Column generation algorithms for exact modularity maximization in networks

Daniel Aloise*
Department of Production Engineering, Universidade Federal do Rio Grande do Norte, Campus Universitário s/n, Natal, RN 59072-970, Brazil

Sonia Cafieri†
Département de Mathématiques et Informatique, École Nationale de l'Aviation Civile, 7 Ave. E. Belin, F-31055 Toulouse, France

Gilles Caporossi,‡ Pierre Hansen,§ and Sylvain Perron‖
GERAD and HEC Montreal, 3000 Chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Leo Liberti¶
LIX, École Polytechnique, F-91128 Palaiseau, France
(Received 4 June 2010; published 21 October 2010)

Finding modules, or clusters, in networks currently attracts much attention in several domains. The most studied criterion for doing so, due to Newman and Girvan [Phys. Rev. E **69**, 026113 (2004)], is modularity maximization. Many heuristics have been proposed for maximizing modularity and yield rapidly near optimal solution or sometimes optimal ones but without a guarantee of optimality. There are few exact algorithms, prominent among which is a paper by Xu *et al.* [Eur. Phys. J. B **60**, 231 (2007)]. Modularity maximization can also be expressed as a clique partitioning problem and the row generation algorithm of Grötschel and Wakabayashi [Math. Program. **45**, 59 (1989)] applied. We propose to extend both of these algorithms using the powerful column generation methods for linear and non linear integer programming. Performance of the four resulting algorithms is compared on problems from the literature. Instances with up to 512 entities are solved exactly. Moreover, the computing time of previously solved problems are reduced substantially.

DOI: 10.1103/PhysRevE.82.046112 PACS number(s): 89.75.Hc, 87.23.Ge, 02.70.−c

I. INTRODUCTION

Clustering is an important chapter of data analysis and data mining with numerous applications in a variety of fields. It aims at solving the following general problem: given a set of entities, find subsets, or clusters, which are homogeneous and/or well separated. As the concepts of homogeneity and of separation can be made precise in many ways, there are a large variety of clustering problems [1–4]. These problems in turn are solved by exact algorithms or, more often and particularly for large data sets, by heuristics. An exact algorithm provides, hopefully in reasonable computing time, an optimal solution together with a proof of its optimality. A heuristic provides, usually in moderate computing time, a near optimal solution or sometimes an optimal solution but without proof of its optimality.

In the last decade, clustering on networks has been extensively studied, mostly in the physics and computer science research communities. Rather than using the term cluster, the words *module* or *community* are often adopted in the physics literature to denote homogeneous and/or well separated subsets of entities. Recall that a network, or graph, $G=(V,E)$ is composed of a set V of n vertices and a set E of m edges, which join pairs of vertices. A vertex v_j is represented by a point and an edge $e_{ij}=\{v_i,v_j\}$ by a line joining its two end vertices v_i and v_j. The shape of this line does not matter, only the presence or absence of an edge is important. In a simple graph, there is at most one edge between any pair of vertices, otherwise one has a multigraph. With a slight abuse of set notation, a loop $e_{ii}=\{v_i,v_i\}$ is an edge for which both end vertices coincide. The degree k_i of a vertex $v_i \in V$ is the number of edges incident with v_i. In this paper, we focus on undirected, unweighted graphs without loops.

A subgraph $G_S=(S,E_S)$ of a graph $G=(V,E)$ induced by a set of vertices $S \subseteq V$ is a graph with vertex set S and edge set E_S equal to all edges with both vertices in S. Such a subgraph corresponds to a cluster (or module or community) and many heuristics aim at finding a partition of V into pairwise disjoint nonempty subsets $V_1, V_2, \ldots, V_N$ inducing subgraphs of G. Various objective functions have been proposed for evaluating such a partition. Roughly speaking, one seeks modules which contain more inner edges (with both vertices in the same module) than cut edges (with vertices in different modules). The degree k_i of the vertex v_i can be split in two: the indegree k_i^{in} or number of neighbors within its community and the outdegree k_i^{out} or number of neighbors outside its community. Several concepts of community follow.

In 2004, Radicchi *et al.* [5] defined a *community in the strong sense* as a subgraph all vertices of which have larger indegree than outdegree and a *community in the weak sense* as a subgraph for which the sum of vertex indegrees is larger than the sum of vertex outdegrees. As an inner edge contrib-

*daniel.aloise@gerad.ca
†sonia.cafieri@enac.fr
‡gilles.caporossi@gerad.ca
§Also at LIX, Ecole Polytechnique, France. pierre.hansen@gerad.ca
‖sylvain.perron@gerad.ca
¶liberti@lix.polytechnique.fr

utes by two to the sum of the indegrees and a cut edge contributes by one to the sum of outdegrees, the number of inner edges in a community in the weak sense must be at least as large as half the number of cut edges. As cut edges contribute to the sum of degrees of two communities, this definition entails that for the network as a whole the number of inner edges is larger than the number of cut edges. Recently, several extensions of the definition of the *community in the weak sense* have been proposed. One may consider the difference for each community of the sum of indegrees and the sum of outdegrees. Then summing these contributions for all communities gives a multiway cut problem. Another approach is to normalize the contribution of each community by dividing it by its number of vertices [6]. The resulting function, to be maximized, is called *modularity density*. Alternatively, contributions of communities may be divided by their number of edges [7]. Finally, one may consider maximizing, in a divisive hierarchical method, the minimum ratio of the number of edges in a community divided by the number of cut edges [8].

A different, and currently mainstream, approach was inaugurated by Newman and Girvan also in 2004 [9]. They propose to find a partition of V which maximizes the sum, over all modules, of the number of inner edges minus the expected number of such edges assuming that they are drawn at random with the same distribution of degrees as in G. In [9] the following precise definition of modularity is given:

$$Q = \sum_s [a_s - e_s], \tag{1}$$

where a_s is the fraction of all edges that lie within module s and e_s is the expected value of the same quantity in a graph in which the vertices have the same expected degrees but edges are placed at random. A maximum value of Q near to 0 indicates that the network considered is close to a random one (barring fluctuations), while a maximum value of Q near to 1 indicates strong community structure. Observe that maximizing modularity gives an optimal partition together with the optimal number of modules.

