Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement

Philipp Schuetz and Amedeo Caflisch
Department of Biochemistry, University of Zurich, Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
(Received 7 December 2007; revised manuscript received 22 February 2008; published 17 April 2008)

Identifying strongly connected substructures in large networks provides insight into their coarse-grained organization. Several approaches based on the optimization of a quality function, e.g., the modularity, have been proposed. We present here a multistep extension of the greedy algorithm (MSG) that allows the merging of more than one pair of communities at each iteration step. The essential idea is to prevent the premature condensation into few large communities. Upon convergence of the MSG a simple refinement procedure called "vertex mover" (VM) is used for reassigning vertices to neighboring communities to improve the final modularity value. With an appropriate choice of the step width, the combined MSG-VM algorithm is able to find solutions of higher modularity than those reported previously. The multistep extension does not alter the scaling of computational cost of the greedy algorithm.

DOI: 10.1103/PhysRevE.77.046112 PACS number(s): 89.75.Fb, 05.10.−a, 89.75.Hc

I. INTRODUCTION

The networks under study in natural and social sciences often show a natural divisibility into smaller modules (or communities) originating from an inherent, coarse-grained structure. In general, these modules are characterized by an abundance of edges connecting the vertices within individual communities in comparison to the number of edges linking the modules.

To detect these partitions several algorithm- or score-based approaches have been developed and applied. Very popular became the approach introduced by Girvan and Newman [1] based on the quality function called "modularity" for partition assessment. This scoring function compares the actual fraction of intracommunity edges with its expectation in the random case given an identical degree distribution. The partition with the highest value of the scoring function is then considered to be the optimal splitting. The modularity Q is defined (for undirected networks) as

$$Q = \sum_{i=1}^{N_C} \left[\frac{I(i)}{L} - \left(\frac{d_i}{2L} \right)^2 \right]$$

with $I(i)$ the weights of all edges linking pairs of vertices in community i, d_i the sum over all degrees of vertices in module i, L the total weight of all edges, and N_C the number of communities.

Intrinsically, the modularity based approach does not prescribe the usage of a particular optimization procedure. In practice, a strategy for optimization has to be chosen. The modularity optimization is a NP-hard problem [2]. Therefore, only an exhaustive search reveals the optimal solution for a generic network. This type of search is extremely demanding and only in a few cases feasible. Thus, many heuristic approaches such as extremal optimization [3], simulated annealing [4], and the greedy algorithm [5] have been developed, refined, and successfully applied. Among the published approaches the greedy algorithm is one of the fastest techniques [6]. On the other hand, many examples show that the greedy algorithm is not capable of finding the solutions with the highest modularity value. Furthermore, recent studies have provided evidence that modularity [7] and Potts model based approaches [8] are endowed with an intrinsic resolution limit (small modules are not detected and amalgamated into bigger ones). Thus, each community has to be refined by subduing it as a separate network to the community detection algorithm. Therefore, a fast and accurate optimization technique is necessary.

In this article, we enhance the greedy algorithm by a multistep feature in combination with a local refinement procedure. The enhanced algorithm finds partitions with higher modularity values than previously reported. This paper is organized as follows. In Sec. II we introduce both procedures and describe the motivation for their construction. In addition, we discuss performance oriented implementations and estimate their running times. Benchmarking results for a set of real-world networks and a comparison with other published results are presented in Sec. III. The conclusions are in Sec. IV. In this paper, all networks are considered as undirected. The extension to directed networks is straightforward.

II. THE ALGORITHM

A. Multistep Greedy algorithm (MSG)

Each vertex is a community
Calculate the modularity change matrix ΔQ
Determine the community degrees d_i
while pair (i,j) with $\Delta Q_{ij} > 0$ exists **do**
  for all elements $(i,j,\Delta Q_{ij})$ in ΔQ matrix, parsed with respect to decreasing ΔQ and increasing (i,j)
  do
    if
      $\left\{ \begin{array}{l} \Delta Q_{ij} > 0 \text{ in best } l \text{ values in } \Delta Q \text{ matrix} \\ i \text{ and } j \text{ unchanged in iteration} \end{array} \right\}$
    then
      MergeCommunities(i,j)
    end if
  end for
end while

Algorithm 1: Flowchart of the MSG algorithm. The modularity change is calculated according to Eq. (1). Details of the algorithm are given in algorithm 2.

