RAIRO-Oper. Res. 50 (2016) 269–283
DOI: 10.1051/ro/2015046

RAIRO Operations Research
www.rairo-ro.org

COMBINED NEIGHBORHOOD TABU SEARCH FOR COMMUNITY DETECTION IN COMPLEX NETWORKS

Olivier Gach[1] and Jin-Kao Hao[1,2,*]

Abstract. Community is one prominent feature of complex networks. Community detection is one important research topic in the area of complex networks analysis. In this paper, we introduce a new heuristic algorithm for community detection using the popular modularity measure. The proposed algorithm, called CNTS for combined neighborhood tabu search (CNTS), relies on a joint use of vertex move and merge operators to improve the quality of visited solutions. A dedicated tabu mechanism provides the algorithm with additional capacities to effectively explore the search space. Experiments using a collection of 21 well-known benchmark instances show that the proposed algorithm competes favorably with state-of-the-art algorithms.

Mathematics Subject Classification. 90-08, 90C27.

Received September 12, 2015. Accepted September 28, 2015.

1. Introduction

Complex networks are a graph-based model which is very useful to represent connections and interactions of the underlying entities in a real networked system such as social, biological, and technological networks [39]. A vertex of the complex network represents an object of the real system while an edge symbolizes an interaction between two objects. For example in a social network, each vertex corresponds to a particular member of the network while the edges incident to the vertex represent the relationships between this member and other members.

Complex networks may be huge and typically display non-trivial topological features and special patterns which characterize its connectivity and impact the dynamics of processes applied to the network [6]. Analysis and synthesis of complex networks help discover these specific features, understand the dynamics of the networks and represent a real challenge for research [1, 2, 14, 46].

A complex network may contain specific groups of highly interconnected vertices which are loosely associated with other groups. Such a group is commonly called community, cluster or still module [39] and all the communities of a network form a clustering. In terms of graph theory, a clustering can be defined as a partition of the vertices of the underlying graph into disjoint subsets, each subset representing a community. A community C is typically characterized by two basic factors: intra-cluster density and inter-cluster density. Intuitively, a

Keywords. Community detection, heuristics, tabu search, graph partitioning, clustering, combinatorial optimization.

[1] LERIA, University of Angers, 2 Bd Lavoisier, 49045 Angers cedex 01, France.

[2] Institut Universitaire de France, Paris, France.

* Corresponding author: hao@info.univ-angers.fr

community is a cohesive group of vertices that are connected more "densely" to each other than to the vertices in other communities. To quantify the quality of a given community and more generally a clustering, *modularity*, as defined in Section 2, is certainly the most popular measure [40]. Under this quality measure, the problem of community detection can be considered as an interesting combinatorial optimization problem.

Community detection with modularity is an important research topic and has a number of concrete applications [14]. In addition to its practical interest, community detection is also notable for its difficulty from a computational point of view. Indeed, the problem is known to be NP-hard [8] and constitutes thus a real challenge for optimization methods.

A number of heuristic algorithms have been proposed recently in the literature for community detection with the modularity measure. These algorithms follow three general solution approaches. First, greedy agglomeration algorithms like [11,38] iteratively merge two clusters that yield a clustering by following a greedy criterion, *e.g.*, permitting the largest increase or the smallest decrease in modularity. Greedy algorithms are generally fast, but the expected quality can hardly match that of other more sophisticated algorithms. Second, local search algorithms like [33, 44, 45] progressively improve the solution quality by transitioning from a clustering to another clustering (often of better quality) by applying a move operator. For instance, the popular vertex move operator transfers a vertex from its current community to another community [26]. The quality of such an algorithm depends strongly (among other things) on the move operator(s) employed. Third, hybrid algorithms like [5, 16, 17, 32, 41] combine several search strategies (*e.g.*, greedy and multi-level methods) in order to take advantage of the underlying methods. Among the existing community detection algorithms, the most efficient methods often use either the vertex move or merge operator, sometimes within the multi-level framework.

In this paper, we introduce a new heuristic algorithm for community detection using the modularity measure. The proposed algorithm, called CNTS for combined neighborhood tabu search (CNTS), relies on a *joint use* of vertex move and merge operators to improve the quality of the visited solutions. To complement these two intensification-oriented operators, the proposed algorithm integrates an additional diversification strategy. A tabu list prevents the search from revisiting previously visited solutions. We also use a new vertex move method to generate the initial solution required by the tabu search procedure.

To assess the performance of the proposed algorithm, we test our algorithm on a set of 21 popular networks in the literature and compare our results with three state-of-the-art methods. The experimental results demonstrate that the proposed algorithm is competitive.

The rest of the paper is organized as follows. In Section 2, we define the problem of community detection under the "modularity" quality measure and the two basic move operators. In Section 3, we explain in details the proposed algorithm. Section 4 is dedicated to computational assessments and comparisons with reference algorithms, followed by conclusions and perspectives in the last section.

2. Definition and basic move operators

2.1. Clustering and modularity

Definition 2.1. Given an undirected graph $G = (V, E)$ with vertex set V ($|V| = n$) and edge set E ($|E| = m$), a *clustering* of G is a partition $\{C_1, C_2, \ldots, C_K\}$ of V, where each set C_i forms a *community* (also called cluster or module) if the vertices of C_i are strongly connected to each other while they are loosely connected to the vertices of other sets.

There are different measures to quantify the quality of a graph clustering [14]. Modularity [40] is certainly the most widely used quality measure which can be described as the sum of the differences in density between the vertices of the graph and vertices in a random graph of same size.

Definition 2.2. The *modularity* of a clustering with K communities is defined by the following formula:

$$Q = \sum_{i=1}^{K} \left[\frac{l_i}{m} - \left(\frac{d_i}{2m} \right)^2 \right]. \tag{2.1}$$

In this formula, l_i is the number of internal edges of community C_i (*i.e.*, both endpoints of each edge are in C_i) and d_i is the sum of the degrees of the vertices of community C_i. This definition can be extended to weighted graphs.

It is easy to show that Q takes real values in the interval [-0.5,1]. A clustering with a small Q value close to the lower bound implies the absence of real communities. A large Q value close to 1 indicates a good clustering containing highly cohesive communities. In particular, the modularity of a trivial clustering with a single cluster has a value of 0.

Given the modularity measure Q, the community detection problem aims to find a particular clustering with the maximal modularity Q among all possible candidates of a given graph. This is thus a highly combinatorial optimization problem and known to be NP-hard [8]. Consequently, heuristic algorithms are indispensable to find approximating solutions within reasonable computing time.

2.2. Move operators for local search

Most of the recent communities detection algorithms using modularity are agglomerative and based on two move operators called vertex mover and merger.

