Modularity Maximization in Networks by Variable Neighborhood Search

Daniel Aloise[1]*, Gilles Caporossi[2], Pierre Hansen[2,3], Leo Liberti[3], Sylvain Perron[2], and Manuel Ruiz[4]

[1] Dept. of Computer Engineering and Automation, Universidade Federal do Rio Grande do Norte, Campus Universitário s/n, Natal-RN, Brazil, 59072-970
aloise@dca.ufrn.br

[2] GERAD & HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal (Québec) Canada, H3T 2A7 {gilles.caporossi, pierre.hansen, sylvain.perron}@hec.ca

[3] LIX, École Polytechnique, F-91128 Palaiseau, France,
liberti@lix.polytechnique.fr

[4] INP-Grenoble, 46, avenue Félix Viallet, 38031 Grenoble Cedex 1, France
manuel.ruiz@grenoble-inp.fr

Abstract. Finding communities, or clusters, in networks, or graphs, has been the subject of intense studies in the last ten years. The most used criterion for that purpose, despite some recent criticism, is modularity maximization, proposed by Newman and Girvan. It consists in maximizing the sum for all clusters of the number of inner edges minus the expected number of inner edges assuming the same distribution of degrees. Numerous heuristics, as well as a few exact algorithms have been proposed to maximize modularity. We apply the Variable Neighborhood Search metaheuristic to that problem. The resulting Variable Neighborhood Decomposition Search heuristic is applied to the Clustering problems of the 10^{th} DIMACS Implementation Challenge. The optimal solution is obtained by the heuristic whenever it is known, and otherwise the best solutions from the literature are improved, except in the case of three instances for which available memory was too small.

Keywords: modularity, community, clustering

1 Introduction

Clustering is an important chapter of data analysis and data mining with numerous applications in natural and social sciences as well as in engineering and medicine. 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 [19, 21, 22, 30]. These problems in turn are solved by exact algorithms or, more often and particularly for large

* corresponding author

data sets, by heuristics, of which there are frequently a large variety. 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, or graphs, has been extensively studied, mostly in the physics and computer science communities, with recently a few forays from operations research. Rather than using the term cluster, the words *module* or *community* are often adopted in the physics literature. We use below the standard notation and terminology for graphs, i.e, a graph $G = (V, E, \omega)$ is composed of a set V of n vertices v_j and a set E of m edges $e_{ij} = \{v_i, v_j\}$. These edges may be weighted by the function $\omega(\{u, v\})$. If they are unweighted $\omega(\{u, v\}) = 1$. A subgraph $G_C = (C, E_C, \omega)$ of a graph $G = (V, E, \omega)$ induced by a set of vertices $C \subseteq V$ is a graph with vertex set C and edge set E_C equal to all edges with both vertices in C. Such a subgraph corresponds to a cluster (or module, or community) and many heuristics aim at finding a partition $\mathcal{C}$ of V into pairwise disjoint nonempty subsets $V_1, V_2, \ldots, V_N$ inducing subgraphs of G and covering V. Various objective functions have been proposed for evaluating such a partition. Among the best known are *multiway cut* [17], *normalized cut* [39], *ratio cut* [3] and *modularity* [33]. Initially proposed by Girvan and Newman in 2002 [16] as a stopping rule for a hierarchical divisive heuristic, modularity was considered later as an independent criterion allowing determination of optimal partitions as well as comparison between partitions obtained by various methods.

Modularity aims at finding 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. The following precise definition of *modularity* is given in [33]:

$$Q = \sum_{C \in \mathcal{C}} \left[a_C - e_C \right],$$

where a_C is the fraction of all edges that lie within module C and e_C 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.

Let the vertex function weight function be defined as:

$$\omega(v) = \begin{cases} \displaystyle\sum_{\{u,v\} \in E} \omega(\{u, v\}) & \text{if } \{v, v\} \notin E \\ \displaystyle\sum_{\{u,v\} \in E, u \neq v} \omega(\{u, v\}) + 2\omega(\{v, v\}) & \text{if } \{v, v\} \in E. \end{cases}$$

Let $\mathcal{C}$ be a partition of V. The sum over modules of their modularities can be written as

$$Q = \frac{\sum_{C \in \mathcal{C}} \sum_{\{u,v\} \in E_C} \omega(\{u,v\})}{\sum_{e \in E} \omega(e)} - \frac{\sum_{C \in \mathcal{C}} \left(\sum_{v \in V_C} \omega(v) \right)^2}{4 \left(\sum_{e \in E} \omega(e) \right)^2}. \tag{1}$$

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 [5, 9, 10, 31, 42], simulated annealing [18, 27, 28], mean field annealing [24], genetic search [41], extremal optimization [13], label propagation [4, 26], spectral clustering [32, 36, 40], linear programming followed by randomized rounding [1], dynamical clustering [6], multilevel partitioning [12], contraction-dilation [29], multistep greedy search [38], quantum mechanics [34] and other approaches [5, 8, 14, 23, 37, 40]. For a more detailed survey, see [15].