The modularity maximization problem has been extensively studied both from the algorithmic and from the applications viewpoints. Some papers discuss a few of its mathematical properties, among which are the following:

(i) Even if the networks under study most of the time have no loops nor multiple edges, the expected number of loops may be positive. Moreover, for some pairs of end vertices, their expected number of edges may be greater than 1 [10];

(ii) Modularity maximization suffers from a resolution limit [11], i.e., when the network is large small modules can be absorbed by larger ones even if they are very dense;

(iii) The partition with optimal or near optimal modularity may contain modules which are not *communities in the weak sense*, or even in the *most weak sense* [12] that twice the number of inner edges is never less than the number of edges joining this module to another one.

Modification to the model and/or to heuristics have been proposed to address these problems. Changing slightly the null model [i.e., the value of e_s in Eq. (1)], by simulation [10] or by using an analytical formula [13] allows the removal of loops and multiple edges. A parameter can also be introduced together with multiresolution heuristics [14,15]. Nevertheless, the original modularity maximization appears presently to be much used. In this paper, we limit ourselves to this classical problem.

Brandes *et al.* [16] have shown that modularity maximization is NP-hard. Numerous heuristics have been proposed to maximize modularity. They are based on divisive hierarchical clustering, agglomerative hierarchical clustering, partitioning, and hybrids. They rely upon various criteria for agglomeration or division [17–21], simulated annealing [10,22,23], mean field annealing [24], genetic search [25], extremal optimization [26], spectral clustering [27–29], linear programming followed by randomized rounding [30], dynamical clustering [31], multilevel partitioning [32], contraction-dilation [33], multistep greedy search [34], quantum mechanics [35], and other approaches [14,21,29,36–38].

In contrast, papers proposing exact algorithms or using mathematical programming are rare for modularity maximization. There are two approaches. In the first one, the original graph $G=(V,E)$ is replaced by a complete weighted graph $K_n=(V,E')$ of order $n=|V|$ as G and such that for any pair of vertices $v_i, v_j \in G$, K_n has an edge e'_{ij} with a weight equal to $\frac{1}{m}(a_{ij}-\frac{k_i k_j}{m})$, where $a_{ij}=1$ if v_i and v_j are adjacent in G and m is the number of edges [27].

In addition to the complexity result mentioned above, Brandes *et al.* [16] give an integer programming formulation for modularity maximization and mention that the optimal solution of two test problems with 34 and with 105 entities were determined. Their approach is in fact close to the work of Grötschel and Wakabayashi [39,40] on clique partitioning. It is discussed in the next section. The second approach works directly on the original graph $G=(V,E)$. Xu, Tsoka, and Papageorgiou [41] propose a mixed integer convex quadratic programming model, discussed below. They solve exactly four test problems with up to 104 entities.

The purpose of the present paper is to assess and advance the state of the art of algorithms for exact modularity maximization. To this effect, we discuss and compare four exact algorithms, two of which are new. Two of these algorithms work on a reduction of modularity maximization to clique partitioning; the other two work on the direct formulation. They are: (i) the row generation algorithm of [39], which subsumes the algorithm of [16], (ii) a new stabilized column generation algorithm for clique partitioning which enhances the efficiency of that approach, (iii) the mixed integer convex quadratic programming approach of [41], (iv) another new stabilized column generation algorithm which enhances the efficiency of the second approach.

Column generation algorithms implicitly take into account all possible communities (or in other words all subsets of the set of entities under study). They replace the problem of finding simultaneously all communities in an optimal partition by a sequence of optimization problems for finding one community at a time, or more precisely a community which improves the modularity of the current solution. So, problems are solved much faster than with previous algorithms and larger instances can be tackled, the largest to date having 512 entities.

Clearly, in many applications of modularity maximization, instances are still larger and sometimes very much

larger, than those which can be solved exactly with the proposed algorithms. Nevertheless, it is our belief that exact algorithms are worthy of study for several reasons.

(i) Instances which are more than toy problems can presently be solved exactly. As shown below, these include many problems used to illustrate the performance of various heuristics.

(ii) As mentioned in several papers, finding the significance of the detected communities is difficult. Indeed, various heuristics often lead to partitions that disagree on two or more of their communities. Having an exact solution solves the problem of separating possible inadequacies of the model from eventual errors resulting from the use of heuristics. Unsuspected communities may be interpreted with more confidence and proposed to the user for a substantive analysis. This is one of the main aims of clustering.

(iii) If solving a given instance proves to be too time consuming, the exact algorithm may often be stopped and the best solution found considered as a heuristic one. It is not uncommon that the optimal solution is found at an early stage of the resolution. Then, the problem of maximizing modularity will, in fact, be solved but without a proof of optimality.

(iv) An exact algorithm can provide a benchmark of exactly solved instances which can be used to compare heuristics and fine tune them. More precisely, the comparison of the symmetric differences between the optimal solution and the heuristically obtained ones may suggest additional moves which improve the heuristic under study. Iterating this approach with several heuristics may lead to performant hybrids. As a rule of thumb, a sophisticated heuristic should be able to find quickly an optimal solution for most or possibly all practical instances which can be solved exactly with a proof of optimality.

(v) Using projection, i.e., fixing some of the communities found by a heuristic, an exact algorithm can be applied to this reduced network in order to improve the heuristic solution [42].

(vi) Getting improved heuristics will in turn lead to more efficient exact algorithms. Indeed, there are usually some steps in such algorithms which can be solved heuristically, always or most of the time, without forfeiting the guarantee of optimality. In column generation algorithms, as discussed below, knowledge of a good or possibly optimal initial solution enhances stabilization substantially. Heuristics are also useful in solving the auxiliary problem.

