Modularity optimization by conformational space annealing

Juyong Lee,[1,*] Steven P. Gross,[2,†] and Jooyoung Lee[1,‡]
[1]*School of Computational Sciences, Korea Institute of Advanced Study, Seoul, Korea*
[2]*Department of Developmental and Cell Biology, University of California, Irvine, California, USA*
(Received 21 February 2012; published 17 May 2012)

We propose a modularity optimization method, Mod-CSA, based on stochastic global optimization algorithm, conformational space annealing (CSA). Our method outperforms simulated annealing in terms of both efficiency and accuracy, finding higher modularity partitions with less computational resources required. The high modularity values found by our method are higher than, or equal to, the largest values previously reported. In addition, the method can be combined with other heuristic methods, and implemented in parallel fashion, allowing it to be applicable to large graphs with more than 10 000 nodes.

DOI: 10.1103/PhysRevE.85.056702 PACS number(s): 05.10.−a, 89.75.Hc, 02.10.Ox, 02.70.−c

I. INTRODUCTION

Network science has emerged as an important framework in which to study complex systems [1,2]. One of the most important properties of networks is the existence of modules or communities; communities are subgraphs of densely interconnected nodes, and nodes in a community are considered to share characteristics [3,4]. Proper community detection allows one to determine potentially hidden relationships between nodes and also allows one to reduce a large complex network into smaller and comprehensible ones. For this reason, good community detection within networks has been a subject of great interest. There exist various definitions of community [4–7]. The most widely used approach to detect such subgroups of nodes with nonrandom connections involves the use of modularity to quantify the quality of a given partition of a network [4,8,9]. Using modularity, the community detection problem is thus recast as a global optimization problem. However, finding the *maximum* modularity solution is an NP-hard problem [10], and enumeration of all possible partitions is intractable in general. Therefore, an efficient optimization algorithm is required to obtain high modularity solutions.

Most of the modularity optimization studies have focused on developing fast heuristic methods generating reasonable quality community structures. Currently, simulated annealing (SA) is considered to be the best algorithm [4,11] and has been adopted in many theoretical and practical studies where communities with high modularity is required [12–14].

In this paper we propose a new modularity maximization method based on a conformational space annealing (CSA) algorithm [15–19]. We show that CSA outperforms SA both in generating better community structures and in computational efficiency. CSA consistently finds community structures with higher modularity using fewer computational resources. Moreover, for networks containing approximately up to 1000 nodes, CSA repeatedly finds converged solutions. Considering the stochastic nature of the algorithm, this suggests that the converged solution is likely to be the optimal solution of the network.

*juyong@kias.re.kr
†sgross@uci.edu
‡jlee@kias.re.kr

II. METHODS

Let us consider a network with N nodes and M edges. Modularity measures the fraction of intracommunity edges minus its expected value from the null model, a randomly rewired network with the same degree assignments. Modularity is defined as

$$Q = \sum_{i=1}^{N_c} \left[\frac{l_i}{M} - \left(\frac{D_i}{2M} \right)^2 \right], \tag{1}$$

where N_c is the number of assigned communities, l_i is the number of edges within the community i, and D_i is the sum of degrees of nodes in the community i.

To benchmark the performance of CSA against that of SA, we implemented SA following existing studies [12,20]. Initially, using $E = -Q$, a simulation starts at a high temperature T, to sample *broad* range of the solution space as well as to avoid trapping in local minima. As the simulation proceeds, T is slowly decreased to more completely explore basins of high modularity. At a given T, a set of stochastic movements, including N^2 single-node moves and N collective moves consisting of random merges and splits of communities, are carried out. To split a community, a "nested" SA method is used [12,20], which isolates a target community from the entire network and splits it into two communities. Each "nested" SA starts with two randomly separated groups for short annealing, and the annealed solution serves as a collective move. For each trial movement, if Q increases, the movement is accepted, otherwise it is accepted with probability $P = \exp(\frac{Q_f - Q_i}{T})$. After a set of movements are tried, T is decreased to αT, where $\alpha = 0.995$.