2.2.1. Vertex move operator

Definition 2.3. The *vertex move operator* (or simply vertex-mover) displaces a vertex from its current community to a neighbor community. The migration of vertex v to community C is written as $\textsc{Vm}(v, C)$ or $\textsc{Vm}(v, C', C)$ if the original community C' of v needs to be indicated.

This move operator is the key operation of some highly popular community detection algorithms like [44] and graph partition algorithms like [26].

Relative to a move operator, we define the *move gain* as the variation in modularity when the move operator is applied to the current clustering to obtain a transformed clustering. Given the move operator $\textsc{Vm}(v, C', C)$, we define now its move gain. Let $\deg(v)$ be the degree of v in G, $d_C(v)$ the internal degree of vertex v within community C, *i.e.*, the number of edges between v and any vertex of C. Let d_C and $d_{C'}$ designate the sum of the internal degrees of communities C and C' respectively. Then Formula (2.2) gives the *move gain* of $\textsc{Vm}(v, C', C)$.

$$\varDelta\textsc{Vm}(v, C', C) = \frac{d_C(v) - d_{C'}(v)}{m} + \deg(v) \, . \left[\frac{d_{C'} - \deg(v) - d_C}{2m^2}\right] \cdot \tag{2.2}$$

This formula shows that the move gain for each vertex move can be calculated efficiently in an incremental way. Firstly because the terms m and $\text{dev}(v)$ are constant and the terms d_C and $d_{C'}$ can be stored in memory for the whole graph and updated incrementally after a move: $d_C \leftarrow d_C + \deg(v)$ and $d_{C'} \leftarrow d_{C'} - \deg(v)$. Secondly because the other terms, $d_C(v)$ and $d_{C'}(v)$, can be calculated during the exploration of all adjacent vertices of v, with an appropriate data structure.

2.2.2. Merger operator

Definition 2.4. The *merge operator* (or simply merger) [11,38] groups the vertices of two communities C and C' to form a new and larger community. The merge operator is denoted by $\textsc{Merge}(C, C')$.

This move operator is directly employed in algorithms like [44,45] and in multi-level algorithms like [5,41].

Like the vertex move operator, the move gain of $\textsc{Merge}(C, C')$ can also be calculated incrementally as follows.

$$\varDelta\textsc{Merge}(C, C') = \frac{l_{C,C'}}{m} - \frac{d_C d_{C'}}{2m^2}, \tag{2.3}$$

where $l_{C,C'}$ refers to the degree between communities C and C', *i.e.*, the number of edges between the two communities, d_C and $d_{C'}$ are respectively the internal degree of communities C and C'.

Here, only the term $l_{C,C'}$ is computationally expensive because we have to explore all the neighbors of the vertices belonging to C. We propose a solution to greatly reduce the computation time in Section 3.6.2.

3. Combined neighborhood tabu search

3.1. General procedure

The general procedure of our combined neighborhood tabu search (CNTS) algorithm is described in Algorithm 1. CNTS begins with an initial clustering $\mathcal{C}$ (lines 2−3, see Sect. 3.2) and then repeats a number of tabu search iterations (lines 3−15, see Sect. 3.3) until no significant improvement can be reached. For each improvement iteration, CNTS first identifies the possible vertex moves as well as merge moves involving only a subset V^k of vertices (lines 4−9, see Sect. 3.3.2). Among those eligible moves (*i.e.*, excluding the moves forbidden by the tabu list), CNTS selects the best move (*i.e.*, with the largest move gain), apply it to obtain a new clustering and updates the tabu list (lines 10−14, see Sect. 3.3.3). When the modularity improvement by tabu search becomes insignificant, CNTS triggers a post-optimization phase (Sect. 3.4) where all the vertices (instead of the vertices of the subset V^k) are considered to further improve the solution.

3.2. Initial solution

Like any local search algorithm, our CNTS algorithm starts with an initial clustering and then iteratively seeks better clusterings with its neighborhood-based search strategies. One simple way is to start the search with a trivial clustering where each vertex forms a community (this clustering has a very low modularity). However, we observed that it is more interesting for CNTS to start its search from an initial clustering of reasonable quality. For this purpose, we introduce the following initialization method.

The proposed method is based on the reverse vertex-mover (RVM) heuristic. RVM can be considered as a modified version of the Vm heuristic. Concretely, we begin with a clustering where each vertex forms a singleton community and then use the RVM heuristic to improve iteratively this clustering. Each iteration of RVM examines all the vertices of the graph and for each considered vertex v, we displace, among all the neighboring vertex of v, the vertex v' into the community of v that increases the most the modularity. At the end of an iteration, all the vertices are examined. We proceed with a new iteration if at least one vertex has migrated.

Unlike the classical Vm heuristic as it is used in the Louvain algorithm [5] where a vertex is moved from its community to a neighbor community at each iteration, a vertex here draws its neighbors into its own community. In CNTS, both RVM and Vm are respectively used during the initialization phase and tabu search phase as its basic heuristics. In Section 4.2.2, we study the effect of using RVM and Vm heuristics for initialization while in Section 4.4, we show how CNTS can improve its initial solutions.

Although the reverse vertex-mover alone cannot lead to excellent clusterings, it can generate clusterings of reasonable quality. In our experimental studies, we observed the initial clustering obtained with RVM has a modularity value varying between 70% and 80% of the best-known value ever reported in the literature.

3.3. Tabu search procedure

3.3.1. Tabu search principal

To improve the initial solution obtained from Section 3.2, we use the tabu search method [20] which is known to be a highly effective metaheuristic for tackling difficult combinatorial optimization problems.

Tabu search relies on a neighborhood to improve the quality of the visited solutions. The notion of neighborhood can be explained in terms of a *move* operator like the Vm operator defined in the last section. Typically applying a move mv to a solution s changes slightly s and leads to a neighboring solution s'. This transition from a solution to a neighbor solution is typically denoted by $s' = s \oplus mv$. Let $\Gamma(s)$ be the set of all possible moves which can be applied to s, then the neighborhood $N(s)$ of s can be defined by: $N(s) = \{s \oplus mv : mv \in \Gamma(s)\}$. At each iteration, tabu search selects always the best eligible neighboring solution to replace the current solution regardless of its quality. This transition rule allows the search to go beyond the local optima encountered. To prevent the search from revisiting a previously examined solution, tabu search maintains in a memory called *Tabu list* a portion of the last visited solutions (more precisely, the solution attributes or moves). A neighboring