The paper is organized as follows: in Section 2, after giving an outline of the variable neighborhood search metaheuristic, we discuss its application to modularity maximization. In Section 3, we recall and extend to the weighted case an exact method for modularity maximization. Experimental results are presented in Section 4 in two tables corresponding to the results for Pareto and Quality challenges respectively. Brief conclusions are drawn in the last section.

2 Description of the heuristic

2.1 Outline of the variable neighborhood search metaheuristic

Variable Neighborhood Search (VNS) is a metaheuristic, or framework for building heuristics, aimed at solving combinatorial and global optimization problems. Since its inception, VNS has undergone many developments and has been applied in numerous fields (see [20] for a recent survey).

Metaheuristics address the problem of escaping, as much as possible, from local optima. A local maximum x_L of an optimization problem is such that

$$f(x_L) \geq f(x), \forall x \in N(x_L) \tag{2}$$

where $N(x)$ denotes the feasible *neighborhood* of x, which can be defined in many different ways each one yielding a different neighborhood structure. In discrete optimization problems, a neighborhood structure consists of all vectors obtained from x by some simple modification. For instance, for x binary, one neighborhood structure can be defined by the set of all vectors obtained from x by complementing one of its components. Another possible neighborhood structure can be defined as the set of all vectors obtained from x by complementing two complementary components of x (i.e., one component is set from 0 to 1 and the other goes from 1 to 0). A *local search* or *improving* heuristic consists of choosing an initial solution x, and then moving to the best neighbor $x' \in N(x)$ in the case $f(x') > f(x)$. If no such neighbor exists, the heuristic stops, otherwise it is iterated.

If many local maxima exist for a problem, the range of values they span may be large. Moreover, the globaly optimum value $f(x^*)$ may differ substantially from the average value of a local maximum, or even from the best such value among many, obtained by some simple randomized heuristic. In order to escape from local maxima and, more precisely, the mountains of which they are the top, VNS exploits the idea of *neighborhood* change. In fact, VNS relies upon the following observations:

Fact 1: *A local maximum with respect to one neighborhood structure is not necessarily so for another*;

Fact 2: *A global maximum is a local maximum with respect to all possible neighborhood structures*;

Fact 3: *For many problems local maxima with respect to one or several neighborhoods are relatively close to each other.*

Let us denote with $\mathcal{N}_t$, $(t = 1, \ldots, t_{max})$, a finite set of pre-selected neighborhood structures, and with $\mathcal{N}_t(x)$ the set of solutions in the t^{th} neighborhood of x. We call x a *local maximum* with respect to $\mathcal{N}_t$ if there is no solution $x' \in \mathcal{N}_t(x)$ such that $f(x') > f(x)$.

In the VNS framework, the neighborhoods used correspond to various types of moves, or perturbations, of the current solution, and are problem specific. The current best solution x found is the center of the search. When looking for a better one, a solution x' is drawn at random in an increasingly far neighborhood and a local ascent is performed from x', leading to another local maximum x''. If $f(x'') \leq f(x)$, x'' is ignored and one chooses a new neighbor solution x' in a further neighborhood of x. If, otherwise, $f(x'') > f(x)$, the search is re-centered around x'' restarting with the closest neighborhood. If all neighborhoods of x have been explored without success, one begins again with the closest one to x, until a stopping condition (e.g. maximum CPU time) is satisfied.

As the size of neighborhoods tends to increase with their distance from the current best solution x, close-by neighborhoods are explored more thoroughly than far away ones. This strategy takes advantage of the three Facts 1–3 mentioned above. Indeed it is often observed that most or all local maxima of combinatorial problems are concentrated in a small part of the solution space. Thus, finding a first local maximum x implies that some important information has been obtained: to get a better, near-optimal solution, one should first explore its vicinity.

The algorithm proposed in this work has two main components: (i) an improvement heuristic, and (ii) exploration of different types of neighborhoods for getting out of local maxima. They are used within a variable neighborhood decomposition search framework [20] which explores the structure of the problem concentrating on small parts of it. The basic components as well as the decomposition framework are described in the next sections.