Our method, CSA, is a global optimization method which combines essential ingredients of three methods: Monte Carlo with minimization (MCM) [21], genetic algorithm (GA) [22], and SA [23]. As in MCM, we consider only the phase or conformational space of local minima; i.e., all solutions are minimized by a local minimizer. As in GA, we consider many solutions (called *bank* in CSA) collectively, and we perturb a subset of bank solutions by crossover between solutions and mutation. Finally, as in SA, we introduce a parameter D_{cut}, which plays the role of the temperature in SA. In CSA each solution is assumed to represent a hypersphere of radius D in the solution space. Diversity of sampling is directly controlled

by introducing a distance measure between two solutions and comparing it with D_{cut}, to deter two solutions from coming too close to each other. Similar to the reduction of T in SA, the value of D_{cut} is slowly reduced in CSA; hence the name conformational space annealing.

To apply CSA to optimize modularity, three ingredients are required: (a) we need a local modularity maximizer for a given network partition, (b) we need a distance measure between two Q-maximized network partitions, and (c) we need ways to combine two parent partitions to generate a daughter partition which will be Q-maximized subsequently.

Here the community structure is represented by assigning an index to each node, where nodes with an identical index belong to the same community. For local maximization of Q, we use a quench procedure which accepts a move if and only if it improves Q, equivalent to SA at $T = 0$.

The distance between two community structures is measured by the variation of information V [24]. For two given partitions of X and Y, V is defined as

$$V(X,Y) = H(X,Y) - I(X;Y),$$

where H is the entropy function and I is the mutual information function of the probability $p(x,y) = n_{x,y}/n$, where n is the number of total nodes, x/y refers to a community from X/Y, and $n_{x,y}$ is the number of nodes shared by x and y. With H and I defined by

$$\begin{aligned} H(X,Y) &= -\sum_{x,y} p(x,y)\log p(x,y) \\ &= -\sum_{x,y} \frac{n_{x,y}}{n}\log\left(\frac{n_{x,y}}{n}\right), \\ I(X;Y) &= \sum_{x,y} p(x,y)\log\left[\frac{p(x,y)}{p(x)p(y)}\right] \\ &= \sum_{x,y} \frac{n_{x,y}}{n}\log\left(\frac{n_{x,y}n}{n_x n_y}\right), \end{aligned}$$

V can be reduced to

$$V(X,Y) = -\frac{1}{n}\sum_{x,y} n_{x,y}\log\left(\frac{n_{x,y}^2}{n_x n_y}\right), \tag{2}$$

where $p(x) = n_x/n$ and n_x is the number of nodes in community x. If X is identical to Y, it is easy to show that $V(X,Y) = 0$. We have also tried other measures such as the Rand index [25] and normalized mutual information (NMI) [26], and they gave no significant difference in results.

In CSA we first generate 50 random partitions, which are subsequently maximized by quench procedures. We call these solutions the *first bank*, which is kept unchanged during the optimization. We make a copy of the first bank and call it the *bank*. The partitions in the bank are updated by better solutions found during the course of optimization. The initial value of D_{cut} is set as $D_{\text{avg}}/2$, where D_{avg} is the average distance between partitions in the first bank. A number of partitions (30 in this study) in the bank are selected as *seed* partitions. With each seed, 20 trial partitions are generated by crossover between the seed and a randomly chosen partition from either the bank or the first bank. An additional five are generated by random mutation of the seed.