Algorithm 1. Combined neighborhood search for community detection.

```
Require: Graph $G = (V, E)$.
Ensure: A clustering $\mathcal{C}$ of $G$ with a maximal modularity.
 1: // Generate an initial solution, see Sect. 3.2 //
 2: $\mathcal{C} \leftarrow$ InitialClustering($G$)
 3: repeat
 4:     // Build the possible moves, see Sect. 3.3.2 //
 5:     $N^k \leftarrow$ Subset($k, V$)                          ▷ $N^k \subset V$ contains $k$ vertices
 6:     $M \leftarrow$ LocalMoves($N^k, \mathcal{C}$)              ▷ $M$ contains the possible vertex moves
 7:     if no improvement of modularity during previous iteration (Sect. 3.5) then
 8:         $M \leftarrow M \cup$ MergeMoves($N^k, \mathcal{C}$)      ▷ Add merge moves
 9:     end if
10:     // Choose and apply a move, see Sect. 3.3.3 //
11:     RemoveTabuMoves($M, T$)                    ▷ Exclude tabu moves from M
12:     $m \leftarrow$ SelectBestMove($M$)                 ▷ Pick the best move
13:     UpdateTabuList($T, m$)                     ▷ Add chosen move $m$ to tabu list $T$
14:     $\mathcal{C} \leftarrow$ ApplyMove($\mathcal{C}, m$)              ▷ Generate a new clustering
15: until no modularity improvement after the process of all vertices of $V$ (Sect. 3.3.4)
16: // Improvement with all the vertices of $V$, see Sect. 3.4 //
17: $\mathcal{C} \leftarrow$ PostImprovement($\mathcal{C}, V$)                                  ▷
```

solution is then called eligible if it is not forbidden by the tabu list or if it is better than any previously encountered solution.

3.3.2. Move operators and combined neighborhood

To improve the modularity, we use jointly the vertex-mover VM and the merger operators. We employ them in a combined way to better exploit their complementary nature and additionally constraint their application to some dedicated subsets of vertices.

Definition 3.1. Let $N^k \subset V$ be a subset of k vertices of graph $G = (V, E)$, then the *combined neighborhood* is defined by the *union* [34] of the two following sets (M_1 and M_2) of neighboring solutions:

- $M_1 = \{s \oplus \text{VM}(v, C)\}$ where v is any vertex of N^k ($v \in N^k$) and C is any community containing at least one vertex adjacent to v ($\exists v' \in C, \{v, v'\} \in E$).
- $M_2 = \{s \oplus \text{MERGE}(C, C')\}$ where v is any vertex of N^k ($v \in N^k$), C' is the community containing v and C is any community with at least one edge linked to v ($\exists v' \in C, \{v, v'\} \in E$).

Notice that contrary to the vertex-mover operator, a bad merge operation is difficult to repair. For this reason, the MERGE operator is allowed only if it improves the modularity.

Our CNTS algorithm examines, at each of its iterations, all the neighboring solutions from M_1 and M_2 and selects the best one for the transition. After each iteration, the move that was just applied is recorded in the tabu list in order to prevent the following iterations from undoing this move. We will explain in Section 3.3.3 the role and management of the tabu list.

3.3.3. Tabu list and move selection strategy

For each move operator $\text{VM}(v, C', C)$ or $\text{MERGE}(C, C')$ (Sect. 2.2), we use the tabu list to forbid the recent moves during a fixed number tt of iterations (tt is called the tabu tenure and is determined experimentally).

More specifically, each time the $\text{VM}(v, C', C)$ operator is applied to transfer a vertex v from community C' to C, v is forbidden to join C' during the period fixed by tt. At each iteration of the search algorithm, any tabu move (*i.e.*, any move present in the tabu list) is excluded from the neighborhood. The tabu status of a move is nevertheless disabled if the move leads to a solution better than the best found so far (this is called aspiration in tabu search).

3.3.4. Stop criterion

At each iteration of the algorithm, k vertices are examined in the neighborhood set N^k. Thus, to explore all the n vertices of V, we have to process n/k iterations. The algorithm ends when n/k iterations (rounded to the equal or superior integer) are reached without a significant improvement of the modularity. This improvement significance is appreciated with respect of a threshold ϵ, which is a non crucial parameter of the algorithm.

3.4. Post-improvement

To complete its search, CNTS applies a post-improvement procedure which combines the vertex move heuristic [26, 44] and the greedy merge procedure of [11, 38]. Each iteration of this post-improvement procedure examines *all* the vertices of the graph and for each considered vertex v, the community of v is merged with the neighbor community that leads to the largest modularity increase. Otherwise, if no such merge is possible (*i.e.*, all merges lead to modularity decreases), we transfer vertex v from its current community C' to a neighboring community C such that this vertex transfer improves the most the modularity. Like with the vertex move heuristic, at the end of an iteration, all the vertices are examined and the post-improvement procedure proceeds with a new iteration if at least one vertex has migrated or two communities have been merged.

3.5. Additional strategies

Based on the main algorithm presented so far, we introduce three additional strategies which prove to be useful in some cases. The first enhancement is related to the application of the MERGE operator. After a MERGE move, this operator is forbidden until the modularity decreases. With this strategy, the search algorithm applies the merge operator one time and waits until the modularity begins to degrade (due to tabu search). The rationale behind this strategy is to use MERGE as a diversification mechanism when the search can no longer find any improved solution.

The second strategy concerns the order in which communities are merged, studied in particular in [41] using the notion of merge prioritizer.

Definition 3.2. A *merge prioritizer* assigns to each pair of community (C, C') a real number called merge priority, and then determines the order in which the algorithm merges community pairs. A prioritizer is a function MP : $\{\mathcal{P}(V), \mathcal{P}(V)\} \longmapsto \mathbb{R}$, with $\mathcal{P}(V)$ contains all the subsets of V.

The simplest function compares potential merges uniquely based on the modularity gain $\Delta\text{MERGE}(C, C')$. We will use a more informative function to associate a priority to a given $\text{MERGE}(C, C')$ move. Our tests show that the function $\Delta\text{MERGE}(C, C')/\min(d_C, d_{C'})$ proposed in [12] performs well in a number of cases. To enhance its performance, we propose an improved merge priority function that takes into account the current modularity to reduce the effect of the division by d_C:

$$\text{MP}(C, C') = \frac{\Delta\text{MERGE}(C, C')}{\min(d_C, d_{C'})^{1-Q}}. \tag{3.1}$$

This function proves to be the best merge prioritizer among those we experimented and is thus integrated in our CNTS algorithm.

The third improving strategy used by CNTS relies on the multi-level optimization framework [4, 24] to reduce the computation time of the algorithm for large graphs. The multi-level version of our CNTS algorithm, denoted by CNTS-ML, makes local refinement at each level by applying CNTS with a reduction factor rf, *i.e.*, until the number of communities is decreasing by rf percents. After the refinement of the graph of the current level, a coarsen graph is generated in which each vertex corresponds to a community in the original graph. Then we apply CNTS again to this coarsen graph until CNTS has no effect on a coarsen graph. During the uncoarsening phase, the clustering at each level is simply improved by the VM heuristic (see Sect. 3.2).

