rows and columns. Owing to the speed and memory efficiency of these proposed methods, they are suitable for large networks. In particular, the second method consumes exceedingly small memory capacity, enabling us to compute the optimal value of the linear programming for the Power Grid network (consisting of 4941 vertices and 6594 edges) on a standard desktop computer.

1 Introduction

Many complex systems are represented as networks. The structures and dynamics of these networks contain meaningful information about the underlying systems. Indeed, network analysis is increasingly applied in diverse fields such as computer science, physics, chemistry, biology, and sociology.

In network analysis, it has been actively studied to divide a set of vertices into *dense* parts [1]. A dense part of a network is called a *community* (also known as a *module* or *cluster*) and the division of a network into communities is termed *community detection* (or *clustering*). Community detection analysis provides valuable insights into complex systems.

Let us consider an undirected graph $G = (V, E)$ consisting of $n = |V|$ vertices and $m = |E|$ edges. In 2004, Newman and Girvan [2] proposed *modularity* as a quality function for community detection. The argument of this function is a division $\mathcal{C} = \{V_1, V_2, \ldots, V_k\}$ of V. Note that the elements of $\mathcal{C}$ are nonempty and disjoint, and moreover satisfy $\bigcup_{i=1}^{k} V_i = V$. Modularity Q is written as:

$$Q(\mathcal{C}) = \frac{1}{2m} \sum_{i \in V} \sum_{j \in V} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \delta(C_i, C_j),$$

where A_{ij} is the $\{i, j\}$ component of the adjacency matrix of G, d_i is the degree of a vertex i, C_i is the community to which a vertex i belongs, and δ represents the Kronecker symbol equal to 1 if two arguments are the same and 0 otherwise. Modularity implies the sum, over all communities, of the number of edges in a community minus the expected number of such edges assuming that they are put at random with the same distribution of vertex degree. Modularity maximization is regarded as a very natural way to identify community structures in real-world networks [1]. Fortunato and Barthélemy [3] pointed out a resolution limit exists in modularity maximization, that is, when the number of edges is large, small communities tend to be undetected even if they are very dense. Nevertheless, community detection is now often conducted through modularity maximization: the input is an undirected graph $G = (V, E)$, and the goal is to find a division $\mathcal{C}$ that maximizes modularity. Modularity maximization forms the basis of the present study.

In 2008, Brandes et al. [4] proved that modularity maximization is NP-hard. In other words, no modularity maximization algorithm that simultaneously satisfies the following exists: (i) finds a division that maximizes modularity (ii) in time polynomial in n and m (iii) for any graphs, unless P = NP. A major current focus in modularity maximization is the development of excellent heuristics. To this end, numerous approaches based on greedy

[a] e-mail: miyauchi.a.aa@m.titech.ac.jp

technique [2,5,6], simulated annealing [7–9], extremal optimization [10], spectral optimization [11,12], dynamical clustering [13], multilevel partitioning [14], quantum mechanics [15], mathematical programming [16], and other techniques have been developed. Recently, Cafieri et al. [17] have proposed a post-processing approach to improve the solutions obtained by such heuristics.

Heuristics are evaluated by three measures: rapidity, memory efficiency, and accuracy. However, because the optimal modularity for large networks (commonly used in computational experiments) is unknown, the accuracy measure is imprecise. Although a few exact algorithms exist for modularity maximization [18–21], even the most efficient one [21] is applicable to only small networks with a maximum of 512 vertices.

In this study, we propose two powerful methods for computing a nontrivial upper bound of modularity. More precisely, our methods can obtain the optimal value of a linear programming relaxation of the standard integer linear programming for modularity maximization. From the analysis of DasGupta and Desai [22], the integrality gap of the linear programming is large for a certain class of graphs. However, consistent with Agarwal and Kempe [16], our computational experiments show that the optimal value of the linear programming is sufficiently close to the optimal modularity for many real-world networks.

Our first method is based on the traditional row generation proposed by Grötschel and Wakabayashi [18]. We present a simple implementation that drastically shorten the computation time. The second method is based on a row and column generation, in which we consider a subproblem of the linear programming with far fewer variables and constraints. This significantly small subproblem is solved by iterative additions of rows and columns. We specify a sufficient condition that an optimal value of the subproblem be equal to that of the original linear programming.