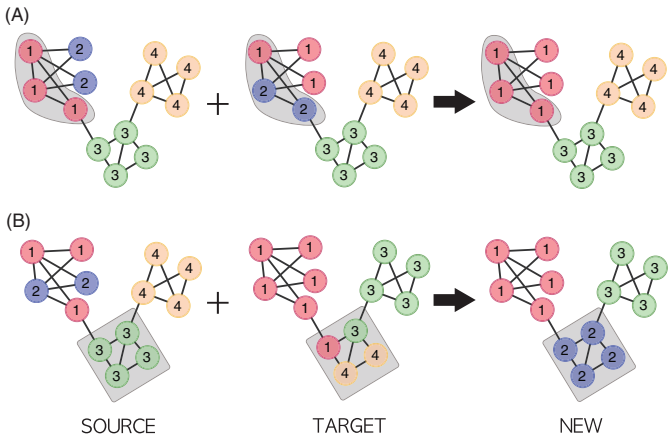

FIG. 1. (Color online) Two crossover operators, (A) convergent copy and (B) divisive copy, are shown. (A) The community indexed by 1 from the source is copied into the target, and the new indices are set to 1 or 3 with the probability of 2/3 and 1/3. (B) The community indexed by 3 from the source is copied into the target, and the new index 2 is assigned.

For a crossover, we use two operations, a convergent copy and a divisive copy, shown in Fig. 1. In both cases one community represented by an index is randomly selected from a source solution, and it is copied into a target solution after assigning a new index. For the convergent copy, the new index is chosen from one of the neighboring indices of the copied nodes from the target in a random fashion. For the divisive copy, a new index not present in the target is chosen. The rationale of using these operators is that the community index itself has no particular meaning, while a well-defined community structure from one solution can serve as an advantageous feature that should be preserved to generate a better solution. For each operation, the minimum number of nodes that should be copied are randomly determined between 1% to 40% of total nodes, and the above operation is repeated until the total number of copied nodes exceeds the number.

For mutation, random merge and split operators were introduced. The random merge was carried out by combining two adjacent communities. The random split operator divides a community into two randomly assigned groups. All trial conformations generated by crossover and mutation operations are optimized by quench procedures. It should be noted that only *local* moves are used in the quench procedures since the divergent and divisive copy operators can act as the merge and split moves used in SA.

After local maximization of trial partitions, these partitions are used to update the bank. The modularity of a trial partition α is compared with the modularities of partitions in the bank. If α is worse than the worst partition of the bank, it is discarded. Otherwise, we find the partition A in the bank which is closest to α, as determined by distance $D(\alpha,A)$. If $D(\alpha,A) < D_{\text{cut}}$, α is considered as similar to A, and it replaces A if $\alpha > A$. If $\alpha < A$ it is discarded. If $D(\alpha,A) > D_{\text{cut}}$, α is regarded as a new partition similar to none in the bank, and it replaces the worst existing partition, that is, it replaces the lowest modularity partition in the bank. We carry out this operation for all trial partitions. With updated bank, new seed partitions are selected

TABLE I. Number of nodes and edges of benchmark networks used in this study.

Network	Nodes	Edges
Dolphins	62	159
Les Misérables	77	254
Political books	105	441
College football	115	613
Jazz	198	2742
USAir97	332	2126
Netscience_main	379	914
C. elegans	453	2025
Electronic circuit (s838)	512	819
E-mail	1133	5451
Erdos02	6927	11 850
PGP	10 680	24 316
condmat2003	27 519	116 181

again from the bank which have not yet been used as seeds. This entire process of generating partitions by perturbation and subsequent local maximization and updating the bank is repeated until all partitions in the bank are used as seeds. At each iteration step, D_{cut} is reduced with a predetermined ratio. After D_{cut} reaches its final value, $D_{\text{avg}}/5$, it is kept constant.

Once all partitions in the bank are used as seeds without generating better partitions, implying that the procedure might have reached a deadlock, we reset all bank partitions to be eligible for seeds and repeat another round of search procedure. After this additional search also reaches a deadlock, we expand our search space by adding an additional 50 randomly generated and optimized partitions to the bank, and repeat the whole procedure. In this study we terminated our calculation after 100 partitions were used as seeds. Additional adding cycles should be considered for more rigorous optimization, especially for problems with high complexity.