3.6. Implementation

3.6.1. Representation of graphs and clustering

A graph is represented by a data structure that contains the incidence matrix. The vertices are represented by numbers from 1 to n. Our implementation supports undirected graphs which can be weighted or unweighted. A clustering is simply represented by a vector $\mathcal{C}$ that associates each vertex indexed by $i \in \{1, \ldots, n\}$ to the community $\mathcal{C}[i]$ containing the vertex. This structure also records useful information necessary for the incremental calculation of move gains (l_i and d_i in the formulas (2.2) and (2.3)).

3.6.2. Graph of communities

Our CNTS algorithm uses a graph of communities $G^{\mathcal{C}} = (V^{\mathcal{C}}, E^{\mathcal{C}})$ associated with a clustering $\mathcal{C}$, where each vertex represents a community, and each edge represents the existence of at least one edge between the two related communities. The edge is weighted with the number of edges of G between the two communities (the sum of the weights if G is weighted). An edge weight in $G^{\mathcal{C}}$ is equal to $l_{C,C'}$ in formula (2.3). The clustering representation does not allow us to scan quickly the vertices of a given community so the complexity of $\Delta\text{Merge}(C, C')$ is $O(m)$, which makes the evaluation of a Merge expensive. The graph community stores $l_{C,C'}$ in the weights of the edges and makes $\Delta\text{Merge}(C, C')$ independent of n and m, as long as the value d_C is also stored.

This technique has the advantage of greatly accelerating the search. The second advantage is that the graph of communities can be used by multi-level approaches like [41]. The disadvantage is that it should be updated after each move. For graphs with a poor community structure, so with a large number of communities, this technique increases the computational time. Yet, for all real graphs that we tested, this technique helps reduce the overall computing time.

4. Computational results

4.1. Protocol for the experiments

This section is dedicated to the performance assessment of the proposed CNTS algorithm which is coded in Free Pascal[3]. For this purpose, we carry out extensive experiments on a set of 21 networks commonly used in the field of community detection (Tab. 1). Directed graphs are transformed into undirected graphs and loops are removed. Our algorithm also takes into account weighted graphs (*USAir97*, *Astro-ph* and *Condmat2003*). Notice that for most of these graphs, the ground truth clutering is unknown. For this reason, our experimental studies focus on modularity optimization.

To report computational results, we follow the common practice of the literature. As our main quality indicators, we use the modularity values that can be achieved and the computing time needed (based on a PC equipped with a Pentium Core i7 870 of 2.93 GHz and of 8 GB of RAM.) The algorithm requires several parameters that are fixed experimentally (see Sect. 4.3). For each run, the program stops when the modularity cannot be further improved.

The vertex move and reverse vertex move heuristics used in CNTS (see Sect. 3.2) are sensitive to the order of vertices of the given graph. This is also the case for the reference algorithms that are used for computational comparison (see below). To make a reliable assessment and fair comparisons, we generate for each tested graph 100 random orders of its vertices (*i.e.* 100 different graph instances) and run each algorithm (our algorithm as well as each reference algorithm) on these 100 instances. We report then the average and maximum modularity from the 100 results.

To assess the performance of the proposed CNTS algorithm, we compare CNTS with three state-of-the-art algorithms:

- SS+ML: a multi-level algorithm based on a single-step greedy coarsening and fast greedy refinement [41].
- MSG-VM: a multistep greedy algorithm with vertex mover [44].
- Louvain: a fast multi-level greedy algorithm [5].

[3]The source code of our CNTS algorithm is available at: `http://www.info.univ-angers.fr/pub/hao/cnts.html`

TABLE 1. 21 benchmark graphs for community detection commonly used in the literature.

Graph	Description	n	m	Source
Karate Club	Zachary karate club network	34	78	[48]
Dolphins	dolphin association network	62	159	[35]
Political Books	network of co-purchased political books	105	441	[29]
College Football	network of games between college football teams	115	613	[18]
Codeminer	Source code structure of a Java program	724	1015	[23]
C. elegans	metabolic network for the nematode C.Elegans	453	2025	[13]
USAir97	direct flight connections between US airports in 1997	332	2126	[3]
Jazz	jazz musician collaborations network	198	2742	[19]
E-mail	university e-mail network	1133	5451	[22]
Power	topology of the Western States Power Grid of the United States	4941	6594	[47]
Yeast	Protein-Protein interaction network in yeast	2284	6646	[9]
Epa	pages linking www.epa.gov in a search engine	4271	8909	[27]
Erdos	Erds collaboration network	6927	11 850	[21]
California	Pages matching the query "California" in a search engine	6175	15 969	[28]
Arxiv	network of scientific papers and their citations	9377	24 107	[25]
PGP	trust network of mutual signing of cryptography keys	10 680	24 316	[7]
Zemail	Email network	6640	54 173	[43]
Condmat2003	scientific coauthorship network in condensed-matter physics	27 519	116 181	[37]
Astro-ph	collaboration network of arXiv Astro Physics	16 046	121 251	[30]
Enron	email network from Enron	36 692	183 831	[31]
Brightkite	friendship network from a location-based social networking service	58 228	214 078	[10]

To make a relevant comparison, we compile and execute all programs on the same machine. For SS+ML and MSG-VM we use the published program written in C++. For Louvain, we use our implementation in Free Pascal, according to the published algorithm. Among the existing algorithms for community detection with modularity optimization, these three algorithms are certainly the most popular and representative, as a consequence, can be considered as highly relevant reference methods for a reliable comparison.

Our first experiment aims to identify the key components of the algorithm as well as appropriate parameter values while the second experiment compares our results with the three reference community detection algorithms.

4.2. Preliminary study

4.2.1. Merge prioritizer

The first important choice concerns the merge priority function (see Sect. 3.5). At each iteration of the algorithm, the examined set of vertices N^k determines a set of potential $\textsc{Merge}(C,C')$. The merge with maximal and positive priority is selected and executed. We decide to assess five classical functions, previously presented in the literature (see a summary in [41]):

- DeltaQ: $\Delta\textsc{Merge}(C,C')$.
- MinDeg: $\frac{\Delta\textsc{Merge}(C,C')}{\min(d_C,d_{C'})}$.
- MinSize: $\frac{\Delta\textsc{Merge}(C,C')}{\min(|C|,|C'|)}$.
- Sig: $\frac{\Delta\textsc{Merge}(C,C')}{\sqrt{d_C d_{C'}}}$ (significance).
- WT: $\min\left(\frac{|C|}{|C'|},\frac{|C'|}{|C|}\right)\Delta\textsc{Merge}(C,C')$ (Wakita and Tsurumi).