(vii) An exact algorithm can also be used as a tool for the theoretical study of maximum modularity partitions, e.g., the fact that they satisfy or not conditions on the number of inner and of cut edges of communities, or conditions of robustness [12].

(viii) With active research, regularly improved mathematical programming packages such as CPLEX [43] and increasing computer power, the size of problems solved exactly is likely to increase substantially with time. A comparison with a central problem of Operations Research, i.e., the traveling salesman problem, illustrates this point. Given a set of cities and pairwise distances between them, the traveling salesman problem is to find a minimum length tour visiting once and only once each city. In 1954, Dantzig, Fulkerson, and Johnson [44] solved optimally an instance with 49 cities. In 2009, Applegate *et al.* [45] were able to solve to optimality an instance with 85 900 cities.

II. MODULARITY MAXIMIZATION AS CLIQUE PARTITIONING

A. Row generation

The modularity function was expressed above as a sum of values over all communities. As shown in [27], modularity can also be written as a sum of values over all edges of the complete graph K_n.

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left(a_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \tag{2}$$

where m corresponds to the cardinality of E, k_i, k_j are the degrees of vertices i and j, respectively, a_{ij} is the ij component of the adjacency matrix of G, equal to 1 if vertices i and j are adjacent, and to 0 otherwise. Finally, $\delta(c_i, c_j)$ is the Kronecker symbol equal to 1 if the communities c_i and c_j, to which i and j belong, are the same, and to 0 otherwise. The quantity $\frac{k_i k_j}{2m}$ is the expected number of edges between vertices i and j in a null model where edges are placed at random, while the distribution of degrees remains the same.

Introducing binary variables x_{ij} equal to 1 if vertices i and j belong to the same module and 0 otherwise, and setting

$$w_{ij} = \frac{1}{m} \left(a_{ij} - \frac{k_i k_j}{2m} \right), \tag{3}$$

modularity maximization can be reformulated as a clique partitioning problem. Since K_n is complete, it is a clique and any of its induced subgraphs are cliques also. Partitioning G is thus equivalent to partitioning K_n into cliques. The resulting partition is an equivalence relation, i.e., reflexive, symmetric, and transitive. From reflexivity (or the fact that each entity is in the same module as itself) all $x_{ii}=1$. Thus the sum of elements on the main diagonal is a constant for all partitions, equal to

$$-C = -\sum_{i \in V} \frac{k_i k_i}{2m}. \tag{4}$$

From symmetry, using $x_{ij}=x_{ji}$, one can eliminate variables corresponding to values of indices $i>j$. The model can then be written, as in [39,40],

$$\begin{aligned} &\max \sum_{i<j \in V} w_{ij} x_{ij} - C, \\ &\text{s.t.} \quad x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for all} \quad 1 \leq i < j < k \leq n, \\ &\qquad x_{ij} - x_{jk} + x_{ik} \leq 1 \quad \text{for all} \quad 1 \leq i < j < k \leq n, \\ &\qquad -x_{ij} + x_{jk} + x_{ik} \leq 1 \quad \text{for all} \quad 1 \leq i < j < k \leq n, \\ &\qquad x_{ij} \in \{0,1\} \quad \text{for all} \quad 1 \leq i < j \leq n. \end{aligned} \tag{5}$$

This model has $\frac{n(n-1)}{2}$ variables and $3\binom{n}{3} = \frac{n(n-1)(n-2)}{2} = O(n^3)$ constraints. The first three sets of constraints express

transitivity, i.e., if entities i and j are in the same module and entities j and k are in the same module, then entities i and k must be in the same module. The fourth (and last) set of constraints expresses integrality, i.e., edges are present in the solution entirely or not at all.

Problem [Eq. (5)] is a linear program in 0–1 variables and thus small instances may be solved by integer linear programming packages such as CPLEX. In the solution process, the linear programming relaxation obtained by replacing the constraints $x_{ij} \in \{0,1\}$ by $x_{ij} \in [0,1]$ is first solved. If the optimal solution of this relaxation is in integers, which is often the case, it corresponds to a partition of maximum modularity.

Should the solution of the continuous relaxation be fractional, one can branch, or add one or several cutting planes, i.e., additional linear constraints which cut off the current fractional solution but do not eliminate any feasible integer solution. Branching is done by setting a fractional 0–1 variable x_{ij} to 1 or 0, i.e., imposing on the one hand that entities i and j belong to the same community and on the other hand that they belong to different communities. Two linear programming subproblems are thus obtained and their solution gives bounds valid for the former and the latter case, respectively. These bounds are not larger than the bound given by the solution of the subproblem on which branching took place. Branching on a subproblem stops if the value of the corresponding bound is smaller than that of the best solution known, or incumbent. As the possible number of branching choices is finite, the algorithm converges. Choosing a variable x_{ij} with value closest to $\frac{1}{2}$ balances improvement of both bounds. These rules and others are discussed in [46]. In practice, branching is done by CPLEX. Several families of cutting planes were obtained in [39,40] and could be used as an alternative or complement to branching.

Note that a similar formulation is presented in [16] but with constraints of the type $x_{ij}+x_{jk}-2x_{ik} \leq 1$ instead of $x_{ij}+x_{jk}-x_{ik} \leq 1$, and so forth. While this does not change the set of feasible solutions in 0–1 variables, the continuous relaxation in the algorithm of [16] will be less tight than that of Eq. (5). Consequently, the algorithm is more time consuming than that of [39] as it is shown in our computational experiments (see Sec. IV below).

Unfortunately, the number of constraints of problem [Eq. (5)] grows rapidly with n. Not all of these constraints will be tight at the optimum. Those which are not could be deleted without changing the optimal solution but it is not known *a priori* which ones they are. Grötschel and Wakabayashi [39] therefore proposed to add the constraints progressively. More precisely, they add by batches of 300 those constraints which are most violated by the current solution. In their experiments on a variety of clustering problems, they found that only a small number of such constraints were tight at the optimum and that the solution of the continuous relaxation was very often integer. Our computation experiments (see Sec. IV below) show this appears also to be the case for modularity maximization.