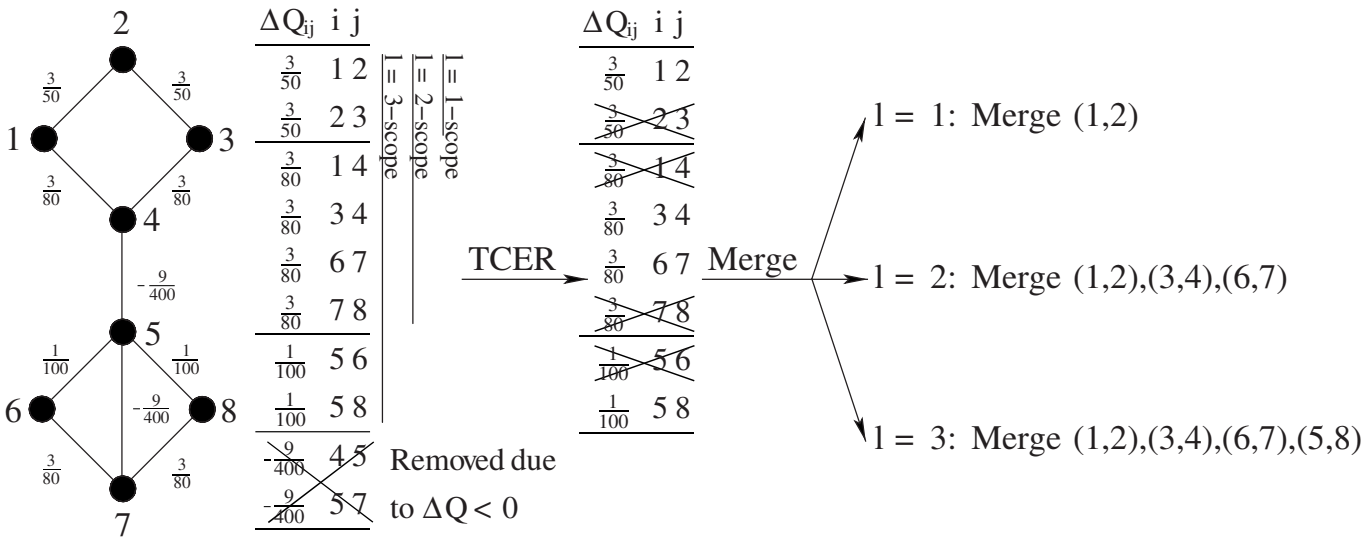

FIG. 1. Effect of different values of level parameter during first MSG iteration on example network.

The classical greedy algorithm (first application in Ref. [5]) joins iteratively the pair of communities that improves modularity most in each step. The essential idea of the "multistep greedy" (MSG) algorithm is to promote the simultaneous merging of several pairs of communities at each iteration. The pseudocode of the MSG algorithm is presented in algorithms 1 and 2, and an illustrative example is given in Fig. 1. The MSG algorithm starts with each vertex separated in its own community. At each iteration the modularity change ΔQ_{ij} upon merge of each pair of connected communities (i,j) is calculated (while nonconnected pairs are ignored because their merging yields a negative modularity change). The triplets $(i,j,\Delta Q_{ij})$ are parsed in the order of decreasing ΔQ value and increasing community index. Those community pairs (i,j) are joined which fulfill the following two criteria:

(1) The modularity change ΔQ_{ij} is within the l most favorable values (levels) and positive.

(2) Touched-community-exclusion-rule (TCER): Neither module i nor j is present in another pair inducing a higher modularity change.

Convergence is reached when all pairwise merges of communities decrease modularity (by induction one can prove that all merges in further iterations would decrease modularity). A level encompasses all triplets $(i,j,\Delta Q_{ij})$ with equal ΔQ_{ij} value and the level parameter l is kept constant. By construction the level parameter is always smaller than the number of edges in the network.

The multiple levels promote the concurrent formation of multiple centers. Simultaneously growing community centers hinder the condensation into few large communities (few formed communities scrape all vertices as the establishment of a new community is too expensive in modularity) as observed in the classical greedy algorithm. The TCER is a second mean against excessive aggregation into few large modules. This rule permits the addition of only one community to an existing community per algorithm iteration. Furthermore, the TCER guarantees that the modularity change upon all performed merges is just the sum over the corresponding ΔQ elements which improves efficiency.