In preliminary tests, we observe that MinDeg and MinSize are better than DeltaQ mostly for graphs with high modularity, typically above 0.8. Thus, we propose a formulation, with an exponentiation of $1-Q$, such that with high modularity the prioritizer approximates DeltaQ and with low modularity the prioritizer tends to become MinDeg or MinSize. In addition, this modification favors merges of small communities at the early stage of the

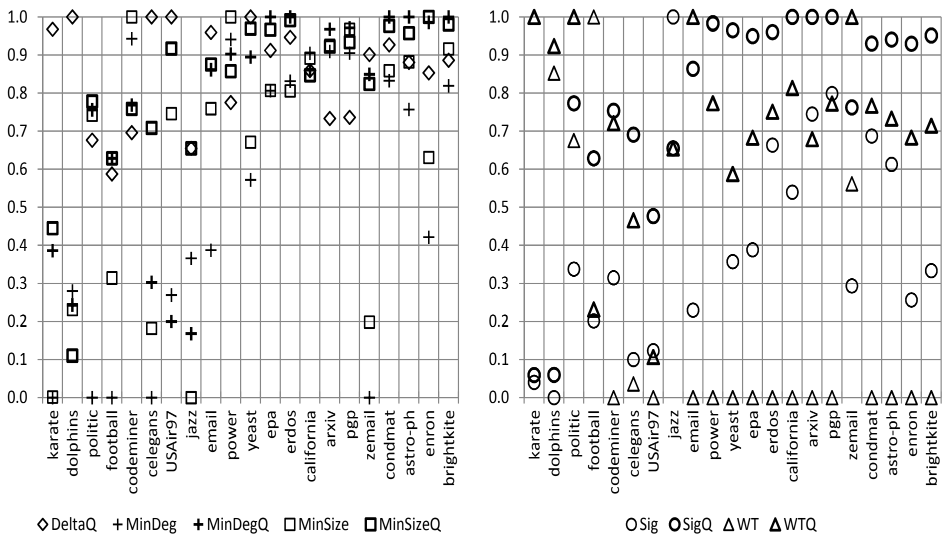

FIGURE 1. Assessment of the merge priority function in CNTS version. Graphics present, for each tested graph, the distribution on a normalized scale between 0 and 1, of the nine (one for each function) average modularity values over the 100 instances of graph.

search (when Q is far from its optimal value) and tends to favor merges of big communities in the end. We generalize this principle for all other functions than DeltaQ:

- MinDegQ: $\frac{\Delta\textsc{Merge}(C,C')}{\min(d_C,d_{C'})^{1-Q}}$.
- MinSizeQ: $\frac{\Delta\textsc{Merge}(C,C')}{\min(|C|,|C'|)^{1-Q}}$.
- SigQ: $\frac{\Delta\textsc{Merge}(C,C')}{(\sqrt{d_C d_{C'}})^{1-Q}}$.
- WTQ: $\min\left(\frac{|C|}{|C'|},\frac{|C'|}{|C|}\right)^{1-Q} \Delta\textsc{Merge}(C,C')$.

Figure 1 shows the results of the five merge priority functions from the literature as well as the four generalized functions using the 21 benchmark graphs. From the left part, one does not observe a clear predominant function among the five. The performance of DeltaQ is good on the set of graphs, but MinSizeQ clearly has a very good behavior from *football* and especially from *email* to the biggest graph. The right graphic shows a good performance of SigQ from *Power*, but a very weak results for the three smallest graphs. For almost all graphs, the performance of exponent $1-Q$ prioritizers is better than the original function. MinSizeQ and MinDegQ both have very good overall results. Since MinDegQ is globally the best, we choose the MinDegQ prioritizer for all the tests that follow.

4.2.2. Reverse vertex-mover and multi-level CNTS

The proposed CNTS algorithm could start with a clustering obtained by either a classical vertex-mover or the reverse vertex-mover procedure (see Sect. 3.2). The multi-level version of CNTS, *i.e.*, CNTS-ML (see Sect. 3.5), could also use vertex-mover or reverse vertex-mover to initially cluster the coarsened graph at each level, before executing the tabu search refinement procedure. In this section, we wish to assess four combinations of these techniques:

(1) CNTS(vm): CNTS presented in Section 3 using the vertex-mover heuristic to produce an initial clustering before the tabu search optimization.

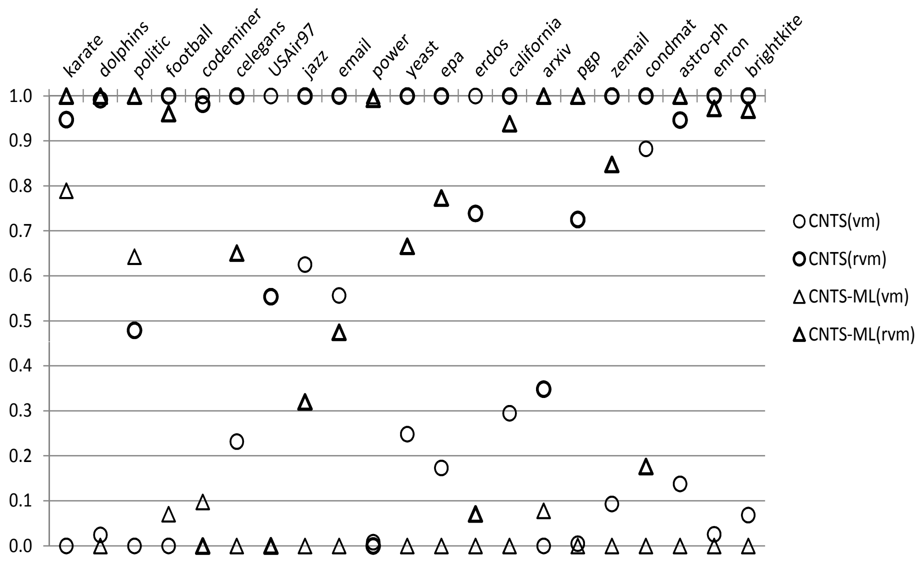

FIGURE 2. Comparison between four versions of CNTS. The two basic CNTS versions (CNTS(vm) and CNTS(rvm)) are executed with parameters $k = 200$ and $tt = 0$, the multi-level CNTS versions (CNTS-ML) are run with a reduction factor $rf = 0.1$. The graphic presents, for each graph, the distribution on a normalized scale between 0 and 1, of the four (one for each version) average modularity over the 100 instances of graph.

(2) CNTS(rvm): CNTS using the reverse vertex-mover heuristic to produce an initial clustering before the tabu search optimization.
(3) CNTS-ML(vm): multi-level CNTS presented in Section 3.5 using the vertex-mover heuristic to produce an initial clustering before the tabu search optimization at each level.
(4) CNTS-ML(rvm): multi-level CNTS using the reverse vertex-mover heuristic to produce an initial clustering before the tabu search optimization at each level.

The comparitive results, presented in Figure 2, show a clear dominance of CNTS over CNTS-ML, and the reverse vertex-mover (bold symbol) over the vertex-mover heuristic (normal symbol). For all graphs, CNTS(rvm) is the best method.