B. Column generation

Column generation is a powerful technique of linear programming which allows the exact solution of linear programs with a number of columns exponential in the size of the input (there may be billions of them and, in some cases, much more). To this effect, it follows the usual steps of the simplex algorithm, apart from finding an entering column with a positive reduced cost in case of maximization which is done by solving an auxiliary problem. The precise form of this last problem depends on the type of problem under study. It is often a combinatorial optimization or a global optimization problem. It can be solved heuristically as long as a column with a reduced cost of the required sign can be found. When this is no longer the case, an exact algorithm for the auxiliary problem must be applied either to find a column with the adequate reduced cost sign, undetected by the heuristic, or to prove that there is no such column and hence the linear programming relaxation is solved. Column generation has proven to be very useful in the solution of large clustering problems, e.g., minimum sum-of-squares clustering [1,47,48].

For modularity maximization clustering, as for other clustering problems with an objective function additive over the clusters, the columns correspond to all subsets of V, i.e., to all nonempty modules.

To express this problem, define $a_{it}=1$ if vertex i belongs to module t and to $a_{it}=0$ otherwise. One can then write the model as

$$\max \sum_{t \in T} c_t z_t - C, \tag{6}$$

$$\text{s.t.} \quad \sum_{t \in T} a_{it} z_t = 1 \quad \forall\, i = 1, \ldots, n, \tag{7}$$

$$z_t \in \{0,1\} \quad \forall\, t \in T, \tag{8}$$

where $c_t=\Sigma_i \Sigma_{j>i} w_{ij} a_{it} a_{jt}$, i.e., the value of the module indexed by t with $t=1 \ldots 2^n-1$.

The objective function (6) of the primal problem expresses that modularity is equal to the sum of modularities of all selected modules minus a constant corresponding to the diagonal terms. The first set of constraints [Eq. (7)] expresses that each entity must belong to one and only one module and the second set of constraints that modules must be selected entirely or not at all.

If the integrality constraints [Eq. (8)] are replaced by

$$z_t \geq 0, \quad \forall\, t \in T, \tag{9}$$

the upper bound $z_t \leq 1$ being implied by constraint [Eq. (7)], one obtains a *relaxation* of Eqs. (6)–(8) which is a linear program.

Recall that to any primal linear program is associated another linear program called its dual. This dual program has as many variables as the primal has constraints and as many constraints as the primal has variables.

The dual of the relaxation of the problem in Eqs. (6), (7), and (9) can be written

$$\min \sum_{i=1}^{n} \lambda_i - C, \tag{10}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_{it}\lambda_i \geq c_t \quad \forall\, t \in T, \tag{11}$$

$$\lambda_i \in \mathbb{R} \quad \forall\, i = 1 \ldots n. \tag{12}$$

The objective function (10) of the *dual* problem [Eqs. (10)–(12)] is equal to the sum of all dual variables minus a constant C. The constraints in Eq. (11) express that the sum of dual variables associated with the entities of any community must be at least as large as its modularity. Finally, the constraints in Eq. (12) express the fact that the dual variables are unrestricted in sign.

From the duality theorem of linear programming, the optimal solutions $(z_1^*, z_2^*, \ldots, z_T^*)$ of the primal and $(\lambda_1^*, \lambda_2^*, \ldots, \lambda_n^*,)$ of the dual have the same value,

$$\sum_{t \in T} c_t z_t^* = \sum_{i=1}^{n} \lambda_i^*. \tag{13}$$

Moreover, any feasible solution of the primal has smaller or equal value than any solution of the dual. Geometrically, the sets of feasible solutions of the primal and dual are polyhedra. Finding the optimal solution of the primal or of the dual amounts to finding an optimal vertex and the corresponding cone in either of these polyhedra.

Problem described by Eqs. (6), (7), and (9) is called the *master problem*. It is the relaxation of a partitioning problem which can in principle be solved by a package such as CPLEX. However, as the number of columns is exponential, this is possible only for very small n. To overcome these difficulties, one resorts to column generation. Following that approach a *reduced master problem* with considerably fewer columns is solved instead of the full master problem [Eqs. (6), (7), and (9)]. As usual in the branch and bound approach to mixed integer programming, one first solves the continuous relaxation of this restricted master problem. One begins with a relaxed problem containing some feasible columns and possibly artificial variables. Then improving columns will be added progressively. Finding such column(s) is the *auxiliary problem* whose role is to find a column with positive (negative) reduced cost in case of maximization (minimization). For problem [Eqs. (6), (7), and (9)], the reduced cost associated with column t will be equal to $c_t - \Sigma_i \lambda_i a_{it}$ where the λ_i are the current values of the dual variables of the continuous relaxation of problem [Eqs. (6), (7), and (9)]. Replacing the coefficients a_{it} by binary variables y_i leads to the following expression of the auxiliary problem,

$$\max_{y \in \mathbb{B}^n} \sum_i \sum_{j>i} w_{ij} y_i y_j - \sum_i \lambda_i y_i.$$

This is a quadratic program in 0–1 variables with a 100% dense matrix of coefficients. Many algorithms and numerous heuristics have been proposed to solve it. In our experiments, we use a Variable Neighborhood Search (VNS) heuristic [49,50] as long as it can find a column with positive reduced cost. VNS is a metaheuristic, i.e., a framework for building heuristics, based on the idea of systematic change of neighborhood during the search. It explores progressively larger neighborhoods of the incumbent (or best known) solution in a probabilistic way. Therefore, often favorable characteristics of the incumbent will be kept and used to obtain promising neighboring solutions. VNS applies a local search routine repeatedly to get from these neighboring solutions to local optima.

When VNS fails to find an improving column, we use as exact method a simple branch and bound algorithm [51] or a recent algorithm using bounds based on semidefinite programming [52].