These proposed methods, being rapidly executed and memory-efficient, are applicable to large networks. In particular, the second method requires exceedingly small memory capacity. Consequently, we succeeded in computing the optimal value of the linear programming relaxation for the famous test network *Power Grid* [23], consisting of 4941 vertices and 6594 edges, on a standard desktop computer. To our knowledge, this value has not been previously determined. However, because a fractional optimal solution was yielded for this network, we cannot be certain that the upper bound equals the optimal modularity.

This paper is structured as follows. In Section 2, formulations for modularity maximization related to our methods are presented. In Section 3, we revisit the traditional row generation technique and introduce our improved version. The second proposed method, based on a row and column generation, is described in Section 4. Results of computational experiments are shown in Section 5. Section 6 concludes the study.

2 Formulations for modularity maximization

2.1 Natural formulation

As described in reference [16], modularity maximization can be formulated in terms of integer linear programming. Let $V = \{1, 2, \ldots, n\}$. We set

$$q_{ij} = A_{ij} - \frac{d_i d_j}{2m},$$

and introduce binary variables x_{ij} equal to 1 if vertices i and j belong to the same community and 0 otherwise. Then, the objective function to be maximized can be written as:

$$\frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_{ij}.$$

Since $x_{ii} = 1$ for all vertices i, the variables x_{ii} can be replaced by a constant, defined as:

$$C = \frac{1}{2m} \sum_{i=1}^{n} q_{ii},$$

which is added to the objective function. Moreover, $q_{ij} = q_{ji}$ and $x_{ij} = x_{ji}$ for all vertices i and j. Thus, the variables x_{ij} $(i > j)$ can be removed by doubling the objective function. Hence, in terms of the variables x_{ij} $(i < j)$, modularity maximization can be formulated as:

$$\begin{aligned}
(\mathrm{IP}) : \max. \ & \frac{1}{m} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_{ij} + C \\
\text{s. t. } & x_{ij} + x_{jk} - x_{ik} \leq 1 && \text{for all } i < j < k, \\
& x_{ij} - x_{jk} + x_{ik} \leq 1 && \text{for all } i < j < k, \\
& -x_{ij} + x_{jk} + x_{ik} \leq 1 && \text{for all } i < j < k, \\
& x_{ij} \in \{0, 1\} && \text{for all } i < j.
\end{aligned}$$

This formulation contains $\binom{n}{2}$ variables and $3\binom{n}{3}$ constraints. The first three sets of constraints stipulate that for any triples of vertices $\{i, j, k\}$, if i and j belong to the same community and j and k also belong to the same community, then i and k belong to the same community. A linear programming relaxation (LP) can be derived by replacing the set of constraints $x_{ij} \in \{0, 1\}$ with $x_{ij} \in [0, 1]$. An upper bound of modularity can be obtained by solving (LP). As mentioned in reference [16], the optimal value of (LP) is sufficiently close to that of (IP) for many real-world networks. However, (LP) is prohibitive for networks exceeding a few hundred vertices, because the number of constraints grows rapidly with n.

2.2 Sparse formulation

In 2011, Dinh and Thai [24] proposed improved formulations for both (IP) and (LP), in which far fewer constraints generate the same sets of optimal solutions. Here, these formulations are reviewed. Let $N(i)$ be the set of all

neighbors of a vertex i. The union of neighbors of vertices i and j is denoted by $N(i,j) = N(i) \cup N(j) \setminus \{i,j\}$. Then, the improved formulation for (IP) can be written as:

$$(\mathrm{IP}_{\mathrm{sparse}}) \ : \ \max. \ \frac{1}{m} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_{ij} + C$$

$$\begin{aligned} \text{s. t. } & x_{ij} + x_{jk} - x_{ik} \leq 1 && \text{for all } i < j < k,\ j \in N(i,k), \\ & x_{ij} - x_{jk} + x_{ik} \leq 1 && \text{for all } i < j < k,\ i \in N(j,k), \\ & -x_{ij} + x_{jk} + x_{ik} \leq 1 && \text{for all } i < j < k,\ k \in N(i,j), \\ & x_{ij} \in \{0,1\} && \text{for all } i < j. \end{aligned}$$

The number of constraints in $(\mathrm{IP}_{\mathrm{sparse}})$ is upper bounded by:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (d_i + d_j) = (n-1) \sum_{i=1}^{n} d_i = \mathrm{O}(mn).$$

Most networks on which we attempt to find community structures are sparse, i.e., m is proportional to n. Under this condition, the number of constraints is substantially reducible to $\mathrm{O}(n^2)$. As mentioned above, the following theorem has been proposed [24].