4.3. Parameter settings

4.3.1. Sample size k used by the combined neighborhood

The main parameter for the performance of the CNTS algorithm is the size of the set of vertices, explored by the VM and MERGE operators of the algorithm (see Sect. 3.3.2). Intuitively, a large k value corresponds to a larger combined neighborhood. A large neighborhood would leads to a better modularity optimization. However, exploring large neighborhoods is clearly more time-consuming. Experiments show a relatively low sample size compared to the number of vertices is sufficient to produce a very good modularity optimization in our algorithm because every merge is followed by a fine adjustment by vertex moves, as it is shown in Figure 3, where we represent the average distance of the modularity found relatively to the maximum modularity for each graph. The distance becomes very small from $k = 200$. For the set of graphs, the distance rate has no extreme values, ranging from 0% to a maximum of 0.08% for *condmat*. We choose thus $k = 200$.

4.3.2. Tabu tenure

In CNTS, the tabu tenure tt represents the number of tours during which a move is classified tabu and will be excluded to being considered for the neighborhood examination. A tour is a complete examination of the

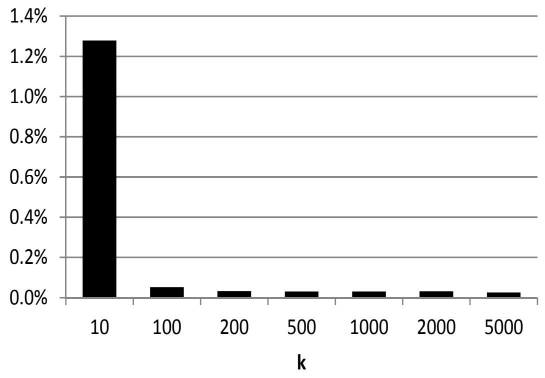

FIGURE 3. Assessment of the sample size parameter k. The CNTS algorithm is executed with the parameter $tt = 0$. The represented data is the average of relative distance between the maximum modularity found for all parameter values and the modularity found for the measured parameter value. For instance, with $k = 10$, Q is almost 1.3% lower than the maximum. We test seven values of k.

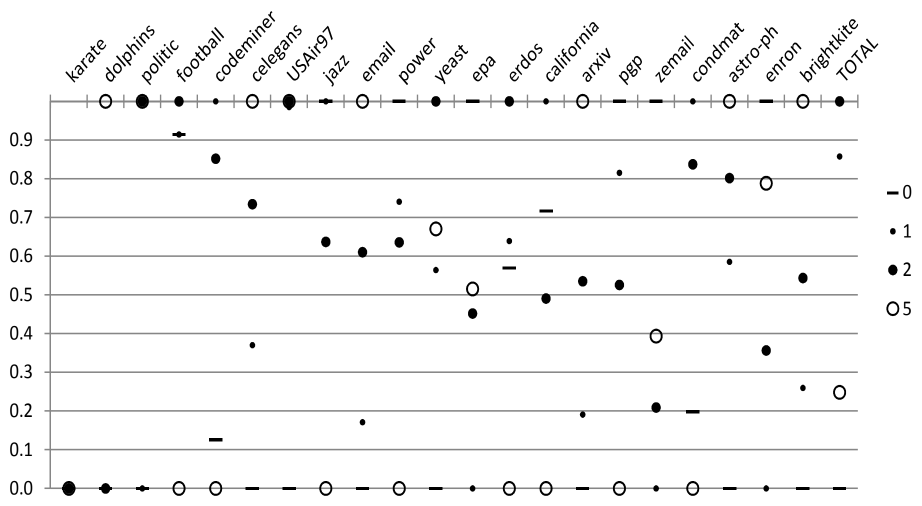

FIGURE 4. Assessment of the tabu tenure parameter tt. The CNTS algorithm is presented with other parameter $k = 200$. The represented data is the same as in Figure 2 with the rate of increase of modularity relative to the minimum. We test four values $tt = \{0, 1, 2, 5\}$. $tt = 2$ seems the best setting.

neighborhood. Since a vertex is examined once each tour, it is logical to express the duration of a prohibition move by a number of tours. The positive effect of tabu tenure is shown in Figure 4. However, this effect is relatively limited (maximum 0.2% of increase of Q) and depends on graphs. The worst results are for the extreme values $tt = 0$ and $tt = 5$ and the best one is observed when $tt = 2$.

4.4. Computational results

In this section, we show a comparison between our proposed approach and the three state-of-the-art algorithms mentioned above: SS+ML (multi-level algorithm), MSG-VM (multi step greedy algorithm) and Louvain (fast multi-level greedy algorithm). We run the CNTS algorithm and its multi-level version CNTS-ML described in Section 3 with the parameters given in Table 2. Since the heuristic used by CNTS for its initialization is

TABLE 2. Parameters used in CNTS and CNTS-ML algorithms.

CNTS	CNTS-ML	Param.	Description
×	×	k	Number of vertices used to build the neighborhood, Section 3.3.2
	×	rf	Reduction factor, *i.e.*, proportion of community reduction before passing to the next level
×	×	tt	Tabu tenure, Section 3.3.3 expressed as a number of iterations
×	×	ϵ	Stop condition for the vertex-mover heuristic, set to 10^{-5} in our tests

TABLE 3. Average modularity for the 21 benchmark graphs. MSG-VM is executed with a parameter level $l = \sqrt{W(V,V)/2}$ (recommended in [44]), SS+ML with a reduction factor of 0.1 (recommended in [41]). For our algorithms, CNTS ($k = 200, tt = 2$) and CNTS-ML ($k = 200, rf = 0.1, tt = 2$). The best performance is indicated in bold and the average size of communities in parenthesis only for the two best algorithms SS+ML and CNTS (CNTS-ML has practically the same values than CNTS).

Graph	MSG-VM	SS+ML		Louvain	CNTS		CNTS-ML
karate	0.3981	0.4198	(4)	0.4164	0.4194	(4)	0.4196
dolphins	0.5246	0.5262	(5)	0.5198	0.5234	(6)	0.5236
politic	0.5254	0.5256	(6)	0.5205	0.5270	(5)	0.5270
football	0.5848	0.5999	(10)	0.6037	0.6044	(10)	0.6045
codeminer	0.8691	0.8724	(32)	0.8665	0.8721	(33)	0.8709
celegans	0.4369	0.4457	(11)	0.4369	0.4454	(11)	0.4443
USAir97		0.2047	(5)	0.1958	0.2119	(4)	0.2042
jazz	0.4447	0.4447	(4)	0.4428	0.4450	(4)	0.4446
email	0.5717	0.5788	(11)	0.5685	0.5789	(12)	0.5772
power	0.9361	0.9381	(41)	0.9357	0.9365	(42)	0.9381
yeast	0.5889	0.6026	(46)	0.5925	0.6015	(50)	0.6009
epa	0.6570	0.6675	(34)	0.6495	0.6687	(38)	0.6669
erdos	0.6986	0.7157	(37)	0.6964	0.7155	(42)	0.7141
california	0.6613	0.6776	(112)	0.6568	0.6726	(118)	0.6726
arxiv	0.8052	0.8211	(64)	0.8143	0.8204	(72)	0.8206
pgp	0.8737	0.8843	(102)	0.8823	0.8843	(112)	0.8849
zemail	0.6731	0.6833	(17)	0.6736	0.6813	(18)	0.6812
condmat		0.8161	(83)	0.8102	0.8159	(90)	0.8151
astro-ph		0.7362	(417)	0.7313	0.7413	(427)	0.7410
enron	0.6130	0.6289	(1212)	0.6105	0.6281	(1267)	0.6278
brightkite	0.6793	0.6969	(641)	0.6854	0.6957	(727)	0.6955
TOTAL	11.5408	13.4853		13.3083	13.4882		13.4735
#best	0	13		0	6		4