B. Implementation details of MSG

The key observation for an efficient implementation of the MSG is the following: Upon merge of communities i and j only those ΔQ elements concerning either of the two modules have to be recalculated. When the modules i and j are joined into a new one called I, the updated modularity changes $\Delta Q_{Ik}^{\text{new}}$ (module k is connected either to community i or j) reads (see Sec. II in Ref. [9] for details)

$$\Delta Q_{Ik}^{\text{new}} = \begin{cases} \Delta Q_{ik} + \Delta Q_{jk}, & i,j, \text{ and } k \text{ pairwise connected}, \\ \Delta Q_{ik} - \dfrac{d_j d_k}{2L^2}, & i \text{ and } k \text{ connected}, j \text{ and } k \text{ not}, \\ \Delta Q_{jk} - \dfrac{d_i d_k}{2L^2}, & j \text{ and } k \text{ connected}, i \text{ and } k \text{ not}, \end{cases} \tag{1}$$

with d_x the sum over all degrees of vertices in community $x=i,j$ and L the total edge weight.

Further efficiency improvements are gained from an appropriate choice of data structures. A set (implementation taken from the C++-STL library) is a sorted binary search tree. In a set individual elements can be found or inserted in $O(\log(n))$ time (n the number of elements) and the extremal entries are found in constant time. The modularity changes are stored in the ΔQ matrix implemented as vector of row structures. The ith row consists of a set with elements $(j,\Delta Q_{ij})$ (j a module linked to the community i) ordered according to the community index j. This data structure obsoletes a separate storage of the topology information. The extraction of the best l modularity changes is handled via the *level set*. For each pair of connected communities i and j the element $(\min\{i,j\},\max\{i,j\},\Delta Q_{ij})$ is added to the *level set*. The *level-set* elements are sorted with respect to decreasing ΔQ and increasing index values. The degree information is stored in a vector henceforth named d. In each iteration a

Boolean vector called "touched" stores whether a community has already been modified in the same round. To save the time to determine the highest index of a present communities, the number of vertices (initial length) is chosen as length of the *touched* vector.

The implementation details of the MSG algorithm are listed in algorithm 2. The calculation of the community degrees involves one parse of the edge information. In the second parse of the edge information the ΔQ matrix and the *level set* is filled. The initial modularity change ΔQ_{ij} upon join of modules (at this stage the vertices) i and j is calculated as (see Sec. II in Ref. [9] for details)

$$\Delta Q_{ij} = \frac{I}{L} - \frac{d_i d_j}{2L^2}$$

with I the weight of the edges connecting the vertices i and j, d_x the degree of vertex $x=i,j$, and L the total edge weight. The modularity value of the initial partition is (N the number of vertices)

$$Q_0 = -\sum_{i=1}^{N} \frac{d_i^2}{4L^2}.$$

Each vertex is a community
Calculate community degrees d and the ΔQ matrix
Determine the initial modularity $Q \leftarrow Q_0 = -\Sigma_{i=1}^{n} \frac{d_i^2}{4L^2}$
level set $\leftarrow$ set of ΔQ elements $(i,j,\Delta Q_{ij})$, sorted with respect to decreasing ΔQ and increasing (i,j)
while first element of *level set* has $\Delta Q > 0$ **do**
  touched $\leftarrow (0,\ldots,0)$ Boolean,
  N-dimensional vector (N= No. vertices)
  {$touched_i = 1$, if module i is modified in *while*-loop}
  $MP \leftarrow$ subset of *level-set* elements $(i,j,\Delta Q_{ij})$ with $\Delta Q_{ij} > 0$ and ΔQ_{ij} among highest l values
  for all elements $(i,j,\Delta Q_{ij})$ of MP **do**
    if (**not** $touched_i$) **and** (**not** $touched_j$) **then**
      while parse $\Delta Q_{i.}$ and $\Delta Q_{j.}$ concurrently **do**
        ΔQ_{ik}

$$\leftarrow \begin{cases} \Delta Q_{ik} + \Delta Q_{jk}, & i,k \text{ and } j,k \text{ are linked,} \\ \Delta Q_{ik} - \frac{d_j d_k}{2L^2}, & i \text{ and } k \text{ are linked,} \\ \Delta Q_{jk} - \frac{d_i d_k}{2L^2}, & j \text{ and } k \text{ are linked,} \end{cases}$$

        $\Delta Q_{ki} \leftarrow \Delta Q_{ik}$
        Update the *level set*
        Update the modularity $Q \leftarrow Q + \Delta Q_{ik}$
      end while
      Empty $\Delta Q_{j.}$
      Flag $touched_i, touched_j \leftarrow 1$
      Update degrees: $d_i \leftarrow d_i + d_j, d_j \leftarrow 0$
    end if
  end for
end while

Algorithm 2: Performance-oriented implementation of MSG algorithm. The touched vector contains the information for the touched-community-exclusion-rule (TCER).