Theorem 1. (IP) *and* $(\mathrm{IP}_{\mathrm{sparse}})$ *have the same set of optimal solutions.*

This theorem states that an optimal solution (and the optimal value) of modularity maximization can be obtained by replacing (IP) with $(\mathrm{IP}_{\mathrm{sparse}})$. A linear programming relaxation $(\mathrm{LP}_{\mathrm{sparse}})$ is therefore derived by replacing the set of constraints $x_{ij} \in \{0,1\}$ with $x_{ij} \in [0,1]$. The formulation adopts the following additional theorem [24].

Theorem 2. (LP) *and* $(\mathrm{LP}_{\mathrm{sparse}})$ *have the same set of optimal solutions.*

According to this theorem, the optimal value of $(\mathrm{LP}_{\mathrm{sparse}})$ provides the optimal value of the full linear programming. Computational experiments reported in reference [24] have shown that $(\mathrm{LP}_{\mathrm{sparse}})$ executes much more rapidly than (LP). However, according to our computational experiments, the execution time of $(\mathrm{LP}_{\mathrm{sparse}})$ remains prohibitively slow for networks containing only a few hundred vertices. Furthermore, because $(\mathrm{LP}_{\mathrm{sparse}})$, like (LP), requires $\binom{n}{2}$ variables and $\mathrm{O}(n^2)$ constraints, the memory capacity of $(\mathrm{LP}_{\mathrm{sparse}})$ precludes its application on networks with a few thousand vertices.

3 Row generation

3.1 Row generation revisited

Although the number of constraints in $(\mathrm{LP}_{\mathrm{sparse}})$ is far fewer, it grows rapidly with n. Therefore, as the number of vertices climbs into the hundreds, the number of constraints in $(\mathrm{LP}_{\mathrm{sparse}})$ excessively increases. The traditional row generation approach [18] was designed to alleviate this situation. The idea of this method is simple, that is, we wish to add only the constraints tight at the optimum. Obviously, an optimal solution can be obtained if $(\mathrm{LP}_{\mathrm{sparse}})$ is solved under such constraints alone; however, it is difficult to know in advance whether a given constraint is tight. Therefore, starting with no constraints, we obtain a temporary optimal solution and iteratively add the violated constraints to $(\mathrm{LP}_{\mathrm{sparse}})$. More specifically, row generation (RG) for $(\mathrm{LP}_{\mathrm{sparse}})$ is carried out in the following steps:

Step 1: obtain a temporary optimal solution $\overline{\boldsymbol{x}}^*$ and the objective value $\overline{Q}^*$ by solving $(\mathrm{LP}_{\mathrm{sparse}})$ with no constraints.

Step 2: scan all constraints of $(\mathrm{LP}_{\mathrm{sparse}})$. If $\overline{\boldsymbol{x}}^*$ satisfies all of them, then return $\overline{Q}^*$. Otherwise, add all constraints violated by $\overline{\boldsymbol{x}}^*$ and solve $(\mathrm{LP}_{\mathrm{sparse}})$. Update $\overline{\boldsymbol{x}}^*$ and $\overline{Q}^*$. Repeat Step 2.

From the computational experiments of Aloise et al. [21], RG is applicable to (LP) only for networks of 115 vertices or less, owing to numerous constraints of (LP). However, the small number of constraints in $(\mathrm{LP}_{\mathrm{sparse}})$ suggests that RG will execute more effectively on $(\mathrm{LP}_{\mathrm{sparse}})$ than on (LP).

3.2 Improved row generation

In this subsection, we present a simple approach that enhances the execution speed of RG. The formulation $(\mathrm{RLP}_{\mathrm{sparse}})$ is derived by replacing $N(i,k)$, $N(j,k)$, and $N(i,j)$ in $(\mathrm{LP}_{\mathrm{sparse}})$ with $N(i) \cap N(k)$, $N(j) \cap N(k)$, and $N(i) \cap N(j)$, respectively. Clearly, $(\mathrm{RLP}_{\mathrm{sparse}})$ is a relaxation of $(\mathrm{LP}_{\mathrm{sparse}})$. Although $(\mathrm{RLP}_{\mathrm{sparse}})$ is easily solved, its optimal value is not generally equal to that of $(\mathrm{LP}_{\mathrm{sparse}})$. Hence, we first solve $(\mathrm{RLP}_{\mathrm{sparse}})$, and must then introduce constraints, as well as perform RG. More precisely, *Improved Row Generation* (IRG) for $(\mathrm{LP}_{\mathrm{sparse}})$ is carried out in the following steps:

Step 1: obtain a temporary optimal solution $\overline{\boldsymbol{x}}^*$ and the objective value $\overline{Q}^*$ by solving $(\mathrm{RLP}_{\mathrm{sparse}})$.