2.2 Improvement heuristic

The improvement heuristic we used is the LPAm+ algorithm proposed by Liu and Murata in [26]. LPAm+ is composed of a label propagation algorithm proposed by Barber and Clark [4] and a community merging routine. A strong feature of this heuristic is that label propagation executes in near linear time (in fact, each iteration of label propagation executes in time proportional to m), while one round of merging pairs of communities can execute in $O(m \log n)$ [38].

From [4], the authors show that when evaluating a vertex, the candidates labels (clusters) can be confined to those of its neighbors and an unused label. With that in mind, we decided to evaluate "labels" instead of vertices. We used LPAm+ modified as follows. A list L of all labels is initialized with all the used labels. Then, from L, we proceed by picking a label $\ell \in L$ until L is empty. Each time a label $\ell \in L$ is picked, we evaluate all its vertices for label updating. If a label is updated, i.e., an improvement in modularity is found, the old and new labels as well as their neighboring labels are put in L.

This modification induces a considerable algorithmic speed-up since only a few clusters need to be evaluated as the algorithm proceeds while most of the other clusters remain fixed.

We then tested this modified LPAm+, and proceeded to improve it based on empirical observations. In the final version, whenever a vertex relabeling yields an improvement, the old and new labels are added to L but only together with the labels of vertices which are adjacent to the relabeled vertex. This version was selected to be used in our experiments due to its benefits in terms of computing times and modularity maximization.

2.3 Neighborhoods for perturbations

In order to escape from local maxima, our algorithm uses five distinct neighborhoods for perturbing a solution. They are:

1. SINGLETON: all the vertices in a cluster are made singleton clusters.
2. DIVISION: splits a community into two equal parts. Vertices are assigned to each part randomly.
3. NEIGHBOR: relabels each vertex of a cluster to one of the labels of its neighbors or to an unused label.
4. FUSION: merges two or more clusters into a single one.
5. REDISTRIBUTION: destroys a cluster and spreads each one of its vertices to a neighboring cluster randomly chosen.

2.4 Variable Neighborhood Decomposition Search

Given the size of the instances proposed in the 10^{th} DIMACS Implementation Challenge, a decomposition framework was used. It allows the algorithm to explore the search space more quickly since just a small part of the solution is searched for improvement at a time. This subproblem is isolated for improvement through selecting a subset of the clusters in the incumbent solution.

The decomposition proposed here is combined with the five neighborhoods presented in the previous section within a variable neighborhood schema. Thus, the decomposition executes over five distinct neighborhood topologies, with subproblems varying their size according to the VNS paradigm. The pseudo-code of the variable neighborhood decomposition search heuristic is given in Figure 1.

```
1  Algorithm VNDS($P$)
2  Construct a random solution $x$ ;
3  $x \leftarrow$ LPAm+($x, P$) ;
4  $s \leftarrow 1$;
5  while stopping condition not satisfied do
6      Construct a subproblem $S$ from $x$ with a randomly selected cluster and $s - 1$
       neighboring clusters ;
7      Select randomly $\alpha \in \{singleton, division, neighbor, fusion, redistribution\}$ ;
8      $x' \leftarrow$ shaking($x, \alpha, S$);
9      $x' \leftarrow$ LPAm+($x', S$) ;
10     if $cost(x') > cost(x)$ then
11         $x \leftarrow$ LPAm($x', P$) ;
12         $s \leftarrow 1$;
13     else
14         $s \leftarrow s + 1$;
15         if $s > \min\{MAX\_SIZE, \#clusters(x)\}$ then
16             $s \leftarrow 1$;
17         end
18     end
19 end
20 return $x$
```

Algorithm 1: Pseudo-code of the decomposition heuristic.

The algorithm VNDS starts with a random solution for an input problem P in line 2. Then, in line 3 this solution is improved by applying our implementation of LPAm+. Note that LPAm+ receives two input parameters, they are: (i) the solution to be improved, and (ii) the space on which an improvement will be searched. In line 3, the local search is applied in the whole problem space P, which means that all vertices are tested for label updating, and all clusters are considered for merging. In line 4, the variable s which controls the current decomposition size is set to 1.