The algorithm iteration starts by initializing the *touched* vector. Subsequently, the *level set* is parsed and all elements with positive ΔQ value, whose modularity change is among the best l (external level parameter) different values, are stored in a set named MP conserving the order of the *level set*. In this order the module pairs are merged unless one of them was part of a amalgamation in the same algorithm iteration. In the merge process, the changed ΔQ matrix elements are calculated as described at the beginning of this paragraph. To determine which case applies in Eq. (1) the fact that each row of the ΔQ matrix is ordered with respect to the community index can be used. More precisely, parse for the merge of modules i and j the corresponding rows concurrently. For each row define an momentarily considered element p. If the community index of p_i is equal to the one of p_j, the first case applies and advance both p's to the next element in the corresponding row. If the index k of p_i is lower than the one of p_j calculate the $\Delta Q_{Ik}^{\text{new}}$ element (I the name of the merged community) according to the second case and advance (if possible) only p_i. If the module index of p_i is larger than the one of p_j, proceed analogously. If one p reaches the end of the row, merge the remaining elements of the other row according to the respective rule. This procedure will be called "asynchronous parsing" in Sec. II C. It is customary to update each ΔQ element after calculation. To complete the merge process it remains to update the community degrees and to flag the modified communities in the touched vector.

C. Running time estimation of MSG

As we adopted the modularity change calculation of Clauset *et al.* (Sec. II in Ref. [9]) we can adopt their method of running time estimation as well. First, we observe that the update of one element in the ΔQ matrix and the *level set* costs in the worst case $O(\log(N))$ (insertion in set, each community has at most N neighbors with N the number of vertices) and $O(\log(M)) = O(\log(N))$ running time (the number of distinct edges M is bounded by the square of the number of vertices N^2), respectively.

Merging communities i and j involves an update of the ΔQ matrix and the *level set* for each element of the corresponding rows of the ΔQ matrix. The calculation of each changed value can be achieved in constant time as during the asynchronous parsing it is known whether the other community is linked as well and all other information (community degrees) is stored in a vector. Thus, the total running time contribution of one merging event is $O((d_i + d_j)\log(N))$ with d_k the number of edge starts/ends on vertices of community $k=i,j$. In the worst case all communities are changed in one algorithm round. As the sum over all d_i values is twice the number of distinct edges, the contribution of the merging processes in one algorithm round is at most $O(M \log(N))$. The other steps of one algorithm round are less consumptive: The extraction of pairs belonging to the best l levels can be performed in constant time. The same is true for the update of the degree information. If D is defined as the depth of the dendrogram of communities, at most D algorithm rounds have to be performed. Thus, the running time expectation for

the iterative part is $O(DM \log(N))$ which is identical to the complexity of the classical greedy algorithm [9].

The initialization involves the read-in processes of the edge information (M constant time operations), the degree calculation (part of read-in process), the calculation of the initial modularity (constant time operation on N elements) and finally the generation of the ΔQ matrix and the *level set* at costs $O(M \log(N))$ (M insertions in a set with at most N or M elements, respectively). In the worst case the expected contribution of the initialization to the running time is $O(M \log(N))$.

In the precedent paragraphs we have shown that the MSG greedy algorithm has the total complexity $O(DM \log(N))$. Among the published strategies for modularity optimization the classical greedy algorithm [9] is the fastest [6]. As the MSG shares the worst case expectation for the running time with the classical greedy algorithm, we conclude that the MSG is one of the fastest procedures for modularity optimization.

D. Vertex mover (VM)

To further improve modularity by "adjusting" misplaced vertices, a refinement step called "vertex mover" (VM) is applied upon convergence of the MSG algorithm. In principle, it could also be applied to other modularity optimization procedures. In the VM, the list of vertices is parsed in the order of increasing degree and vertex index (to resolve the degeneracy of multiple vertices with equal degree) and every vertex is reassigned to the neighboring community with maximal modularity improvement. This parsing-and-reassignment procedure is repeated until no modularity improvement is observed.

The VM procedure is similar to the Kernighan-Lin algorithm [10] (applied to modularity optimization in Ref. [11]). In contrast to the Kernighan-Lin algorithm the VM procedure has a perfectly local focus. In other words, instead of repetitively searching for the optimal vertex to reassign, the VM procedure parses the vertices in the aforementioned order and identifies the optimal community for the considered vertex. Furthermore, each reassignment of the VM approach improves modularity. Therefore, the selection of the optimal intermediate partition as in the Kernighan-Lin algorithm is not necessary.