Step 2: scan all constraints of $(\mathrm{LP}_{\mathrm{sparse}})$. If $\overline{\boldsymbol{x}}^*$ satisfies all of them, then return $\overline{Q}^*$. Otherwise, add all constraints violated by $\overline{\boldsymbol{x}}^*$ and solve $(\mathrm{LP}_{\mathrm{sparse}})$. Update $\overline{\boldsymbol{x}}^*$ and $\overline{Q}^*$. Repeat Step 2.

As mentioned above, this simple device effectively reduces the computation time. However, because the number of variables is unchanged, IRG is prohibitive for networks containing a few thousand vertices.

4 Row and column generation

In this section, we propose a row and column generation that computes the optimal value of $(\mathrm{LP}_{\mathrm{sparse}})$ with small

memory requirements. Initially, we construct a subproblem of ($\mathrm{LP}_{\mathrm{sparse}}$). The set of variable indices in ($\mathrm{LP}_{\mathrm{sparse}}$) is denoted by:

$$I_{\mathrm{all}} = \{(i,j) \mid i,j \in V, i < j\}.$$

Additionally, the set of edge indices in I_{all} is

$$I_{\mathrm{init}} = \{(i,j) \mid (i,j) \in I_{\mathrm{all}}, \{i,j\} \in E\},$$

and the remainder is denoted by:

$$I_{\mathrm{out}} = I_{\mathrm{all}} \setminus I_{\mathrm{init}}.$$

We note that the indices of all variables with positive coefficients are included in I_{init}. We consider the following subproblem of ($\mathrm{LP}_{\mathrm{sparse}}$), which includes all variables whose indices are in I_{init} and some variables whose indices are in I_{out}:

$$\begin{aligned}
\text{max. } & \frac{1}{m}\left(\sum_{(i,j)\in I_{\mathrm{init}}} q_{ij}x_{ij} + \sum_{(i,j)\in I_{\mathrm{added}}\subset I_{\mathrm{out}}} q_{ij}x_{ij}\right) + C \\
\text{s. t. } & x_{ij} + x_{jk} - x_{ik} \le 1 \\
& \text{for all } (i,j),(j,k),(i,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}},\ j \in N(i,k), \\
& x_{ij} - x_{jk} + x_{ik} \le 1 \\
& \text{for all } (i,j),(j,k),(i,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}},\ i \in N(j,k), \\
& -x_{ij} + x_{jk} + x_{ik} \le 1 \\
& \text{for all } (i,j),(j,k),(i,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}},\ k \in N(i,j), \\
& x_{ij} \in [0,1] \\
& \text{for all } (i,j) \in I_{\mathrm{init}} \cup I_{\mathrm{added}}.
\end{aligned}$$

This formulation is represented as ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) because it depends on how $I_{\mathrm{added}} \subset I_{\mathrm{out}}$ is specified. For instance, ($\mathrm{LP}_{\mathrm{sparse}}(\emptyset)$) is the smallest formulation containing edge variables alone, and ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{out}})$) is equivalent to ($\mathrm{LP}_{\mathrm{sparse}}$) itself. The following lemma embodies a useful property of ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$).

Lemma 1. *For any I and I' such that*

$$I \subset I' \subset I_{\mathrm{out}},$$

the optimal value of ($\mathrm{LP}_{\mathrm{sparse}}(I)$) *is greater than or equal to that of* ($\mathrm{LP}_{\mathrm{sparse}}(I')$).

Proof. Consider an arbitrary I and I' satisfying the above condition, and an arbitrary feasible solution

$$\boldsymbol{x}' = (x'_{ij})_{(i,j)\in I_{\mathrm{init}}\cup I'}$$

of ($\mathrm{LP}_{\mathrm{sparse}}(I')$). Clearly,

$$\boldsymbol{x} = (x_{ij})_{(i,j)\in I_{\mathrm{init}}\cup I},$$

obtained by removing all $(i,j) \in I' \setminus I$ components of $\boldsymbol{x}'$, is a feasible solution of ($\mathrm{LP}_{\mathrm{sparse}}(I)$). Moreover, the objective value of $\boldsymbol{x}$ is greater than or equal to that of $\boldsymbol{x}'$ because