It is well known that column generation algorithms suffer from slow convergence particularly when the optimal solution is *degenerate*, i.e., when such a solution has many variables equal to 0, which is the case for clustering problems. Column generation algorithms also suffer from the plateau effect, i.e., the optimal solution keeps the same value for several or many iterations [53].

To alleviate these defects, one can use a variant of the stabilization methods for column generation due to du Merle *et al.* [54], which we call *focussed column generation*.

The principle is to identify, from a heuristic solution, a small region in the space of dual variables hopefully containing the optimal solution $(\lambda_1^*, \lambda_2^*, \ldots, \lambda_n^*,)$. Then departures from this zone are penalized. To that effect, we first seek a good heuristic solution of the modularity maximization problem. Recently, Noack and Rotta [55] compared experimentally codes for eight heuristics. We used the SS+ML heuristic which is the best according to their experiments, i.e., Single-Step greedy coarsening by Significance with Multi-Level Fast Greedy refinement (SS+ML). This heuristic solution can be further improved by a local application of modularity maximization to each pair of modules at a time [42]. Then, a dual solution is derived from this last heuristic primal solution, and intervals hopefully containing the optimal values are determined. This is done by computing the increase (respectively, the decrease) of the objective function value if a vertex is duplicated (respectively, removed). Penalties for getting out of these intervals are imposed and progressively diminished until they get down to 0. Details on this method are given in [54].

III. MODULARITY MAXIMIZATION BY MIXED 0–1 QUADRATIC PROGRAMMING

A. Direct formulation

Maximizing modularity by the clique partitioning approach discussed in Sec. II has a drawback: it replaces a usually sparse matrix of coefficients by a 100% dense one. An alternative approach is to work directly with a graph $G = (V, E)$ instead of the complete graph K_n. This was done by Xu, Tsoka, and Papageorgiou [41] and leads to a 0–1 mixed integer quadratic problem whose continuous relaxation is convex, and which can therefore be solved by CPLEX. We next recall the main elements of Xu *et al.*'s model as they provide the necessary background for a new column generation algorithm described in the second part of this section. Considering again the definition of Q as a sum over modules of their modularities rewrite Q as

$$Q = \sum_s [a_s - e_s] = \sum_s \left[\frac{m_s}{m} - \left(\frac{D_s}{2m} \right)^2 \right], \tag{14}$$

where m_s denotes the number of edges in module s, i.e., the subgraph induced by the set of vertices $V_s \subset V$ and D_s denotes the sum of degrees k_i of the vertices of module s. Binary variables are then used to identify the modules to which each vertex and each edge belongs. To this effect, list all edges with a single index $r=1,2,\ldots,m$. Then, introduce the following variables:

$$X_{rs} = \begin{cases} 1 & \text{if edge } r \text{ belongs to module } s \\ 0 & \text{otherwise,} \end{cases}$$

for $r=1,2,\ldots m$ and $s=1,2,\ldots M$ and

$$Y_{is} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to module } s \\ 0 & \text{otherwise.} \end{cases}$$

The number of edges and sum of vertex degrees can then be expressed as

$$m_s = \sum_r X_{rs}$$

and

$$D_s = \sum_i k_i Y_{is}.$$

A second series of constraints express that each vertex belongs to exactly one module,

$$\sum_s Y_{is} = 1 \quad \forall\, i = 1,2,\ldots n.$$

A third series of constraints express that any edge $r=\{v_i,v_j\}$ with end vertices indiced by i and j can only belong to module s if both of those end vertices belong to that module.

$$X_{rs} \le Y_{is} \quad \forall\, r=\{v_i,v_j\} \in E,$$

$$X_{rs} \le Y_{js} \quad \forall\, r=\{v_i,v_j\} \in E.$$

Note that these constraints are part of Fortet's [56] linearization of quadratic 0–1 programs. They impose that $X_{rs}=0$ if either Y_{is} or Y_{js} or both are equal to 0. They do not impose that X_{is} is equal to 1 if $Y_{is}=Y_{js}=1$. Although the constraints $X_{rs} \ge Y_{is}+Y_{js}-1$ could be added to impose that, this is not necessary as adding an edge between vertices i and j when they are in the same module increases m_s and hence Q.

The number of modules is *a priori* unknown; indicator variables $u_s=1$ if module s is nonempty and $u_s=0$ otherwise are used. Then constraints

$$u_s \le u_{s-1} \quad \forall\, s \in 2,3,\ldots S,$$

where S is an upper bound on the number of modules, are added and express that module s can be nonempty only if module $s-1$ is so.

Consequently,

$$\sum_r X_{rs} \ge u_s$$

and

$$\sum_r X_{rs} \le (n-s+1)u_s.$$

The value $n-s+1$ in the latter constraints is due to the fact that each of the modules $1,2,\ldots s-1$ must be nonempty. In fact, Xu *et al.* use more general parametric formulas which allow imposing lower and upper bounds on the cardinality of the modules.

It is well-known that for any given solution to a clustering problem, alternative equivalent solutions can be obtained by simply reindexing clusters. For an optimal solution with M modules, $M!$ equivalent solutions exist. Symmetry-breaking constraints for clustering problems can be found in the literature [57–59], and were used in [41]. Such symmetry breaking constraints notwithstanding, this problem has $M(n+m)$ binary variables and M continuous variables, subject to $M(1+2m+2n)$ linear constraints.