E. VM implementation

The modularity change ΔQ upon reassignment of vertex v from community i to j can be written as

$$\Delta Q = \frac{\text{links}(v \leftrightarrow j) - \text{links}(v \leftrightarrow i)}{L} - \frac{k_v(d_j - d_{i\setminus v})}{2L^2} \qquad (2)$$

with k_v the degree of vertex v, d_j the sum over the degrees of all vertices in community j, $d_{i\setminus v}=d_i-k_v$ the corresponding degree for community i without vertex v, and L the total weight of all edges.

The most time consuming part of the VM is the calculation of the modularity changes upon reassignment of the vertices. Consequently, Eq. (2) reduces this bottleneck to the calculation of weight of the edges connecting the vertex to the neighboring communities. The connectivity information of vertex v is stored in a sparse vector [i.e., a vector of elements (u, w_{vu}) with u a vertex linked to v and w_{vu} the total weight of all edges connecting vertices u and v]. These rows are stored in a vector and form the topology matrix. To determine the total edge weight connecting vertex v with community j the vth row is parsed and for each entry the weight is added to the subtotal edge weight of the corresponding community. To keep access times short a N-dimensional vector (N the number of vertices) is chosen to store the intermediate links$(v \leftrightarrow j)$ results. The optimal reassignment partner for vertex v is the community with smallest index yielding the maximal modularity improvement.

F. Estimation of VM running time

Calculating the modularity changes upon reassignment of one vertex to any neighboring community involves one parse of its edge list supplemented with direct memory access to determine the community affiliation and some constant time operations for the actual modularity calculation. Therefore, the running time contribution of one vertex is proportional to its degree. One algorithm round requires $O(L)=O(\Sigma_i d_i)$ running time. The estimation of the number of needed iterations is not possible as it depends on the quality of the MSG result. In all examples tested by us the running time of the VM was always at least one order of magnitude smaller and less than one minute even for the biggest networks under study.

III. RESULTS

A. Test set of networks

For benchmarking algorithms that optimize modularity the networks commonly used are the collaboration network (coauthorships in cond-mat articles) [12], the graph of metabolic reactions in *caenorhabitis elegans* [13], the email network [14], the network of mutual trust (PGP-key signing) [15,16], the conference graph of college football teams [17], the network of jazz groups with common musicians [18] and the Zachary karate club example [19]. In addition, we include less frequently used examples such as the graph of the metabolic reactions in *Escherichia coli* [20], two different data set describing the protein-protein interactions in S. cerevisiae (budding yeast) [21,22] with labels "PPI" and "yeast." To cover linguistic applications we benchmark the word association network [23] and the graph of the coappearing words in publication titles (co)authored by Martin Karplus [24] who has the third highest h factor [25] among chemists [26]. Further aspects of social webs were incorporated by considering the graph of costarring actors in the IMDB database [27]. Noticeable, the actor network—being the network with the largest number of edges—serves as a proof of concept for such big networks being treatable as well. From computer science we include the internet routing network [28] and the graph of World Wide Web pages [29]. With this selection of networks most currently known application fields of networks are covered. To study the effect of