$$\begin{aligned}
&\frac{1}{m}\sum_{(i,j)\in I_{\mathrm{init}}\cup I'} q_{ij}x'_{ij} + C \\
&\quad = \frac{1}{m}\left(\sum_{(i,j)\in I_{\mathrm{init}}} q_{ij}x'_{ij} + \sum_{(i,j)\in I} q_{ij}x'_{ij} \right. \\
&\qquad \left. + \sum_{(i,j)\in I'\setminus I} q_{ij}x'_{ij}\right) + C \\
&\quad \le \frac{1}{m}\left(\sum_{(i,j)\in I_{\mathrm{init}}} q_{ij}x_{ij} + \sum_{(i,j)\in I} q_{ij}x_{ij}\right) + C.
\end{aligned}$$

The above inequality follows because $q_{ij} \le 0$ for all $(i,j) \in I_{\mathrm{out}}$.

By this lemma, we present the following corollary.

Corollary 1. *For any $I_{\mathrm{added}} \subset I_{\mathrm{out}}$, the optimal value of* ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) *is greater than or equal to that of* ($\mathrm{LP}_{\mathrm{sparse}}$).

From the above, we deduce that ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) provides an upper bound of the optimal value of ($\mathrm{LP}_{\mathrm{sparse}}$) for any $I_{\mathrm{added}} \subset I_{\mathrm{out}}$ (Corollary 1), and that adding variables can improve the upper bound or at least maintain it (Lemma 1). Given these properties of ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$), we formulate the following natural column generation; we first solve ($\mathrm{LP}_{\mathrm{sparse}}(\emptyset)$) and add variables iteratively. However, it is not yet known (i) which variables should be added in each iteration and (ii) when the upper bound equates to the optimal value of ($\mathrm{LP}_{\mathrm{sparse}}$). The second of these unknowns is solved by the following lemma, that is, we seek a sufficient condition under which the optimal value of ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) equals that of ($\mathrm{LP}_{\mathrm{sparse}}$).

Lemma 2. *If an optimal solution*

$$\overline{\boldsymbol{x}}^* = (\overline{x}^*_{ij})_{(i,j)\in I_{\mathrm{init}}\cup I_{\mathrm{added}}}$$

of ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) *satisfies the following condition*

$$(*)\begin{cases}
\overline{x}^*_{ij} + \overline{x}^*_{jk} \le 1 \\
\text{for all } (i,j),(j,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}},\ j \in N(i,k), \\
\qquad (i,k) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}, \\
\overline{x}^*_{ij} + \overline{x}^*_{ik} \le 1 \\
\text{for all } (i,j),(i,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}},\ i \in N(j,k), \\
\qquad (j,k) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}, \\
\overline{x}^*_{jk} + \overline{x}^*_{ik} \le 1 \\
\text{for all } (j,k),(i,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}},\ k \in N(i,j), \\
\qquad (i,j) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}},
\end{cases}$$

then

$$\boldsymbol{x}^* = (x^*_{ij})_{(i,j)\in I_{\mathrm{all}}}$$

such that

$$x^*_{ij} = \begin{cases} \overline{x}^*_{ij} & \text{if } (i,j) \in I_{\mathrm{init}} \cup I_{\mathrm{added}}, \\ 0 & \text{otherwise}, \end{cases}$$

is an optimal solution of ($\mathrm{LP}_{\mathrm{sparse}}$).

Proof. Taking an arbitrary optimal solution

$$\overline{\boldsymbol{x}}^* = (\overline{x}^*_{ij})_{(i,j)\in I_{\mathrm{init}}\cup I_{\mathrm{added}}}$$

of ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) satisfying the condition ($*$), we generate the solution

$$\boldsymbol{x}^* = (x^*_{ij})_{(i,j)\in I_{\mathrm{all}}}$$

as above. Clearly, the objective value of $\boldsymbol{x}^*$ in ($\mathrm{LP}_{\mathrm{sparse}}$) equals that of $\overline{\boldsymbol{x}}^*$ in ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$). From this fact and Corollary 1, it suffices to show that $\boldsymbol{x}^*$ is feasible for ($\mathrm{LP}_{\mathrm{sparse}}$). To this end, we consider the first set of constraints

$$x_{ij} + x_{jk} - x_{ik} \le 1 \quad \text{for all } i < j < k,\ j \in N(i,k)$$

in ($\mathrm{LP}_{\mathrm{sparse}}$). If $(i,j),(j,k),(i,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}}$, the constraints are satisfied because they are also in ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$). If $(i,j) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}$ or $(j,k) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}$, the constraints are also satisfied because we have $x^*_{ij} = 0$ or $x^*_{jk} = 0$. Furthermore, if $(i,j),(j,k) \in I_{\mathrm{init}} \cup I_{\mathrm{added}}$ and $(i,k) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}$, the constraints are again satisfied owing to the condition ($*$). The remaining two sets of constraints are treated similarly.