The central part of the algorithm VNDS consists of the loop executed in lines 5-19 until a stopping criterion is met (this can be the number of non improving iterations for the Pareto Challenge or maximum allowed CPU time for the Quality Challenge). This loop starts in line 6 by constructing a subproblem from a randomly selected cluster and $s - 1$ neighboring clusters. Then, in line 7 a neighborhood α is randomly selected for perturbing the incumbent solution x. Our algorithm allows choosing α by specifying a probability distribution on the neighborhoods. Thus, the most successful neighborhoods are more often selected. The shaking routine is actually performed in line 8 in the chosen neigborhood α

and in the search space defined by subproblem S. In the following, the improving heuristic LPAm+ is applied over x' in line 9 only in the current subproblem S. If the new solution x' is better than x, a faster version of the improving heuristic, denoted LPAm, is applied over x' in the whole problem P. In this version, the improving heuristic does not evaluate merging clusters. The resulting solution of LPAm application is assigned to x in line 11 and s is reset to 1 in line 12. Otherwise, if x' is not better than x, the size of the decomposition is increased by one in line 14. This value is reset to 1 in line 16 if it exceeds the minimum between a given parameter MAX_SIZE and the number of clusters (i.e., #clusters(x)) in the current solution x (line 15). Finally, a solution x is returned by the algorithm in line 20.

3 Description of the exact method

Column generation together with branch-and-bound can be used to obtain the optimal partition. Column generation algorithms for clustering 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 and for the problem under study a community which improves the modularity of the current solution. In [2], several stabilized column generation algorithms have been proposed for modularity maximization and compared on a series of well-known problems from the literature. The column generation algorithm based on extending the mixed integer formulation of Xu et al. [43] appears to be the most efficient. We summarize below an adaptation of this algorithm for the case of weighted networks.

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. 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 considered. 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 remains no such column and hence the linear programming relaxation is solved.

For modularity maximization clustering, as for other clustering problems with an objective function additive over the clusters, the columns correspond to the set T of all subsets of V, i.e., to all nonempty modules, or in practice to a subset T' of T. To express this problem, define $a_{it} = 1$ if vertex i belongs to module t and $a_{it} = 0$ otherwise. One can then write the model as

$$\max \sum_{t \in T} c_t z_t \tag{3}$$

$$\text{s.t.} \sum_{t \in T} a_{it} z_t = 1 \qquad \forall i = 1, \ldots, n \tag{4}$$

$$z_t \in \{0, 1\} \qquad \forall t \in T, \tag{5}$$

where c_t corresponds to the modularity value of the module indexed by t with $t = 1 \ldots 2^n - 1$. The problem (3)-(5) is too large to be written explicitly. A reduced problem with few columns, i.e., those with index $t \in T'$, is solved instead. One first relaxes the integrality constraints and uses column generation for solving the resulting linear relaxation.

The auxiliary problem, for the weighted case, can be written as follows:

$$\begin{aligned} \max_{x \in \mathbb{B}^n, D \in \mathbb{R}} \quad & \sum_{e} \frac{x_e}{M} - \left(\frac{D}{2M}\right)^2 - \sum_{u \in V} \lambda_u y_u \\ \text{s.t.} \quad & D = \sum_{u \in V} \omega(u) y_u \\ & x_e \leq y_u \qquad \forall e = \{u, v\} \in E \\ & x_e \leq y_v \qquad \forall e = \{u, v\} \in E \end{aligned}$$

where $M = \sum_{e \in E} \omega(e)$. Variable x_e is equal to 1 if edge e belongs to the community which maximizes the objective function and to 0 otherwise. Similarly, y_u is equal to 1 if the vertex v 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_u of the dual variables times the indicator variables y_u. As in [2], the 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 none remain. If the optimal solution of the linear relaxation is not integer, one proceeds to branching on the condition that two selected entities belong to the same community or to two different ones.

4 Experimental Results

The algorithms were implemented in C++ and compiled by gcc 4.5.2. The instances are taken from the *Clustering* chapter of the 10^{th} DIMACS Implementation Challenge (http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml). Computational experiments were performed on a AMD 64 bits platform with a 3 GHz clock and 8 Gbytes of RAM memory, except for instance `road_central`, which was executed in a Intel X3353 2.6 Ghz platform with 24Gbytes of RAM memory. Instances `road_usa`, `uk-2002` and `uk-2007-05` were not executed due to memory limitations.