III. RESULTS

To compare performance of CSA and SA, we applied CSA and SA to a number of real-world networks commonly used in existing modularity optimization studies, shown in Table I. All networks considered are undirected and unweighted. Due to the stochastic nature of both methods, we performed 50 independent simulations for each method. The results are summarized in Table II. The maximum, average, and standard deviation of modularity values obtained by both methods are displayed. As a measure of required computational resources, we counted the number of function evaluations performed until the maximum modularity solution is found, $N^{\max}$. We observe that CSA consistently finds equal or higher modularity solutions than does SA for all networks tested, with a smaller number of function evaluations. To demonstrate the search efficiency of CSA more clearly, we also measured the number of function evaluations required by CSA to generate a solution equivalent to the best modularity obtained by SA, which is denoted as $N_{\text{CSA}}^{\text{equal}}$ in Table II. CSA clearly requires many fewer function evaluations to generate solutions better than the best ones obtained by SA. For small networks (e.g., up to the Jazz musician network), CSA finds the best solution with less than 10% of the function evaluations required by SA, and for the worst case, the *C. elegans* network, CSA requires only 25% of the function evaluations of SA.

It should be noted that CSA can be applied to networks containing more than 10^3 nodes where for SA this is impractical. For the three largest networks in Table II, CSA found good solutions within a reasonable computational time, whereas SA runs did not yield reasonable value of modularity within

TABLE II. Modularity optimization results obtained by 50 separate runs of Mod-CSA and SA are displayed. $Q_{\max}$ denotes the maximum modularity value, Q_{avg} the average of maximum modularity value of each run, σ the standard deviation of the modularity value, $N^{\max}$ the number of function evaluations until the calculation reached the maximum modularity, $N_{\text{CSA}}^{\text{equal}}$ the number of function evaluations required for CSA runs to sample equal or higher modularity solutions than the maximum modularity of SA runs, t the average execution time to find the best solution in seconds. All simulations were performed on an Intel Nehalem core at 2.93 GHz. CSA and SA runs were performed with 64 cpus and single cpu, respectively. For the three largest networks, SA results are not available because calculations were not finished within 2 days.

	CSA			SA						
Network	$Q_{\max}$	Q_{avg}	σ	$Q_{\max}$	Q_{avg}	σ	$N_{\text{CSA}}^{\max}/N_{\text{SA}}^{\max}$	$N_{\text{CSA}}^{\text{equal}}/N_{\text{SA}}^{\max}$	t_{CSA}	t_{SA}
Dolphins	0.52852	0.52852	0	0.52852	0.52507	0.0036	0.077	0.077	0.09	0.74
Les Misérables	0.56001	0.56001	0	0.56001	0.55194	0.0071	0.362	0.362	0.01	0.18
Political books	0.52724	0.52724	0	0.52724	0.52723	0	0.055	0.019	0.07	2.52
College football	0.60457	0.60457	0	0.60457	0.60457	0	0.093	0.093	0.05	0.26
Jazz	0.44514	0.44514	0	0.44487	0.44477	1.6e-4	0.073	0.052	0.17	679.4
USAir97	0.36824	0.36824	0	0.35376	0.34787	0.0044	0.271	0.010	0.13	429.2
Netscience_main	0.84859	0.84859	0	0.84383	0.83544	0.0044	0.345	0.019	1.3	263.3
C. elegans	0.45325	0.45325	0	0.45212	0.44927	0.0026	0.960	0.246	16.8	2512.3
Electronic circuit (s838)	0.81936	0.81936	0	0.81871	0.80812	0.0048	0.639	0.424	2.6	1129.4
E-mail	0.58283	0.58282	2.2e-5	0.58198	0.58015	0.0015	0.510	0.119	73.6	42 296
Erdos02	0.71843	0.71782	3.2e-4	–	–	–	–	–	3356	–
PGP	0.88675	0.88648	1.1e-4	–	–	–	–	–	10 757	–
condmat2003	0.76745	0.76484	0.0010	–	–	–	–	–	57 609	–