In fact, Lemma 2 implicitly solves the unknown (i) above; the variables x_{ij} with $(i,j) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}$ appearing in violated inequalities in ($*$) should be added in each iteration, until a temporary optimal solution satisfies ($*$). We now propose *row and column generation* (RCG) which operates as follows:

Step 1: set $I_{\mathrm{added}} = \emptyset$.
Step 2: solve ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$) by RG to obtain a temporary optimal solution $\overline{\boldsymbol{x}}^*$ and the objective value $\overline{Q}^*$.
Step 3: if $\overline{\boldsymbol{x}}^*$ satisfies the condition ($*$), then return $\overline{Q}^*$. Otherwise, add all indices $(i,j) \in I_{\mathrm{out}} \setminus I_{\mathrm{added}}$ that appear in violated inequalities in the condition ($*$) to I_{added}. Return to Step 2.

Given an optimal solution of ($\mathrm{LP}_{\mathrm{sparse}}(I_{\mathrm{added}})$), we can check whether the solution satisfies ($*$) in an exceedingly shorter time than by solving linear programming parts. From Lemma 2, we may state the following fact.

Theorem 3. RCG *obtains an optimal solution and the optimal value of* ($\mathrm{LP}_{\mathrm{sparse}}$).

5 Computational experiments

The goal of this section is to (i) compare the performances of conventional methods and our proposed methods and (ii) provide an unknown optimal value of ($\mathrm{LP}_{\mathrm{sparse}}$) for a famous large test network. Table 1 lists the networks on which experiments were conducted. These networks are commonly used to evaluate heuristics for modularity maximization. For reference, Table 2 lists modularity obtained by several popular heuristics and the best known lower bound. In all experiments, linear programming was solved using Gurobi Optimizer 4.5.0 on a Windows-based computer with 3.6 GHz processors and 4 GB RAM.

Table 1. Networks used in our computational experiments, stating the number of vertices and edges. The networks are constructed from the following datasets: friendships between members of a karate club [25], communications between dolphins living in Doubtful Sound, New Zealand [26], the co-appearance of characters in the famous novel Les Misérables [27], the co-purchasing of books on U.S. politics by the same buyer [28], American football games between Division IA colleges [29], connections between U.S. airports [30], electronic circuits [31], co-authorships in network theory and experiment [32], and the topology of the Western States Power Grid of the United States [23].

ID	Name	n	m
1	Zachary's karate club	34	78
2	Dolphin's social network	62	159
3	Les Misérables	77	254
4	Books about U.S. politics	105	441
5	American College Football	115	613
6	USAir97	332	2126
7	Electronic Circuit (s838)	512	819
8	Netscience	1589	2742
9	Power Grid	4941	6594

The results are detailed in Table 3, which infers the following:

(i) the simple method for solving ($\mathrm{LP}_{\mathrm{sparse}}$) (named SIMPLE) requires a relatively long computation time to obtain the upper bounds. In particular, network 6 consumes 22 006 s using this approach. Moreover, computation on networks 8 and 9 are precluded by limited memory capacity. Although ($\mathrm{LP}_{\mathrm{sparse}}$) has considerably fewer constraints than (LP), the applicability of SIMPLE is severely restricted;
(ii) the computational speed and memory efficiency of RG significantly exceed those of SIMPLE. For all networks, less time is required to compute the upper bounds and fewer constraints are added to the solver. Consequently, RG can derive the upper bound for network 8 in 512.99 s. However, like SIMPLE, RG cannot be implemented on network 9;
(iii) IRG is more powerful than RG. The upper bounds are obtained in less time for all networks, while the number of constraints is not excessively increased. However, IRG, like its predecessors, cannot be implemented on network 9;
(iv) most strikingly, the number of variables added to the solver is drastically reduced in RCG. In particular, only 1.93% and 6.95% of the variables in networks 8 and 9, respectively, are input to RCG. RCG is undoubtedly the best approach in terms of memory efficiency;
(v) although the computation time is extended, RCG provides the optimal value of ($\mathrm{LP}_{\mathrm{sparse}}$) for network 9. To our knowledge, this nontrivial upper bound has not been previously reported;