Limited computational experiments allowed to set the parameters of the VNDS algorithm as follows:

- MAX_SIZE = 15
- Probability distribution for selecting α is drawn with:
  - 30% of chances of selecting SINGLETON
  - 30% of chances of selecting DIVISION
  - 30% of chances of selecting NEIGHBOR
  - 5% of chances of selecting FUSION
  - 5% of chances of selecting REDISTRIBUTION

The stopping condition in algorithm VNDS was defined depending on the challenge, Pareto or Quality, in which VNDS is used. Thus, the same algorithm is able to compete in both categories by just modifying how it is halted.

4.1 Results for Pareto Challenge

In this challenge, VNDS stops whenever it attains either N iterations without improving the incumbent solution or after 1 hour of CPU. For this challenge we divided the instances into two categories. For the instances in category $P1$, VNDS uses $N = 1000$, while for those in category $P2$, the algorithm uses $N = 100$.

Table 1 shows computational results obtained in five independent runs of algorithm VNDS. The first column refers to the category of the instance indicated in the second column. The third and fourth columns refer to the number of nodes (n) and edges (m) of each instance. The fifth and sixth columns refer to average modularity values and average computing times, respectively. Finally, the seventh and eighth columns present, for the best solution obtained in the five runs, the modularity value and the corresponding number of clusters.

The following observations can be made regarding the results presented in Table 1:

- VNDS finds the optimal solution, proved by the exact method of Section 3, of instances `karate`, `chesapeake`, `dolphins`, `lesmis`, `polbooks`, `adjnoun`, `football`, and `jazz`. Except for instance `adjnoun`, where the optimal solution is found in 2 out of 5 runs, the optimal solutions of the aforementioned instances are obtained in all runs.
- VNDS is stopped due to CPU time limit in the instances for which the average computing time $t_{avg} = 3600.00$.

In particular, the results presented for the instance `road_central` in the next section can also be used in this category of the challenge.

4.2 Results for Quality challenge

Since the amount of work to compute a solution is not taken into consideration for this challenge, the VNDS algorithm was allowed to run for a longer period of time than before, the CPU time limit being the unique stopping condition. In

Table 1. Results for the Pareto Challenge in the Clustering instances of the 10^{th} DIMACS Implementation Challenge.

category	instance	n	m	Q_{avg}	t_{avg}	Q_{best}	$\|\mathcal{C}\|_{best}$
P1	`karate`	34	78	0.419790	0.00	0.419790	4
	`chesapeake`	39	170	0.265796	0.00	0.265796	3
	`dolphins`	62	159	0.528519	0.00	0.528519	5
	`lesmis`	77	254	0.566688	0.00	0.566688	6
	`polbooks`	105	441	0.527237	0.00	0.527237	5
	`adjnoun`	112	425	0.312268	0.09	0.313367	7
	`football`	115	613	0.604570	0.00	0.604570	10
	`jazz`	198	2742	0.445144	0.01	0.445144	4
	`celegansneural`	297	2148	0.503485	0.02	0.503485	6
	`celegans_metabolic`	453	2025	0.452713	0.91	0.453209	9
	`e-mail`	1133	5451	0.582575	3.49	0.582769	10
	`polblogs`	1490	16715	0.427105	0.10	0.427105	278
	`netscience`	1589	2742	0.959900	0.02	0.959900	407
	`power`	4941	6594	0.939356	2.45	0.940615	44
	`hep-th`	8361	15751	0.855869	25.48	0.856486	1378
	`PGPgiantcompo`	10680	24316	0.885538	44.95	0.885695	113
	`cond-mat`	16706	121251	0.852140	123.83	0.852779	1262
P2	`astro-ph`	16726	47594	0.740415	20.95	0.742530	1086
	`as-22july06`	22963	48436	0.676344	53.44	0.677174	46
	`cond-mat-2003`	31163	120029	0.771163	51.60	0.772827	1703
	`cond-mat-2005`	40421	175691	0.740288	174.45	0.742244	1927
	`preferentialAttachment` [5]	100000	499985	0.314008	1431.27	0.315957	9
	`smallworld`	100000	499998	0.791888	63.33	0.792505	256
	`G_n_pin_pout`	100000	501198	0.491101	3005.74	0.495370	135
	`caidaRouterLevel`	192244	609066	0.867855	2160.12	0.868722	513
	`coAuthorsCiteseer`	227320	814134	0.901568	1847.30	0.902181	320
	`citationCiteseer`	268495	1156647	0.819532	3600.00	0.819964	228
	`coAuthorsDBLP`	299067	977676	0.831789	3600.00	0.832233	542
	`cnr-2000`	325557	2738969	0.912926	1282.92	0.912929	474
	`coPapersCiteseer`	434102	16036720	0.917655	3600.00	0.917741	465
	`coPapersDBLP`	540486	15245729	0.857130	3600.00	0.857700	385
	`eu-2005`	862664	16138468	0.940836	3600.00	0.941250	438
	`in-2004`	14081816	16933413	0.980278	3600.00	0.980345	1702

our set of experiments, the instances were split into three different categories. The algorithm was allowed to run for 180 seconds (3 minutes) for instances in category Qu1, 1800 seconds (30 minutes) for instances in category Qu2, 10800 seconds (3 hours) for instances in category Qu3, and 36000 seconds (10 hours) for the instance in category Qu4 (i.e., `road_central`).