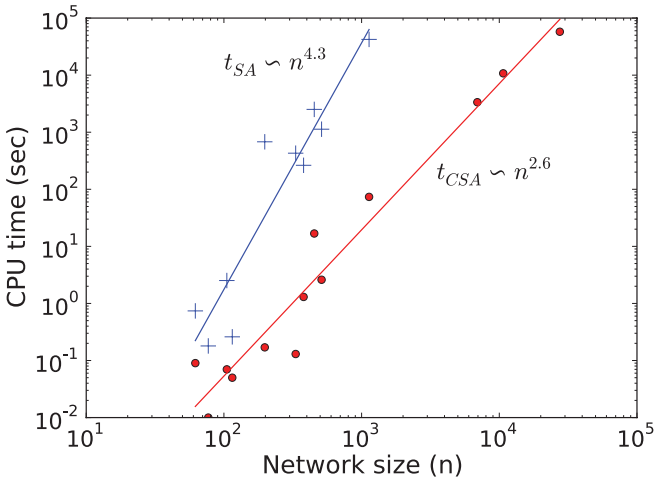

FIG. 2. (Color online) Comparison of time complexities of CSA (red, circle) and SA (blue, cross) is shown. Network size n represents the number of nodes, and CPU time corresponds to the average time to find final solutions in seconds.

2 days of wall clock time. This difference in computational time reflects a number of factors. It is partly due to the high parallel efficiency of the CSA algorithm [35]. In SA generation of a trial solution is dependent on its previous state, which makes it impractical to implement the algorithm in a parallel fashion. However, the majority of computation in CSA consists of independent local maximization procedures on hundreds of trial solutions generated by crossover and mutation, and each maximization can be separately carried out. The quench procedure in CSA consists of local moves only, which is rather fast with large networks. On the other hand, the most time-consuming operation in SA is the splitting move by the nested SA procedure, which we find is indeed essential to obtain good SA solutions. In CSA the operation of the divisive copy when generating trial solutions plays the equivalent role of the split move in SA. To compare the computing efficiency of CSA with existing methods, the time complexities of CSA and SA are estimated based on the simulation results with the benchmark networks. As shown in Fig. 2, the time complexity of CSA is estimated to be $O(n^{2.6})$, which is comparable to other heuristic methods [36,37] and better than that of SA, $O(n^{4.3})$, where n is the number of nodes.

In terms of convergence, CSA yields more robust solutions than SA. Except for the political books and college football networks, the maximum modularity solution found by SA varied from simulation to simulation. For networks containing over 300 nodes, SA failed to sample the optimal solution, which raises serious concerns when applying SA to modularity optimization for practical use [11]. However, for all test networks up to about 10^3 nodes, all CSA runs converged to the same solution, except the e-mail network, where 41 out of 50 converged. Considering the small size of networks and the stochastic nature of the algorithm, we believe that the converged solution of each network is likely to be the true maximum modularity of the network.

We also compared maximum modularities obtained by CSA with the maximum values from previous publications; see Table III. CSA finds equivalent or higher Q values compared to existing studies in all networks tested.

Recently the exact maximum modularity values of several small benchmark networks up to 512 nodes were reported; they are displayed in Table III as Q_{opt} [29]. We performed 50 independent runs for these networks, and all runs converged to the optimal solutions *without exception*. This result supports the hypothesis that CSA is efficient enough to find the putative maximum modularity solution for a network containing up to 10^3 nodes.