Table 2. Modularity obtained by several popular heuristics and the best known lower bound. The first column lists the ID assigned to each network in Table 1. GN, CNM, SA, SD, and CHL represent the Girvan-Newman algorithm [29], the hierarchical agglomerative method by Clauset et al. [5], the simulated annealing by Guimerà and Amaral [33], the Newman's spectral divisive method [11], and the recent divisive method by Cafieri et al. [34], respectively (note that the values of GA and SA are reported in Ref. [16], and the values of CNM, SD, and CHL are reported in Ref. [34]. NA indicates that the network is not considered in the literature). LB in the last column represents the best known lower bound of modularity (note that LB for networks 1–7 is the optimal modularity reported in Ref. [34]. LB for networks 8 and 9 are reported in Refs. [35] and [36], respectively).

ID	GN	CNM	SA	SD	CHL	LB
1	0.401	0.38067	0.420	0.39341	0.41880	0.41979
2	0.520	0.49549	0.527	0.49120	0.52646	0.52852
3	0.540	0.50060	0.556	0.51383	0.54676	0.56001
4	NA	0.50197	0.527	0.46718	0.52629	0.52724
5	0.601	0.57728	0.604	0.49261	0.60091	0.60457
6	NA	0.32039	NA	0.31666	0.35959	0.3682
7	NA	0.80556	NA	0.73392	0.81663	0.8194
8	NA	NA	NA	NA	NA	0.954
9	NA	0.93402	NA	0.53516	0.93937	0.9396

Table 3. Results of computational experiments. The first column lists the ID assigned to each network in Table 1. UB in the second column represents the optimal value of (LP_{sparse}) obtained by conventional methods and/or our methods. The columns of (LP_{sparse}) lists the number of variables and the number of constraints in the formulation. The column of SIMPLE gives the computation time required to simply solve (LP_{sparse}). OM in some columns indicates that computation could not be started owing to a shortage of memory capacity. The columns of RG and IRG list the computation time and the number of constraints added to the solver by each method, respectively. The columns of RCG list the number of variables added to the solver, in addition to the computation time and the number of constraints.

ID	UB	(LP_{sparse})		SIMPLE	RG		IRG		RCG		
		var.	constr.	time(s)	time(s)	constr.	time(s)	constr.	time(s)	var.	constr.
1	0.41979	561	4466	0.39	0.17	539	0.12	546	0.17	347	546
2	0.53146	1891	18157	0.78	0.58	1757	0.48	2081	0.75	881	1996
3	0.56088	2926	35292	0.97	0.81	2438	0.67	3032	1.00	1311	3007
4	0.52759	5460	86024	15.37	3.11	9336	2.14	9044	3.87	2693	8989
5	0.60563	6555	132571	27.75	3.54	8472	3.03	10902	4.81	3424	10079
6	0.37320	54946	1310971	22006.36	4091.88	212020	2615.86	126697	2722.44	29960	125671
7	0.82612	130816	832899	686.44	313.24	68991	294.76	69054	423.22	28795	69764
8	0.95990	1261666	8689566	OM	512.99	54065	413.38	58165	33.47	24412	61683
9	0.94295	12204270	65123193	OM	OM	–	OM	–	1694050	848802	2009011

(vi) in all experiments, the upper and lower bounds are very close.

From the above results, we posit that RCG is superior to other approaches because its computation time is comparable to that of the other approaches, yet RCG alone can compute the upper bound for network 9. We note that the optimal solution of (LP_{sparse}) for network 9 obtained by RCG is fractional. Therefore, it remains uncertain whether this upper bound equates to the optimal modularity.

6 Conclusions

In this study, we proposed two powerful methods (IRG and RCG) for computing a nontrivial upper bound of modularity. More specifically, these methods can obtain the optimal value of a linear programming relaxation of the standard formulation for modularity maximization. IRG is based on the traditional row generation, and RCG is based on a row and column generation. RCG requires exceedingly small memory capacity owing to the reduced number of variables that are input to the solver. As a result, RCG can compute the upper bound for the famous test network consisting of 4941 vertices and 6594 edges. To our knowledge, this value was previously unknown. The present study could make valuable contributions to evaluation of heuristics for modularity maximization.