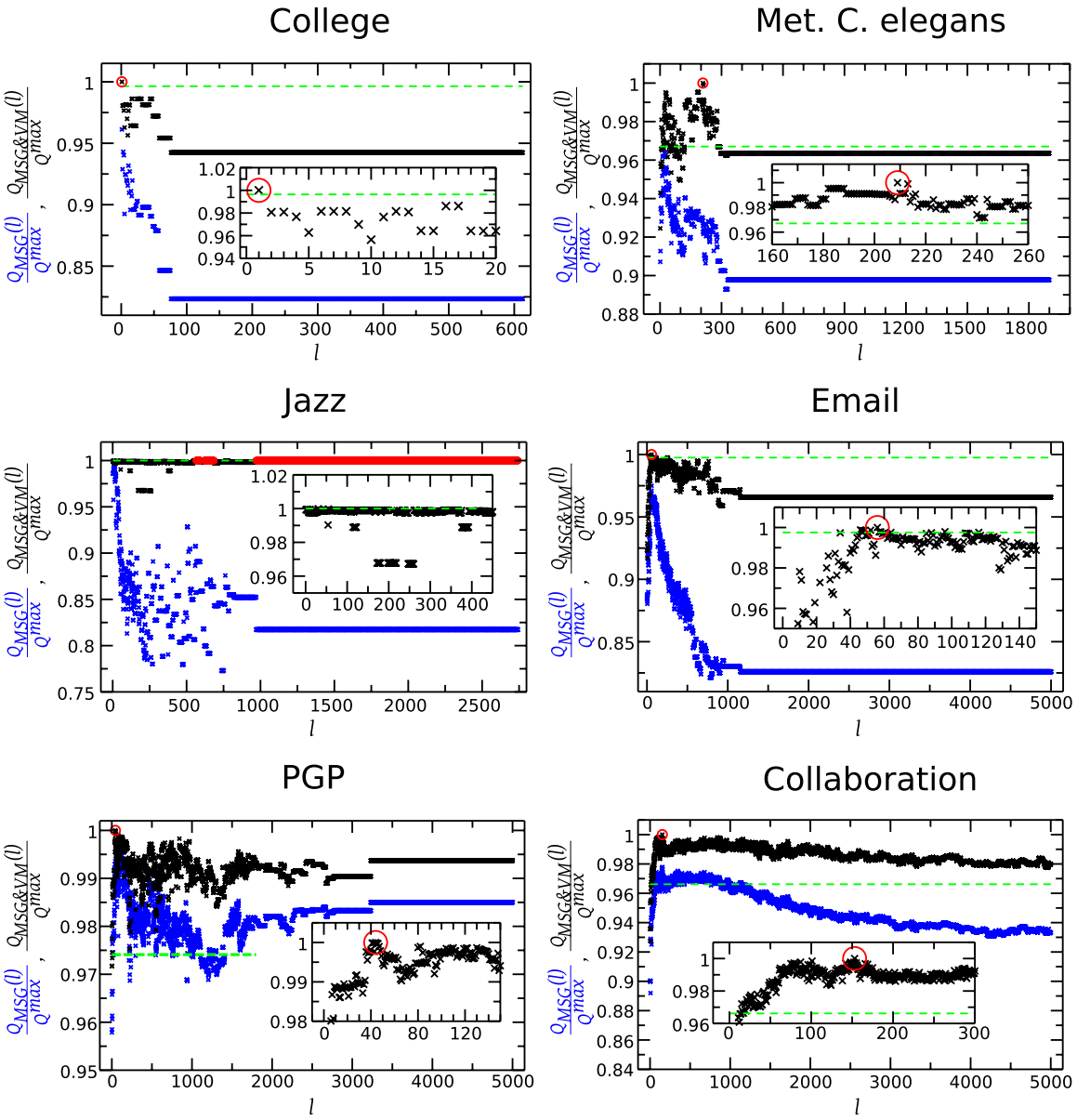

FIG. 2. (Color online) Dependence of MSG modularity value $Q_{\mathrm{MSG}}(l)$ (blue), MSG-VM modularity value $Q_{\mathrm{MSG\text{-}VM}}(l)$ (black) on the level parameter l relative to maximal MSG-VM modularity value $Q^{\max}$. The previously published highest modularity $Q_{\mathrm{pub}}/Q^{\max}$ (dashed green line) is also shown as basis of comparison. The red circles indicate the value of l that yields maximal modularity. A significant number of l values yield higher modularity than the previously published maximal modularity for all but the smallest two networks, i.e., Zachary (not shown) and College. In the latter, only $l=1$ yields a higher modularity than Q_{pub}.

disconnected graphs and weighted networks, we consider in both cases the full network as well as the largest connected component (suffix "CP") and the unweighted variant, respectively. Unless stated otherwise the networks are treated unweighted.

B. Dependence on l and vertex labeling

It is important to investigate the robustness upon the choice of l and to determine the highest modularity values achievable with the MSG-VM algorithm. There is a minor dependence on the value of l (Fig. 2) which changes the MSG-VM modularity by less than 2% for large networks. Moreover, the maximal modularity is obtained with $l<300$ for 14 of the 19 networks (Table I). An empirical formula for the optimal choice of the level parameter will be presented elsewhere.

Noteworthily, for a labeled graph and a chosen level parameter the algorithm is deterministic. To assess the contribution of the labeling, the benchmarking procedure is performed also on hundred copies of the smallest ten networks with permuted vertex labels. This permutation leaves the topology invariant, but modifies the order in which the community pairs are considered. In comparison to the maximal modularity value found for the unscrambled variants a maximal improvement of 0.94% is observed.