Table 2 presents the computational results obtained in five independent runs of algorithm VNDS. The columns in the table have the same meaning of those

[5] The results for this instance were obtained by a VNS [20] code without decomposition.

in Table 1. One column is added to the end of the table, Q_{lit}, which refers to the best results found in the literature[6] [7].

Table 2. Results for the Quality Challenge in the Clustering instances of the 10th DIMACS Implementation Challenge.

category	instance	Q_{avg}	Q_{best}	$\|\mathcal{C}\|_{best}$	Q_{lit}
Qu1	`karate`	0.419790	0.419790	4	0.4198 [1, 4, 25, 26, 35, 40, 43]
	`chesapeake`	0.265796	0.265796	3	`ukn`
	`dolphins`	0.528519	0.528519	5	0.529 [1, 43]
	`lesmis` [8]	0.566688	0.566688	6	`ukn`
	`polbooks`	0.527237	0.527237	5	0.527 [1, 25, 26]
	`adjnoun`	0.313367	0.313367	7	0.308 [25]
	`football`	0.604570	0.604570	10	0.605 [1, 25, 26]
	`jazz`	0.445144	0.445144	4	0.445 [1, 26]
	`celegansneural`	0.503782	0.503782	5	`ukn`
	`celegans_metabolic`	0.453240	0.453248	9	0.452 [26, 40]
Qu2	`e-mail`	0.582636	0.582799	12	0.582 [26]
	`polblogs`	0.427105	0.427105	278	0.426 [25]
	`netscience`	0.959900	0.959900	407	0.9555 [4]
	`power`	0.940776	0.940874	41	0.93937 [7]
	`hep-th`	0.857601	0.857692	1379	`ukn`
	`PGPgiantcompo`	0.885989	0.886043	111	0.8841 [26, 35]
	`cond-mat`	0.853247	0.853402	1264	`ukn`
	`astro-ph`	0.744559	0.744887	1080	`ukn`
	`as-22july06`	0.677403	0.677825	38	`ukn`
	`cond-mat-2003`	0.776422	0.776717	1686	`ukn`
	`cond-mat-2005`	0.744534	0.744882	1917	`ukn`
Qu3	`preferentialAttachment` [9]	0.315266	0.316778	9	`ukn`
	`smallworld`	0.792823	0.792910	252	`ukn`
	`G_n_pin_pout`	0.498791	0.499332	177	`ukn`
	`caidaRouterLevel`	0.869898	0.870414	499	`ukn`
	`coAuthorsCiteseer`	0.902691	0.902994	291	`ukn`
	`citationCiteseer`	0.820328	0.820750	207	0.8037 [11]
	`coAuthorsDBLP`	0.833034	0.833569	513	0.8269 [11]
	`cnr-2000`	0.912997	0.913029	431	`ukn`
	`coPapersCiteseer`	0.920414	0.920493	448	`ukn`
	`coPapersDBLP`	0.861000	0.861514	347	`ukn`
	`eu-2005`	0.941113	0.941359	400	`ukn`
	`in-2004`	0.980454	0.980513	1733	`ukn`
Qu4	`road_central`	0.997216	0.997294	1215	`ukn`

[6] Q_{lit} values are indicated with the same precision as reported in the referred papers.

[7] For some instances, we have not found available results in the literature. Their associated results are indicated by the label `ukn` in Table 2.

[8] The instance `lesmis` is often tested in the literature in its unweighted version. In the DIMACS challenge, the instance is weighted.