CSA algorithm presented in this work aims to obtain optimal modularity solutions, and the method is not free from the problem of the resolution limit arising from using modularity [38]. However, the CSA procedure and operators proposed in this work are general and can be used to optimize other fitness functions. To overcome the resolution limit issue, more robust fitness functions should be considered to be combined with

TABLE III. Comparison between the maximum modularity values obtained by CSA, $Q_{\max}$, with previously published ones, Q_{pub}, and the maximum values obtained by the exact method [29], Q_{opt}, is displayed. N_c denotes the number of communities found by CSA. Source indicates the reference that the modularity value is collected. $\%_{\text{opt}}^{\text{SA}}$ denotes the percentage of SA runs that reached the optimal modularity community structure.

	CSA					
Network	N_c	$Q_{\max}$	Q_{pub}	Q_{opt}	$\%_{\text{opt}}^{\text{SA}}$	Source
Dolphins	5	0.52852	0.5285	0.5285	16.0	[27–29]
Les Misérables	6	0.56001	0.5600	0.5600	20.0	[29]
Political books	5	0.52724	0.5272	0.5272	100.0	[28–30]
College football	10	0.60457	0.6046	0.6046	100.0	[28,29,31]
Jazz	4	0.44514	0.4451	–	–	[28,30,32,33]
USAir97	6	0.36824	0.3682	0.3682	0.0	[29]
Netscience_main	19	0.84859	0.8486	0.8486	0.0	[29]
C. elegans	9	0.45325	0.452	–	–	[34]
Electronic circuit (s838)	16	0.81936	0.8194	0.8194	0.0	[29]
E-mail	10	0.58283	0.582	–	–	[34]
Erdos02	40	0.71843	0.7162	–	–	[30]
PGP	100	0.88674	0.8841	–	–	[30,34]
condmat2003	80	0.76745	0.761	–	–	[31]

CSA, such as the map equation [39] or the partition density [7]. It should be noted that the current work can be extended to deal with directed or weighted networks in conjunction with modified modularity functions [40,41]. In order to handle large networks, CSA can be combined with other efficient heuristics, such as the fast unfolding method [42], instead of the stochastic quench procedure used in this study.

IV. CONCLUSION

In this paper, we propose a new modularity optimization method based on conformational space annealing algorithm, Mod-CSA. Compared to SA, our method is faster. Further, while it finds equivalent modularity partitions for relatively small networks, for the larger more challenging ones, it typically finds higher modularity partitions. For small networks consisting up to 10^3 nodes, despite its stochastic nature, Mod-CSA solutions converge to an identical solution, which appears to be the best solution possible; this is not possible in other stochastic algorithms. Mod-CSA can be implemented in a highly parallel fashion and is thus applicable to large networks where SA is not. In addition, Mod-CSA can be extended to deal with large networks by using fast heuristic methods such as a local optimizer.

ACKNOWLEDGMENTS

The authors acknowledge support from Creative Research Initiatives (Center for In Silico Protein Science, 20110000040) of MEST/KOSEF. We thank the Korea Institute for Advanced Study for providing computing resources (KIAS Center for Advanced Computation Linux Cluster) for this work.