TABLE I. Results on real-world examples. Among all tested level parameters (all positive integers smaller than 5000 or the number of edges if smaller) the value l_{opt} yields the highest value of Q for the considered network. N_C is the number of communities found. In most cases, a larger number of communities (larger N_C) is identified by the classical greedy than the MSG-VM extension because the former partitions the network in few large communities and many small communities with less than ten vertices (mostly 2–20 times more small modules identified by greedy than MSG-VM). The MSG-VM approach prevents the condensation into few large modules: The three largest modules contain between 1.5 and 4 times less vertices in the MSG-VM partition than in the greedy partition (not shown). The running time (on a recent laptop) is reported for a single run of the algorithm. The entry "na" indicates that the running time is shorter than 1 s and therefore not displayed. The suffix "CP" points out that only the largest connected component (the "central part") was considered. The acronym "PPI" stands for "protein-protein interaction."

Network				MSG-VM				Greedy		
Name	Ref.	Vertices	Edges	l_{opt}	Q	Time [s]	N_C	Q	Time [s]	N_C
Zachary Karate Club	[19]	34	78	3	0.398	na	4	0.381	na	3
Metabolic *E. coli*	[20]	443	586	6, 8	0.816	na	19	0.811	na	20
College Football	[17]	115	613	1	0.603	na	8	0.556	na	6
Metabolic *C. elegans*	[13]	453	1899	209	0.450	na	8	0.412	na	13
Jazz	[18]	198	2742	566	0.445	na	4	0.439	na	4
Email	[14]	1133	5451	56	0.575	na	10	0.503	na	12
Yeast (PPI, CP)	[21]	2552	7031	35	0.706	na	33	0.675	na	51
M. Karplus weighted	[24]	1166	13423	91	0.316	na	11	0.264	na	18
PPI-CP *S. cerevisiae*	[22]	4626	14801	170	0.545	na	24	0.500	na	38
PPI *S. cerevisiae*	[22]	4713	14846	170	0.546	na	65	0.501	na	81
M. Karplus weighted	[24]	1166	18991	173	0.320	na	13	0.296	na	11
Internet	[28]	11174	23409	278	0.625	8	35	0.584	8	49
PGP-key signing	[15,16]	10680	24340	44	0.878	2	140	0.849	3	195
Word Association (CP)	[23]	7204	31783	71	0.541	4	16	0.452	7	52
Word Association	[23]	7207	31784	97	0.540	3	17	0.465	7	38
Collaboration	[12]	27519	116181	153	0.748	14	82	0.661	103	381
WWW	[29]	325729	1117563	3034	0.939	562	674	0.927	7640	2183
Actor	[27]	82583	3666738	2429	0.543	1722	238	0.470	6288	406
Actor weighted	[27]	82583	4475520	389	0.536	5099	322	0.480	3541	361

C. Performance and running time

The modularity values obtained with the MSG-VM approach are listed in Table II. For five of the seven networks considered here the MSG-VM algorithm finds solutions with modularity higher than previously published. Only for the Zachary Karate network the MSG-VM procedure yields a smaller modularity value. For the jazz network a solution with the identical Q value is obtained. For the networks without published modularity values we compare the optimal values obtained by the MSG-VM algorithm with the classical greedy algorithm for modularity optimization as introduced by Newman [5] in Table I. We observe that the MSG-VM algorithm outperforms the original greedy algorithm significantly.

The running time estimations in Secs. II C and II F are based on a worst case scenario. To investigate the running time behavior on real-world examples, we compare the running times of the classical greedy variant and the MSG-VM algorithm in Table I. These data show that given the appropriate level parameter choice the MSG-VM algorithm is in almost all cases faster than the classical greedy algorithm and, at the same time, reaches a higher value of modularity.

IV. CONCLUSIONS

To prevent premature condensation into few large communities the greedy algorithm for modularity optimization has been extended by a procedure for simultaneous merging of more than one pair of communities at each step. Further-

TABLE II. Comparison of maximal value of modularity obtained by the MSG-VM algorithm $Q_{max}^{MSG\text{-}VM}$ with previously published results Q_{pub}. The highest published value was extracted from the referenced paper ("Source") where it has been calculated by the "Method" whose reference is listed in the last column.