B. Column generation reformulation

Once again, a linear programming master problem will be solved by the simplex algorithm where the entering column will be determined by solving an auxiliary problem. The master problem will be the same as in the previous column generation algorithm, i.e., its equations are given in Eqs. (6), (7), and (9) of Sec. II B. The auxiliary problem will be different from the quadratic 0–1 program used in the clique partitioning formulation. It will be close to the formulation of Xu *et al.* [41] described in Sec. III A, but much simpler. As a single community is to be determined at a time, it can be written as follows:

$$\max_{x \in \mathbb{B}^n, D \in \mathbb{R}} \sum_r \frac{x_r}{m} - \left(\frac{D}{2m} \right)^2 - \sum_i \lambda_i y_i,$$

$$\text{s.t.} \quad D = \sum_i k_i y_i,$$

$$x_r \le y_i \quad \forall\, r=\{i,j\} \in E,$$

$$x_r \le y_j \quad \forall\, r=\{i,j\} \in E.$$

As before, edges are indexed by r and vertices by i (or j). Variable x_r is equal to 1 if edge r belongs to the community which maximizes the objective function and to 0 otherwise. Similarly, y_i is equal to 1 if the ith vertex belongs to the community and 0 otherwise. The objective function is equal to the modularity of the community to be determined minus the scalar product of the current value λ_i of the dual variables times the indicator variable y_i. Observe that this last term is the same as in the objective function of the auxiliary problem for clique partitioning. This is a mixed integer quadratic problem with $n+m$ binary variables and 1 continuous variable, in the objective function, subject to $2m+1$ linear con-

TABLE I. Name, order and size of 11 test problems. All networks are undirected, unweighted and without loops.

Problem ID	Name	n	m
1	Zachary's karate club	34	78
2	Dolphins social network	62	159
3	Les Misérables	77	254
4	A00_main	83	135
5	Protein p53	104	226
6	Books about U.S. politics	105	441
7	American College Football	115	613
8	A01_main	249	635
9	USAir97	332	2126
10	Netscience_main	379	914
11	Electronic Circuit (s838)	512	819

straints. In the objective function there is a single concave nonlinear term. Clearly, the size of this auxiliary problem is much smaller than that of the direct formulation described in Sec. III A, particularly for large number of communities M. This auxiliary problem is first solved with a VNS heuristic as long as a column with a positive reduced cost can be found. When this is no more the case, CPLEX is called to find such a column or prove that there are no more.

IV. COMPUTATIONAL COMPARISON

To compare the four algorithms described in the previous sections, we selected 11 test problems from the modularity maximization literature. Their names, orders and sizes are given in Table I.

The data of these test problems can be found in various databases mentioned in a recent paper from Noack and Rotta. Note that we always assume unit edge weights.

Results are given in Table II. The first column gives the problem ID as in Table I, the next six columns give the modularity value (Q) and the number of modules (M) for: (a) the solution obtained by Noack-Rotta's heuristic (NR Sol); (b) the improved solution obtained from Noack-Rotta's solution (Imp Sol); (c) the optimal solution (Opt Sol).

The remaining columns summarize the performance of the different exact algorithms:

(i) *CPRG*: clique partitioning row generation of [39];

(ii) *CPCG-HJM*: clique partitioning column generation with the exact method of [51] for the auxiliary problem;

(iii) *CPCG-BE*: clique partitioning column generation with the exact method of [52] for the auxiliary problem;

(iv) *0–1 MIQP*: 0–1 mixed integer programming formulation of [41];

(v) *0–1 MICG*: 0–1 mixed integer column generation.

For *CPRG* column, the CPU time is reported when possible or an "OM" is used to indicate that the program reaches the memory limit. The formulation of [39] is implemented in AMPL [60] and solved using the "lazy constraints" feature of CPLEX. We do not present the results for the formulation of [16] as it is much more time consuming. For example, the solution of Problem 1 takes 674 s instead of 0.23 s.

For the *0–1 MIQP* column, we give the CPU time reported in [41] or indicate by "NA" the problem not considered in [41]. It was not possible for us to obtain comparable

TABLE II. Results of the comparison between algorithms for modularity maximization, in terms of CPU time, on the 11 test problems described in Table I. These algorithms are: the clique partitioning row generation of [39] (*CPRG*), the clique partitioning column generation with the exact method of [51] for the auxiliary problem (*CPCG-HJM*), the clique partitioning column generation with the exact method of [52] for the auxiliary problem (*CPCG-BE*), the 0–1 mixed integer programming formulation of [41] (*0–1 MIQP*), the 0–1 mixed integer column generation (*0–1 MICG*). In parentheses, the number of calls to the exact algorithm for the auxiliary problem and the number of calls to the heuristic for the same purpose are shown. NR Sol, Imp Sol, and Opt Sol denote, respectively, the solution obtained by Noack-Rotta's heuristic, the improved solution obtained from that heuristic solution [42] and the optimal solution. Q and M are the modularity value and the number of modules. Results for *CPRG* were known for problems 1–2-3–6-7 [13], as well as results for *0–1 MIQP* for problems 1–2-3–5 [41]. The other results are new and show superiority of column generation. Best results are reported in boldface. OM: memory limit attained (on a computer with a 3GB RAM); OT: optimal solution not found after 100 000 s; NA: test problem not considered.

Pb.	NR Sol		Imp Sol		Opt Sol		*CPRG*	*CPCG-*				*0–1 MIQP*	*0–1 MICG*	
ID	Q	M	Q	M	Q	M		*HJM*		*BE*				
1	0.4198	4	0.4198	4	0.4198	4	0.23	**0.18**	(3/7)	0.73	(3/7)	1.03	0.34	(3/8)
2	0.5238	4	0.5285	5	0.5285	5	9.70	**7.49**	(1/13)	18.69	(1/13)	197.89	7.75	(1/13)
3	0.5600	6	0.5600	6	0.5600	6	**5.07**	6.89	(1/13)	12.92	(1/13)	55.58	7.26	(1/13)
4	0.5251	7	0.5283	9	0.5309	9	17.09	**3.57**	(1/26)	262.13	(1/26)	NA	3.66	(1/26)
5	0.5322	7	0.5350	6	0.5351	7	4164.15	**11.33**	(1/44)	351.87	(1/44)	1844.31	11.60	(1/44)
6	0.5269	4	0.5272	5	0.5272	5	663.05	318.99	(2/13)	32287.23	(2/13)	NA	**45.65**	(2/13)
7	0.6002	10	0.6046	10	0.6046	10	282.58	OT		593.01	(1/8)	NA	**249.41**	(1/8)
8	0.6203	12	0.6203	12	0.6329	14	OM	OT		OT		NA	**1014.48**	(12/145)
9	0.3658	6	0.3660	6	0.3682	6	OM	OT		OT		NA	**16216.77**	(3/104)
10	0.8474	19	0.8485	19	0.8486	19	OM	OT		OT		NA	**1615.14**	(12/66)
11	0.8162	16	0.8166	16	0.8194	12	OM	OT		OT		NA	**7655.56**	(140/225)

computing time as the parameters setting used to obtain these results were not detailed in the paper.