A few remarks are in order regarding the results shown in Table 2:

- Instances in category Qu1 were always solved up to optimality proved by algorithm in Section 3, except for instance `celegans_metabolic` in which optimality of the best know solution is not proved but was found by the heuristic 4 out of 5 times.
- Particularly for instance `jazz`, Duch and Arenas reported in [13] a solution with cost 0.4452 and 5 clusters. However, either the `jazz` instance used in the paper is different from the one of the DIMACS challenge or the authors commited a slight mistake, since the solution with cost 0.4551 and 4 clusters was proved optimal by the exact approach presented in Section 3.
- Out of the 34 instances tested, 14 had been previoulsy used in the literature. In 9 of them, the decomposition algorithm proposed here improved the best known solutions. In the remaining five, the algorithm matched the best known solutions (in fact, they were already optimal as proved by the exact algorithm of Section 3).

5 Conclusion

Several integer programming approaches and numerous heuristics have been applied to modularity maximization. They are due mostly to the physics and computer sciences research communities. We have applied the variable neighborhood search metaheuristic to that problem and it proves to be very effective. For problems with known optimum values, the heuristic always found an optimal solution at least once. For the other instances, the best know solution was improved in all cases except for large instances for which memory lacked for the implementation.

Acknowledgments: D.A. was partially supported by CAPES/Brazil grant number 2479-04-4. L.L and P.H. were partially supported by Digiteo grant *Senior Chair* 2009-14D "RMNCCO". G.C., P.H., and S.P. were partially supported by FQRNT (Fonds de recherche du Québec – Nature et technologies) team grant PR-131365. G.C. has been partially supported by NSERC(Natural Sciences and Engineering Research Council of Canada) grant 298138-2009. P.H. has been partially supported byNSERC grant 105574-07. S.P. has been partially supported by the NSERC grant 327435-06.

References

1. Agarwal, G. and Kempe, D., "Modularity-maximizing graph communities via mathematical programming", *The European Physical Journal B*, vol. 66, no. 3, pp. 409–418, (2008).
2. Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Liberti, L., Perron, S., "Column Generation Algorithms for Exact Modularity Maximization in Networks", *Physical Review E* vol. 82, no. 046112, (2010).

[9] The results for instance `preferentialAttachment` were obtained by a VNS [20] code without decomposition.

3. Alpert, C.J. and Yao, S.-Z., “Spectral Partitioning: The More Eigenvectors, The Better”, *Proc. 32nd ACM/IEEE Design Automation Conference* , pp. 195–200, (1995).
4. Barber, M.J. and Clark, J.W., “Detecting network communities by propagating labels under constraints”, *Physical Review E* vol. 80, no. 026129, (2009).
5. Blondel, V., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E., “Fast unfolding of communities in large networks”, *Journal of Statistical Mechanics*, p. P10008, (2008).
6. Boccaletti, S., Ivanchenko, M., Latora, V., Pluchino, A., and Rapisarda, A., “Detecting complex network modularity by dynamical clustering”, *Physical Review E* 75 045102(R), (2007).
7. Cafieri, S., Hansen, P., and Liberti, L. “A locally optimal heuristic for modularity maximization of networks”, *Physical Review E*, 83 056105(1-8), (2011).
8. Chen, D., Fu, R., and Shang, M., “A fast and efficient heuristic algorithm for detecting community structures in complex networks”, *Physica A*, vol. 388, no. 13, pp. 2741–2749, (2009).
9. Clauset, A., Newman, M., and Moore, C., “Finding community structure in very large networks,” *Physical Review E*, vol. 70, no. 066111, (2004).
10. Danon, L., Diaz-Guilera, A., and Arenas, A., “The effect of size heterogeneity on community identification in complex networks”, *Journal of Statistical Mechanics*, vol. P11010, (2006).
11. Delling, D., Görke, R., Schulz, C., and Wagner D. “ORCA reduction and contraction graph clustering”. In *5th Int. Conf. on Algorithmic Aspects in Information and Management (AAIM)*, pp. 152–165, (2009).
12. Djidjev, H., “A scalable multilevel algorithm for graph clustering and community structure detection”, *Lecture Notes in Computer Science*, vol. 4936, (2008).
13. Duch, J. and Arenas, A., “Community identification using extremal optimization”, *Physical Review E* 72 027104 (2), (2005).
14. Fan, Y., Li, M., Zhang, P., Wu, J., and Di, Z., “Accuracy and precision of methods for community identification in weighted networks”, *Physica A*, vol. 377, no. 1, pp. 363–372, (2007).
15. Fortunato, S.,“ Community detection in graphs”, *Physics Reports* , vol. 486, no. 3-5, pp. 75–174, (2010).
16. Girvan M. and Newman M. E. J., “Community structure in social and biological networks”, *Proc. Natl. Acad. Sci. U.S.A.* 99, 7821, (2002).
17. Goldschmidt, O. and Hochbaum, D.S., “A polynomial algorithm for the k-cut problem for fixed k”, *Mathematics of Operations Research* vol. 19, no.1, pp. 24–37, (1994).
18. Guimerà, R. and Amaral, A., “Functional cartography of complex metabolic networks”, *Nature*, vol. 433, pp. 895–900, (2005).
19. Hansen, P. and Jaumard, B., “Cluster analysis and mathematical programming”, *Mathematical Programming*, vol. 79, pp. 191–215, (1997).
20. Hansen, P., Mladenović, N., and Pérez, J.A.M., ‘Variable neighbourhood search: methods and applications”, *4OR*, vol. 6, pp. 319–360, (2008).
21. Jain A., Murty M., and Flynn P., “Data clustering: A review”, *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, (1999).
22. Kaufman L. and Rousseeuw P., *Finding Groups in Data. An Introduction to Cluster Analysis*, 2nd edition. Wiley Series in Probability and Statistics, (2005).
23. Kumpula, J., Saramaki, J., Kaski, K., and Kertesz, J., “Limited resolution and multiresolution methods in complex network community detection”, *Fluctuation and Noise Letters*, vol. 7, no. 3, pp. L209–L214, (2007).