Network	$Q_{max}^{MSG\text{-}VM}$	Q_{pub}	Source	Method
Zachary Karate Club	0.398	0.419	[11]	[11]
College Football	0.603	0.601	[17]	[17]
Metabolic *C. elegans*	0.450	0.435	[11]	[11]
Jazz	0.445	0.445	[11]	[3]
Email	0.575	0.574	[11]	[3]
PGP-key signing	0.878	0.855	[11]	[11]
Collaboration	0.748	0.723	[11]	[11]

more, this "multistep" greedy variant has been combined with a simple vertex-by-vertex *a posteriori* refinement. On seven networks with previously published modularity values the MSG-VM algorithm combination outperforms all other frequently used, generic techniques except for the smallest of the seven examples. In addition, a single run of the MSG-VM algorithm requires similar computer time as the greedy algorithm. In most cases less than 10 independent (i.e., embarrassingly parallel) runs of MSG-VM are required to obtain a modularity within 1% of the highest value because an empirical formula has been derived for the appropriate choice of the optimal step width [30]. Therefore, the MSG-VM algorithm is an efficient tool to find network partitions with high modularity [31].

ACKNOWLEDGMENTS

The authors thank Stefanie Muff and Francesco Rao for helpful discussions. Christian Bolliger, Thorsten Steenbock, and Dr. Alexander Godknecht are acknowledged for maintaining the Matterhorn cluster where most of the parameter studies were performed. We are thankful to Drs. Arenas, Barabási, Gleiser, and Newman for providing the network data. This work was supported by a Swiss National Science Foundation grant to A.C.

[1] M. E. J. Newman and M. Girvan, Phys. Rev. E **69**, 026113 (2004).
[2] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner, e-print arXiv:physics/0608255.
[3] J. Duch and A. Arenas, Phys. Rev. E **72**, 027104 (2005).
[4] R. Guimerà and L. A. N. Amaral, Nature (London) **433**, 895 (2005).
[5] M. E. J. Newman, Phys. Rev. E **69**, 066133 (2004).
[6] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas, J. Stat. Mech. **2005**, P09008 (2005).
[7] S. Fortunato and M. Barthélemy, Proc. Natl. Acad. Sci. U.S.A. **104**, 36 (2007).
[8] J. M. Kumpula, J. Saramäki, K. Kaski, and J. Kertész, Eur. Phys. J. B **56**, 41 (2007).
[9] A. Clauset, M. E. J. Newman, and C. Moore, Phys. Rev. E **70**, 066111 (2004).
[10] B. Kernighan and S. Lin, Bell Syst. Tech. J. **49**, 291 (1972).
[11] M. E. J. Newman, Proc. Natl. Acad. Sci. U.S.A. **103**, 8577 (2006).
[12] M. E. J. Newman, Proc. Natl. Acad. Sci. U.S.A. **98**, 404 (2001).
[13] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A. L. Barabási, Nature (London) **407**, 651 (2000).
[14] R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas, Phys. Rev. E **68**, 065103(R) (2003).
[15] X. Guardiola, R. Guimerà, A. Arenas, A. Díaz-Guilera, D. Streib, and L. A. N. Amaral, e-print arXiv:cond-mat/0206240.
[16] M. Boguñá, R. Pastor-Satorras, A. Díaz-Guilera, and A. Arenas, Phys. Rev. E **70**, 056122 (2004).
[17] M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. U.S.A. **99**, 7821 (2002).
[18] P. Gleiser and L. Danon, Adv. Complex Syst. **6**, 565 (2003).
[19] W. W. Zachary, J. Anthropol. Res. **33**, 452 (1974).
[20] H. Ma and A.-P. Zeng, Bioinformatics **19**, 270 (2003).
[21] N. J. Krogan *et al.*, Nature (London) **440**, 637 (2006).
[22] V. Colizza, A. Flammini, A. Maritan, and A. Vespignani, Physica A **352**, 1 (2005).
[23] D. L. Nelson, C. L. McEvoy, and T. A. Schreiber, Behav. Res. Methods Instrum. Comput. **36**, 402 (2004).
[24] P. Schuetz and A. Caflisch, the network of words in the titles of Martin Karplus' publications (unpublished).
[25] J. E. Hirsch, Proc. Natl. Acad. Sci. U.S.A. **102**, 16569 (2005).
[26] P. Ball, Nature (London) **448**, 737 (2007).
[27] A.-L. Barabási and R. Albert, Science **286**, 509 (1999).
[28] Internet Network. Undirected, unweighted network of the Internet at the Autonomous System level from data collected by the Oregon Route Views Project (http://www.routeviews.org/) in May 2001, where vertices represent Internet service providers and edges connections among them. The file reports the list of connected pairs of nodes.
[29] R. Albert, H. Jeong, and A.-L. Barabási, Nature (London) **401**, 130 (1999).
[30] Ph. Schuetz and A. Caflisch (in preparation).
[31] The code is available at http://www.biochem-caflisch.uzh.ch/communitydetection/