MSG-VM is unable to process weighted graphs (*USAir97*, *condmat* and *astro-ph*).

a simplified version of Louvain, the comparison presented in this section implicitly shows how much CNTS improves its initial solutions.

Table 3 shows the average modularity of 100 results for each graph (recall that 100 random orders of vertices are generated for each graph). Clearly, the algorithm SS+ML has the best average modularity for the largest number (13) of graphs, but CNTS has the largest total of average modularity. MSG-VM and Louvain achieve the worst results for the set of 21 graphs. An important difference concerns the average number of communities found by the two best algorithms, SS+ML and our CNTS method. For all the large graphs, from *power*, CNTS founds significantly more communities than SS+ML. This reveals an attenuated trend, for the maximization of modularity, to cluster a graph with few large communities. This behavior is especially due to the reverse vertex-mover procedure.

Table 4 shows the maximum modularity reached by each competing algorithm. From Table 4, we observe that our algorithms CNTS and CNTS-ML compete very favorably with the reference algorithms that are based on

TABLE 4. Maximum modularity for the 21 graphs. The algorithms are run under the same conditions as indicated in Table 3.

Graph	MSG-VM	SS+ML	Louvain	CNTS	CNTS-ML
karate	0.3981	0.4198	0.4198	0.4198	0.4198
dolphins	0.5246	0.5286	0.5278	0.5286	0.5286
politic	0.5254	0.5273	0.5205	0.5270	0.5273
football	0.6032	0.6045	0.6046	0.6046	0.6046
codeminer	0.8705	0.8728	0.8702	0.8728	0.8728
celegans	0.4457	0.4504	0.4472	0.4513	0.4499
USAir97		0.2059	0.1958	0.2119	0.2141
jazz	0.4447	0.4451	0.4452	0.4452	0.4452
email	0.5746	0.5813	0.5758	0.5820	0.5815
power	0.9381	0.9392	0.9371	0.9380	0.9392
yeast	0.5948	0.6068	0.5962	0.6053	0.6055
epa	0.6639	0.6707	0.6561	0.6726	0.6714
erdos	0.7037	0.7173	0.7000	0.7173	0.7162
california	0.6696	0.6799	0.6666	0.6779	0.6771
arxiv	0.8108	0.8226	0.8167	0.8226	0.8227
pgp	0.8802	0.8849	0.8842	0.8855	0.8858
zemail	0.6776	0.6843	0.6794	0.6832	0.6829
condmat		0.8167	0.8102	0.8159	0.8160
astro-ph		0.7381	0.7345	0.7432	0.7429
enron	0.6213	0.6300	0.6241	0.6315	0.6311
brightkite	0.6840	0.6989	0.6909	0.6970	0.6971
TOTAL	11.6301	13.5240	13.4019	13.5322	13.5307
#best	0	11	3	11	10

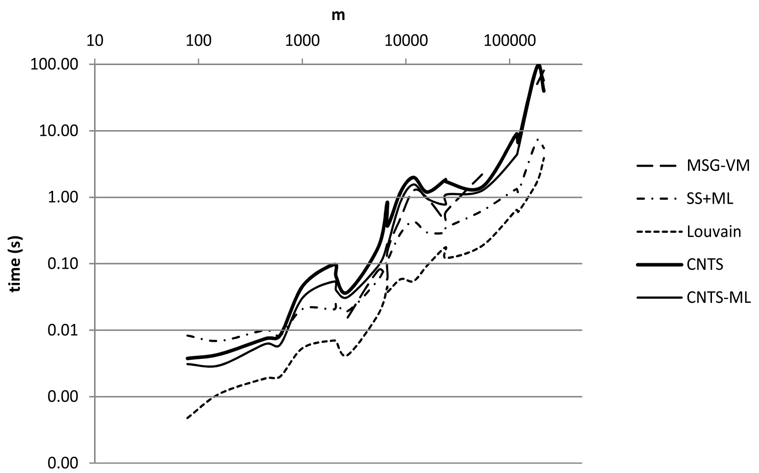

FIGURE 5. Comparison of execution times in seconds relative to the number of edges (m) of the tested graphs. The conditions of experimentation are the same as for the average modularity. Both axes are in log-scale.

several different approaches. Indeed, CNTS and CNTS-ML reaches the highest modularity value for 11 and 10 respectively, compared to MSG-VM, SS+ML, Louvain which reaches the best value for 0, 11 and 3 respectively. Moreover, like for the average modularity, CNTS has the best overall maximum score of 13.5322 while CNTS-ML has the second best overall maximum score.

Finally, Figure 5 shows a comparison of computing times of the competing algorithms. As we can observe, Louvain is the fastest algorithm for all the tested graphs. Overall, it seems that the time of CNTS and CNTS-ML grow slightly faster than SS+ML and Louvain, with almost 100 s for the larger graph against less than 10 with these algorithms.

5. Conclusions

Community detection using a global criterion (*i.e.* modularity maximization) is a hot research topic for complex network analysis. We have presented a new heuristic approach to solving this hard combinatorial optimization problem. The proposed algorithm is based on the tabu search metaheuristic and uses two complementary move operators. The vertex-move and merge operators are jointly applied to ensure an efficient and intensified exploitation of the search space. Additional strategies like merge conditions and multi-level optimization are also studied.

The proposed CNTS algorithm has been assessed on a set of 21 popular benchmark graphs and has shown competitive results with respect to three state of the art methods in terms of modularity maximization and community size.

This paper focuses on the very popular modularity measure which, however, is deemed to have a tendency to produce too large communities at the expense of relatively small communities (well-known resolution limit [15]). Our algorithm seems to reduce this effect because the number of communities found is larger than other algorithms, with similar modularity values. Other quality measures like generalized modularity Q_λ [42] or merit factor [36] have been proposed to mitigate this limit. It would be interesting to check how the approach proposed in this paper can be adapted to these measures.