In addition, our methods improve the community detection method proposed by Agarwal and Kempe [16]. Their algorithm first solves (LP) and then applies a rounding technique to the obtained fractional solution. Their computational experiments show that the algorithm outperforms other conventional methods in terms of accuracy. However, their algorithm is prohibitive for networks exceeding a few hundred vertices, because it solves (LP) simply. Applying our methods makes their algorithm more rapid and memory-efficient.

In closing, we remark that our methods are applicable to other modularity-reliant problems such as modularity maximization on weighted graphs. Moreover, they are easily adaptable to a wide range of graph partitioning problems represented by the clique partitioning problem which is a natural generalization of modularity maximization.

The authors would like to thank Dr. Masakazu Kojima for the fruitful discussions and helpful advices.

References

1. S. Fortunato, Phys. Rep. **486**, 75 (2010)
2. M.E.J. Newman, M. Girvan, Phys. Rev. E **69**, 026113 (2004)
3. S. Fortunato, M. Barthélemy, Proc. Natl. Acad. Sci. USA **104**, 36 (2007)
4. U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, D. Wagner, IEEE Trans. Knowl. Data Eng. **20**, 172 (2008)
5. A. Clauset, M.E.J. Newman, C. Moore, Phys. Rev. E **70**, 066111 (2004)
6. V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, J. Stat. Mech. **2008**, P10008 (2008)
7. R. Guimerà, L.A.N. Amaral, Nature **433**, 895 (2005)
8. C.P. Massen, J.P.K. Doye, Phys. Rev. E **71**, 046101 (2005)
9. A.D. Medus, G. Acuña, C.O. Dorso, Physica A **358**, 593 (2005)
10. J. Duch, A. Arenas, Phys. Rev. E **72**, 027104 (2005)
11. M.E.J. Newman, Proc. Natl. Acad. Sci. USA **103**, 8577 (2006)
12. T. Richardson, P.J. Mucha, M.A. Porter, Phys. Rev. E **80**, 036111 (2009)
13. S. Boccaletti, M. Ivanchenko, V. Latora, A. Pluchino, A. Rapisarda, Phys. Rev. E **75**, 045102 (2007)
14. H.N. Djidjev, Lect. Notes Comput. Sci. **4936**, 117 (2008)
15. Y. Niu, B. Hu, W. Zhang, M. Wang, Physica A **387**, 6215 (2008)
16. G. Agarwal, D. Kempe, Eur. Phys. J. B **66**, 409 (2008)
17. S. Cafieri, P. Hansen, L. Liberti, Discrete Appl. Math., in press, DOI: 10.1016/j.dam.2012.03.030
18. M. Grötschel, Y. Wakabayashi, Math. Prog. **45**, 59 (1989)
19. M. Grötschel, Y. Wakabayashi, Math. Prog. **47**, 367 (1990)
20. G. Xu, S. Tsoka, L.G. Papageorgiou, Eur. Phys. J. B **60**, 231 (2007)
21. D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, L. Liberti, Phys. Rev. E **82**, 046112 (2010)
22. B. DasGupta, D. Desai, J. Comput. System Sci. **79**, 50 (2013)
23. D.J. Watts, S.H. Strogatz, Nature **393**, 440 (1998)
24. T.N. Dinh, M.T. Thai, arXiv:1108.4034 (2011)
25. W. Zachary, J. Anthropol. Res. **33**, 452 (1977)
26. D. Lusseau, K. Schneider, O.J. Boisseau, P. Haase, E. Slooten, S.M. Dawson, Behav. Ecol. Sociobiol. **54**, 396 (2003)
27. D.E. Knuth, *The Stanford GraphBase: A Platform for Combinatorial Computing* (Addison-Wesley, Reading, MA, 1993)
28. http://www.orgnet.com/
29. M. Girvan, M.E.J. Newman, Proc. Natl. Acad. Sci. USA **99**, 7821 (2002)
30. http://vlado.fmf.uni-lj.si/pub/networks/data/
31. http://www.weizmann.ac.il/mcb/UriAlon/
32. M.E.J. Newman, Phys. Rev. E **74**, 036104 (2006)
33. R. Guimerà, L.A.N. Amaral, J. Stat. Mech. **2005**, P02001 (2005)
34. S. Cafieri, P. Hansen, L. Liberti, Phys. Rev. E **83**, 056105 (2011)
35. W. Li, D. Schuurmans, in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, IJCAI'11 (AAAI Press, 2011), Vol. 2, pp. 1366–1371
36. S. Cafieri, A. Costa, P. Hansen, Ann. Oper. Res., in press, DOI: 10.1007/s10479-012-1286-z