For the column generation algorithms, the table contains CPU time as well as, in parentheses, the number of calls for the exact algorithm for the auxiliary problem followed by the number of calls to the heuristic. The term "OT" means that the optimal solution could not be found after more than 100 000 s.

All results are in seconds of CPU. Except for *0–1 MIQP*, all results were obtained on a dual processor computer Intel Pentium computer with 3.20 GHz, 2 Mb cache memory, 3 GB RAM running under Linux. ILOG CPLEX 10.110 was used for the linear programming part of all algorithms.

From Table II it appears that:

(i) Surprisingly, in view of the many heuristics proposed for modularity maximization, those applied here seldom reach the optimal value: in 2 cases out of 11 of Noack and Rotta and 5 cases out of 11 for the improved heuristic;

(ii) Both row generation and column generation algorithms based on reformulation of modularity maximization as a clique partitioning problem are competitive for small instances, but become too time or memory consuming for larger ones due to the rapid increase in the number of variables and constraints;

(iii) For the largest instances, the *0–1 MICG* algorithm appears to be the only one able to find an optimal solution;

(iv) The reduction in resolution times in comparison of the results of [41] are very substantial: the use of *0–1 MICG* algorithm divides these times by a factor of 3 to 159;

(v) The size of the largest problem solved is raised from 105 entities to 512 entities. Larger problems could not be solved because of memory or time limit reached by all the considered algorithms.

In summary, the column generation algorithm reformulated from [41] direct formulation (*0–1 MICG*) appears to be the best choice since its computing time is comparable for small instances, lower for medium instances and is the only algorithm able to solve large instances to optimality.

We note that the four exact algorithms that we compared provided the same partitions on all the tested data sets. This does not imply that alternate optimal partitions do not exist, but should they do they are not likely to be very numerous.

V. CONCLUSIONS

In this paper, we have studied exact maximization of the modularity of a network according to the definition of Newman and Girvan. Two approaches were proposed previous to this work: on the one hand the row generation algorithm of [39] to which modularity maximization can be reduced and on the other hand the direct formulation of Xu *et al.* [41]. We have proposed column generation algorithms based on these two approaches and performed a computation comparison on a series of well-known problems from the literature. These results show that: (i) while row generation is fast for small problems, it is outperformed by column generation for larger ones; (ii) the column generation algorithm based on direct formulation performs best for the larger problems; (iii) size of the larger problems has substantially augmented: the largest problem solved to date has 512 entities versus 105 entities before; (iv) computation times are substantially reduced. While it would of course be desirable to solve exactly larger problems, the results obtained by the proposed algorithm already appear to be useful for the several reasons detailed in the introduction.

It is easy to see that the first two approaches can also be applied to several variants of the standard modularity definition, i.e., weighted networks, directed networks, and networks which are both directed and weighted [61]. Moreover, the column generation framework is easily adaptable to yield exact algorithms for other definitions of communities than those of Newman and Girvan, allowing comparison which do not depend on the specific heuristic used.

Among topics for future research are (i) the design of better heuristics, (ii) the use of cutting planes in the solution of the master problem and/or the auxiliary problem, (iii) a further study of the properties of the auxiliary problem which might lead to improvements in its resolution. The importance of this last point stems from the fact that the proportion of the resolution time devoted to the resolution of the auxiliary problem tends to increase with problem size.

ACKNOWLEDGMENTS

Financial support by Grants ANR 07-JCJC-0151 "ARS," Digiteo 2009-14D "RMNCCO," Digiteo 2009-55D "ARM," are gratefully acknowledged. P.H. acknowledges support from Digiteo Foundation. S.P. has been supported by the NSERC (Natural Sciences and Engineering Research Council of Canada) Grant No. 327435-06. G.C., P.H., and S.P. would like to thank the research office of HEC Montreal for support of their participation to the 2009-2010 research workshop.327435-06

[1] P. Hansen and B. Jaumard, Math. Program. **79**, 191 (1997).

[2] A. Jain, M. Murty, and P. Flynn, ACM Comput. Surv. **31**, 264 (1999).

[3] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley Series in Probability and Statistics (Wiley, New York, 2005).

[4] B. Mirkin, *Clustering for Data Mining: A Data Recovery Approach* (Chapman and Hall; CRC, Boca Raton, FL, 2005).

[5] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, Proc. Natl. Acad. Sci. U.S.A. **101**, 2658 (2004).

[6] Z. Li, S. Zhang, R.-S. Wang, X.-S. Zhang, and L. Chen, Phys. Rev. E **77**, 036109 (2008).

[7] A. D. Medus and C. O. Dorso, Phys. Rev. E **79**, 066111 (2009).

[8] S. Cafieri, P. Hansen, and L. Liberti, Phys. Rev. E **81**, 026105 (2010).