Acknowledgements. We are grateful to the referees for the comments and questions which helped us to improve the paper. We thank the editors of this Special Issue for handling our paper. The work is partially supported by the Pays de la Loire Region (France) within the RaDaPop (2009−2013) and LigeRO (2009−2013) projects.

References

[1] R. Albert and A.-L. Barabási, Statistical mechanics of complex networks. *Rev. Mod. Phys.* **74** (2002) 47.

[2] A. Barrat, M. Barthlemy and A. Vespignani, Dynamical Processes on Complex Networks. Cambridge University Press (2008).

[3] V. Batagelj and A. Mrvar, Email network. http://vlado.fmf.uni-lj.si/pub/networks/data/default.htm.

[4] U. Benlic and J.K. Hao, A multilevel memetic approach for improving graph k-partitions. *IEEE Trans. Evol. Comput.* **15** (2011) 624-472.

[5] V.D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebvre, Fast unfolding of communities in large networks. *J. Stat. Mech.* **10** (2008) 8.

[6] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez and D. Hwang, Complex networks: Structure and dynamics. *Phys. Rep.* **424** (2006) 175–308.

[7] M. Boguñá, R. Pastor-Satorras, A. Díaz-Guilera and A. Arenas, Models of social networks based on social distance attachment. *Phys. Rev. E* **70** (2004) 056122.

[8] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski and D. Wagner, On modularity clustering, *Knowl. Data Eng.* **20** (2008) 172–188.

[9] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling and N. Zhang, Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Research* **31** (2003) 2443–2450.

[10] E. Cho, Seth A. Myers and J. Leskovec, Friendship and Mobility. ACM Press (2011) 1082.

[11] A. Clauset, M.E.J. Newman and C. Moore, Finding community structure in very large networks. *Phys. Rev. E* **70** (2004) 066111.

[12] L. Danon, A. Díaz-Guilera and A. Arenas, The effect of size heterogeneity on community identification in complex networks. *J. Stat. Mech.* **2006** (2006) P11010.

[13] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Phys. Rev. E* **72** (2005) 027104.

[14] S. Fortunato, Community detection in graphs. *Phys. Rep.* **486** (2010) 75–174.

[15] S. Fortunato and M. Barthélemy, Resolution limit in community detection. *Proc. Natl. Acad. Sci.* **104** (2007) 36–41.

[16] O. Gach and J.K. Hao, A memetic algorithm for community detection in complex networks. In PPSN 2012. Edited by C. Coello *et al.* Vol. 7492 of *Lect. Notes Comput. Sci.* (2012) 327–336.

[17] O. Gach and J.K. Hao, Improving the Louvain algorithm for community detection with modularity maximization. In Conf. of AE 2013. Edited by P. Legrand *et al.* In vol. 8752 of *Lect. Notes Comput. Sci.* (2014) 145–156.

[18] M. Girvan and M.E.J. Newman, Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* **99** (2002) 7821–7826.

[19] P. Gleiser and L. Danon, Community structure in social and biological networks. *Adv. Complex Syst.* **6** (2003) 565–573,.

[20] F. Glover and M. Laguna, Tabu Search. In Modern Heuristic Techniques for Combinatorial Problems. Edited by C. Reeves. Blackwell Scientific Publishing, Oxford, England (1993).

[21] J. Grossman, *The erdös number project.* Available at http://www.oakland.edu/enp/ (2007).

[22] R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt and A. Arenas, Self-similar community structure in a network of human interactions. *Phys. Rev. E* **68** (2003) 065103.

[23] J. Heymann and S. Palmier, Source code structure of a java program. Available at http://wiki.gephi.org/index.php/Datasets.

[24] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* **20** (1998) 359–392.

[25] *KDD. Cornell kdd cup.* Available at http://www.cs.cornell.edu/projects/kddcup/ (2003).

[26] B.W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs. *Bell Syst. Technical J.* **49** (1970) 291–307.

[27] J. Kleinberg, *A network of pages linking www.epa.gov in a search engine.* Available at http://www.cs.cornell.edu/courses/cs685/2002fa/.

[28] J. Kleinberg, *A network of pages matching the query "california" in a search engine.* Available at http://www.cs.cornell.edu/courses/cs685/2002fa/.

[29] V. Krebs, *A network of books about recent us politics sold by the online bookseller amazon.com.* Available at http://www.orgnet.com (2008).

[30] J. Leskovec, J. Kleinberg and C. Faloutsos, Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data* **1** (2006) 2.

[31] J. Leskovec, K.J. Lang, A. Dasgupta and M. Mahoney, Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.* **6** (2008) 66.

[32] X. Liu and T. Murata, Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A* (2009).

[33] Z. Lü and W. Huang, Iterated tabu search for identifying community structure in complex networks. *Phys. Rev. E* **80** (2009) 026130.

[34] Z. Lü, J.K. Hao and F. Glover, Neighborhood analysis: a case study on curriculum-based course timetabling. *J. Heuristics* **17** (2011) 97118.

[35] D. Lusseau, K. Schneider, O.J. Boisseau, P. Haase, E. Slooten and S.M. Dawson, The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behav. Ecol. Sociobiol.* **54** (2003) 396–405.

[36] A.D. Medus and C.O. Dorso, Alternative approach to community detection in networks. *Phys. Rev. E* **79** (2009) 066111.

[37] M.E.J. Newman, The structure of scientific collaboration networks. *Proc. of Natl. Acad. Sci. USA* **98** (2001) 404–409.

[38] M.E.J. Newman, Fast algorithm for detecting community structure in networks. *Phys. Rev. E* **69** (2004) 066133.

[39] M.E.J. Newman, Networks: An Introduction. Oxford University Press (2010).

[40] M.E.J. Newman and M. Girvan, Finding and evaluating community structure in networks. *Phys. Rev. E* **69** (2004) 026113.

[41] A. Noack and R. Rotta, Multi-level algorithms for modularity clustering. Preprint arXiv:0812.4073 (2008).

[42] J. Reichardt and S. Bornholdt, Statistical mechanics of community detection. *Phys. Rev. E* **74** (2006) 1–14.

[43] R. Rotta, *Email network.* http://studiy.tu-cottbus.de/~rrotta/

[44] P. Schuetz and A. Caflisch, Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Phys. Rev. E* **77** (2008) 046112.

[45] P. Schuetz and A. Caflisch, Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement **77** (2008) 046112.

[46] S.H. Strogatz, Exploring complex networks. *Nature* **410** (2001) 268–276.

[47] D.J. Watts and S.H. Strogatz, Collective dynamics of small-world networks. *Nature* **393** (1998) 440–2.

[48] W.W. Zachary, An information flow model for conflict and fission in small groups. *J. Anthropological Res.* **33** (1977) 452–473.