[1] M. E. J. Newman, A.-L. Barabási, and D. J. Watts, *The Structure and Dynamics of Networks* (Princeton University Press, Princeton, 2006), p. 624.
[2] G. Caldarelli, *Scale-Free Networks: Complex Webs in Nature and Technology* (Oxford University Press, Oxford, 2007), p. 336.
[3] M. Girvan and M. E. J. Newman, Proc. Nat. Acad. Sci. USA **99**, 7821 (2002).
[4] S. Fortunato, Phys. Rep. **486**, 75 (2010).
[5] S. Wasserman, K. Faust, and D. Iacobucci, *Social Network Analysis* (Cambridge University Press, Cambridge,1994).
[6] S. Schaeffer, Comput. Sci. Rev. **1**, 27 (2007).
[7] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, Nature (London) **466**, 761 (2010).
[8] M. E. J. Newman and M. Girvan, Phys. Rev. E **69**, 026113 (2004).
[9] M. E. J. Newman, Phys. Rev. E **69**, 066133 (2004).
[10] U. Brandes *et al.*, IEEE Trans Knowl. Data Eng. **20**, 172 (2008).
[11] B. H. Good, Y. A. de Montjoye, and A. Clauset, Phys. Rev. E **81**, 046106 (2010).
[12] R. Guimerà and L. A. N. Amaral, Nature (London) **433**, 895 (2005).
[13] Y. Sohn, M.-K. Choi, Y.-Y. Ahn, J. Lee, and J. Jeong, PLoS Comput. Biol. **7**, e1001139 (2011).
[14] Z. Wang and J. Zhang, PLoS Comput. Biol. **3**, e107 (2007).
[15] J. Lee, H. Scheraga, and S. Rackovsky, J. Comput. Chem. **18**, 1222 (1997).
[16] J. Lee, A. Liwo, and H. A. Scheraga, Proc. Natl. Acad. Sci. USA **96**, 2025 (1999).
[17] J. Lee, I. H. Lee, and J. Lee, Phys. Rev. Lett. **91**, 080201 (2003).
[18] K. Joo, J. Lee, I. Kim, S. Lee, and J. Lee, Biophys. J. **95**, 4813 (2008).
[19] J. Lee, J. Lee, T. N. Sasaki, M. Sasai, C. Seok, and J. Lee, Proteins **79**, 2403 (2011).
[20] R. Guimerà and L. A. N. Amaral, J. Stat. Mech. (2005) P02001.
[21] Z. Li and H. A. Scheraga, Proc. Natl. Acad. Sci. USA **84**, 6611 (1987).
[22] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley Professional, New York, 1989).
[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Science **220**, 671 (1983).
[24] M. Meila, J. Multivariate Anal. **98**, 873 (2007).
[25] W. Rand, J. Am. Stat. Assoc. **66**, 846 (1971).
[26] A. L. Fred and A. K. Jain, IEEE Computer Society Conference on Computer Vision and Pattern Recognition **2**, 128 (2003).
[27] G. Xu, S. Tsoka, and L. Papageorgiou, Eur. Phys. J. B **60**, 231 (2007).
[28] G. Agarwal and D. Kempe, Eur. Phys. J. B **66**, 409 (2008).
[29] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, and L. Liberti, Phys. Rev. E **82**, 046112 (2010).
[30] A. Noack and R. Rotta, in *Experimental Algorithms*, Lecture Notes in Computer Science, Vol. 5526, edited by J. Vahrenhold (Springer, Berlin, 2009), p. 257.
[31] Z. Ye, S. Hu, and J. Yu, Phys. Rev. E **78**, 046115 (2008).
[32] P. Schuetz and A. Caflisch, Phys. Rev. E **77**, 046112 (2008).
[33] J. Duch and A. Arenas, Phys. Rev. E **72**, 027104 (2005).
[34] X. Liu and T. Murata, Physica A **389**, 1493 (2010).
[35] J. Lee, J. Pillardy, C. Czaplewski, Y. Arnautova, D. Ripoll, A. Liwo, K. Gibson, R. Wawak, and H. Scheraga, Comput. Phys. Commun. **128**, 399 (2000).
[36] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, J. Stat. Mech. (2005) P09008.
[37] A. Lancichinetti and S. Fortunato, Phys. Rev. E **80**, 056117 (2009).
[38] S. Fortunato and M. Barthélemy, Proc. Nat. Acad. Sci. USA **104**, 36 (2007).
[39] M. Rosvall and C. T. Bergstrom, Proc. Natl. Acad. Sci. USA **105**, 1118 (2008).
[40] M. E. J. Newman, Phys. Rev. E **70**, 056131 (2004).
[41] A. Arenas, J. Duch, A. Fernández, and S. Gómez, New J. Phys. **9**, 176 (2007).
[42] V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech. (2008) P10008.