[9] M. Newman and M. Girvan, Phys. Rev. E **69**, 026113 (2004).
[10] C. P. Massen and J. P. K. Doye, Phys. Rev. E **71**, 046101 (2005).
[11] S. Fortunato and M. Barthelemy, Proc. Natl. Acad. Sci. U.S.A. **104**, 36 (2007).
[12] X. S. Zhang, R. S. Wang, Y. Wang, J. Wang, Y. Qiu, L. Wang, and L. Chen, EPL **87**, 38002 (2009).
[13] S. Cafieri, P. Hansen, and L. Liberti, Phys. Rev. E **81**, 046102 (2010).
[14] J. M. Kumpula, J. Saramäki, K. Kaski, and J. Kertész, Fluct. Noise Lett. **7**, L209 (2007).
[15] J.-G. Wang, L. Wang, Y.-Q. Qui, Y. Wang, and X.-S. Zhang, in *Proceedings of the Third International Symposium on Optimization and Systems Biology*, Zhangjiajie, China, Vol. 11 of Lecture Notes in Operations Research, edited by L. Chen, X.-S. Zhang, L.-Y. Wu, and Y. Wang (World Publishing Corp., Beijing, 2009), pp. 142–150.
[16] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, IEEE Trans. Knowl. Data Eng. **20**, 172 (2008).
[17] M. E. J. Newman, Phys. Rev. E **69**, 066133 (2004).
[18] A. Clauset, M. E. J. Newman, and C. Moore, Phys. Rev. E **70**, 066111 (2004).
[19] L. Danon, A. Diaz-Guilera, and A. Arenas, J. Stat. Mech.: Theory Exp. (2006) P11010.
[20] K. Wakita and T. Tsurumi, e-print arXiv:cond-mat/0702048.
[21] V. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech.: Theory Exp. (2008) P10008.
[22] R. Guimerà and L. A. Nunes Amaral, Nature (London) **433**, 895 (2005).
[23] A. Medus, G. Acuna, and C. Dorso, Physica A **358**, 593 (2005).
[24] S. Lehmann and L. Hansen, Eur. Phys. J. B **60**, 83 (2007).
[25] M. Tasgin, A. Herdagdelen, and H. Bingol, e-print arXiv:0711.0491.
[26] J. Duch and A. Arenas, Phys. Rev. E **72**, 027104 (2005).
[27] M. E. J. Newman, Proc. Natl. Acad. Sci. U.S.A. **103**, 8577 (2006).
[28] T. Richardson, P. J. Mucha, and M. A. Porter, Phys. Rev. E **80**, 036111 (2009).
[29] Y. Sun, B. Danila, K. Josic, and K. E. Bassler, EPL **86**, 28004 (2009).
[30] G. Agarwal and D. Kempe, Eur. Phys. J. B **66**, 409 (2008).
[31] S. Boccaletti, M. Ivanchenko, V. Latora, A. Pluchino, and A. Rapisarda, Phys. Rev. E **75**, 045102(R) (2007).
[32] H. N. Djidjev, Lect. Notes Comput. Sci. **4936**, 117 (2008).
[33] J. Mei, S. He, G. Shi, Z. Wang, and W. Li, New J. Phys. **11**, 043025 (2009).
[34] P. Schuetz and A. Caflisch, Phys. Rev. E **77**, 046112 (2008).
[35] Y. Niu, B. Hu, W. Zhang, and M. Wang, Physica A **387**, 6215 (2008).
[36] D. Chen, Y. Fu, and M. Shang, Physica A **388**, 2741 (2009).
[37] J. Ruan and W. Zhang, Phys. Rev. E **77**, 016104 (2008).
[38] Y. Fan, M. Li, P. Zhang, J. Wu, and Z. Di, Physica A **377**, 363 (2007).
[39] M. Grötschel and Y. Wakabayashi, Math. Program. **45**, 59 (1989).
[40] M. Grötschel and Y. Wakabayashi, Math. Program. **47**, 367 (1990).
[41] G. Xu, S. Tsoka, and L. Papageorgiou, Eur. Phys. J. B **60**, 231 (2007).
[42] S. Cafieri, P. Hansen, and L. Liberti, Matheuristics 2010 (Vienna, 2010).
[43] ILOG, *ILOG CPLEX 11.0 User's Manual, ILOG S.A* (Gentilly, France, 2008).
[44] G. Dantzig, R. Fulkerson, and S. Johnson, Oper. Res. **2**, 393 (1954).
[45] D. Applegate, R. Bixby, V. Chvátal, W. Cook, D. Espinoza, M. Goycoolea, and K. Helsgaun, Oper. Res. Lett. **37**, 11 (2009).
[46] T. Achterberg, T. Koch, and A. Martin, Oper. Res. Lett. **33**, 42 (2005).
[47] O. du Merle, P. Hansen, B. Jaumard, and N. Mladenovic, SIAM J. Sci. Comput. (USA) **21**, 1485 (1999).
[48] D. Aloise, P. Hansen, and L. Liberti, Math. Program. (to be published).
[49] N. Mladenovic and P. Hansen, Comput. Oper. Res. **24**, 1097 (1997).
[50] P. Hansen and N. Mladenovic, Eur. J. Oper. Res. **130**, 449 (2001).
[51] P. Hansen, B. Jaumard, and C. Meyer, Les Cahiers du GERAD Report No. G-2000–59, 2000 (unpublished).
[52] A. Billionnet and S. Elloumi, Math. Program. **109**, 55 (2007).
[53] F. Vanderbeck and M. Savelsbergh, Oper. Res. Lett. **34**, 296 (2006).
[54] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, Discrete Math. **194**, 229 (1999).
[55] A. Noack and R. Rotta, Lect. Notes Comput. Sci. **5526**, 257 (2009).
[56] R. Fortet, Cahier du centre d'études de recherche opérationnelle **1**, 5 (1959).
[57] G. Klein and J. Aronson, Naval Research Logistics **38**, 447 (1991).
[58] H. Sherali and J. Desai, J. Global Optim. **32**, 281 (2005).
[59] F. Plastria, Eur. J. Oper. Res. **140**, 338 (2002).
[60] R. Fourer and D. Gay, *The AMPL Book* (Duxbury Press, Pacific Grove, 2002).
[61] S. Fortunato, Phys. Rep. **486**, 75 (2010).