24. Lehmann, S. and Hansen, L., "Deterministic modularity optimization", *European Physical Journal B*, vol. 60, pp. 83–88, (2007).
25. Li, W. and Schuurmans, D., "Modular Community Detection in Networks", *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp. 1366–1371, (2011).
26. Liu, X. and Murata, T., "Advanced modularity-specialized label propagation algorithm for detecting communities in networks", *Physica A* , vol.389, pp.1493–1500, (2010).
27. Massen, C. and Doye, J., "Identifying communities within energy landscapes", *Physical Review E*, vol. 71, no. 046101, (2005).
28. Medus, A., Acuna, G., and Dorso, C., "Detection of community structures in networks via global optimization", *Physica A*, vol. 358, pp. 593–604, (2005).
29. Mei, J., He, S., Shi, G., Wang, Z., and Li, W., "Revealing network communities through modularity maximization by a contraction-dilation method", *New Journal of Physics*, vol. 11, no. 043025, (2009).
30. Mirkin, B., *Clustering for Data Mining: A Data Recovery Approach*, Boca Raton, FL: Chapman and Hall/CRC, (2005).
31. Newman, M., "Fast algorithm for detecting community structure in networks", *Physical Review E*, vol. 69, no. 066133, (2004).
32. Newman, M., "Modularity and community structure in networks", *Proceedings of the National Academy of Sciences, USA*, pp. 8577–8582, (2006).
33. Newman, M. and Girvan, M., "Finding and evaluating community structure in networks", *Physical Review E*, vol. 69, no. 026133, (2004).
34. Niu, Y., Hu, B., Zhang, W., and Wang, M., "Detecting the community structure in complex networks based on quantum mechanics", *Physica A*, vol. 387, no. 24, pp. 6215–6224, (2008).
35. Noack, A., and Rotta, R., "Multi-level algorithms for modularity clustering", *Lecture Notes in Computer Science*, vol. 5526, pp. 257–268, (2009).
36. Richardson, T., Mucha, P., and Porter, M., "Spectral tripartitioning of networks", *Physical Review E*, vol. 80, p. 036111, (2009).
37. Ruan, J. and Zhang, W., "Identifying network communities with a high resolution", *Physical Review E*, vol. 77, no. 016104, (2008).
38. Schuetz, P. and Caflisch, A., "Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement", *Physical Review E*, vol. 77, no. 046112, (2008).
39. Shi, J. and Malik, J., "Normalized cuts and image segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, 888–905, (2000).
40. Sun, Y., Danila, B., Josic, K., and Bassler, K.E., "Improved community structure detection using a modified fine-tuning strategy", *Europhysics Letters*, vol. 86, no. 28004, (2009).
41. Tasgin, M., Herdagdelen, A., and Bingol, H., "Community detection in complex networks using genetic algorithms", *arXiv:0711.0491*, (2007).
42. Wakita, K. and Tsurumi, T. "Finding community structure in mega-scale social networks", Tech. Rep. 0702048v1, arXiv, (2007).
43. Xu, G., Tsoka, S., and Papageorgiou, L., "Finding community structures in complex networks using mixed integer optimization", *The European Physical Journal B*, vol. 60, pp. 231–239, (2007).