# Dynamic Community Detection Algorithm Based on Automatic Parameter Adjustment

Kai Lu[1,2], Xin Wang[1,2(✉)], and Xiaoping Wang[1,2]

[1] College of Computer Science, National University of Defense Technology, Changsha, People's Republic of China
{kailu,xiaopingwang}@nudt.edu.cn, dywangxin@foxmail.com
[2] Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, People's Republic of China

**Abstract.** Community detection is widely used in social network analysis. It clusters densely connected vertices into communities. As social networks get larger, scalable algorithms are drawing more attention. Among those methods, the algorithm named Attractor is quite outstanding both in terms of accuracy and scalability. However, it is highly dependent on the parameter, which is abstract for users. The improper parameter value can bring about some problems. There can be a huge community (monster) sometimes; other time the communities are generally too small (fragments). The existing fragments also need eliminating. Such phenomenon greatly deteriorates the performance of Attractor. We modify the algorithm and propose mAttractor, which adjusts the parameter automatically. We introduce two constraints to limit monsters and fragments and to narrow the parameter range. An optional parameter is also introduced. The proposed algorithm can choose to satisfy or ignore the optional parameter by judging whether it is reasonable. Our algorithm also eliminates the existing fragments. A delicate pruning is designed for fast determination. Experiments show that our mAttractor outperforms Attractor by 2%–270%.

**Keywords:** Community detection · Social network · Data mining

## 1 Introduction

Community detection is to cluster nodes into communities so that nodes are densely connected inside communities [19]. It can reveal fundamentally related entities in various networks [7], and is widely used in statistics, biology, and computer science [4,17].

There are various methods of community detection, but most of them are not satisfying either in accuracy or in scalability. The algorithm named Attractor [22] is outstanding in both areas. Nonetheless, Attractor highly depends on the parameter, which is quite abstract for users. The blindly determined parameter is likely to bring about some problems. There can be a huge community sometimes, which is usually called a monster; other time the communities are


H. Yin et al. (Eds.): IDEAL 2017, LNCS 10585, pp. 9–19, 2017.
https://doi.org/10.1007/978-3-319-68935-7_2

generally too small, which are named fragments. Moreover, whatever the parameter value is, the fragments always exist, which impact the performance unless properly eliminated. Such phenomenon makes it difficult for Attractor to achieve its satisfying performance.

We proposed a modified algorithm, mAttractor, which can adjust the parameter and eliminate the fragments. By two constraints it regulates the quality and the parameter range. It also eliminates the fragments and utilizes the requirement of users. Our main contributions are listed as follows:

1. We produce a mechanism to adjust the parameter automatically.
2. Our algorithm can limit the monsters and the fragments.
3. It can eliminate fragments and maintain the scalability.
4. Our algorithm can choose to satisfy or ignore the user-specified parameter by judging whether it will deteriorate result.

The rest of the paper is organised as follows. Section 2 gives the related works. Section 3 discusses the algorithm and Sect. 4 shows the experiments. The paper is concluded in Sect. 5.

## 2 Related Works

As social networks get larger, community detection is required to be both accurate and scalable. Most algorithms, however, are not always satisfying in either aspect. The Markov Cluster Algorithm (MCL) [24] is quite reasonable, but it suffers from scalability. The Label Propagation Algorithm (LPA) [15,20,26] is fast, but it is neither accurate nor stable. Algorithms based on artificial criteria, such as Modularity [18] and Normalized Cut [23], are influential because of their high performance. But they can lead to the resolution limit [5], in which case the small communities are neglected.

Among those methods, Attractor is outstanding both in terms of accuracy and scalability. It views the graph as a dynamical system, where nodes interact with each other. By the interactions, some nodes are moving nearer while others are further, and communities are detected then. Attractor is scalable, stable and accurate. Because it works naturally, it is free from the resolution limit.

Nonetheless, Attractor highly depends on a parameter $\lambda$, which determines the direction of influence of exclusive adjacent nodes. To determine $\lambda$, therefore, is abstract for users. Users may determine $\lambda$ blindly, which can result in poor performance. Sometimes there can be a huge community (monster); other time too many communities become fragments which have only a few nodes. Even though we limit the percentage of fragments, some still remain. In a word, because the parameter is hard to determine and the fragments need eliminating, the algorithm performance is greatly deteriorated.

In our proposed mAttractor, $\lambda$ is adjusted automatically and the fragments are eliminated while the scalability is maintained.

## 3 The Detailed Description of the Algorithm

This section firstly studies $\lambda$. Then we add constraints on the result. The fragments elimination is given then, followed by the entire algorithm.

### 3.1 The Influence of Parameter

We use the Lancichinetti-Fortunato-Radicchi (LFR) benchmark [12,13] to generate graphs with 200 nodes. The average degree ranges from 10 to 50. For each network, we repeatedly run Attractor with increasing $\lambda$, and study the community number. We also extend the range of $\lambda$ from $[0, 1]$ to $[0, +\infty)$. In Fig. 1, as $\lambda$ grows, the community number roughly increases and the average community size decreases. So we can adjust $\lambda$ by such correlation. If there are only a few communities, we can increase $\lambda$; otherwise, we can decrease $\lambda$.

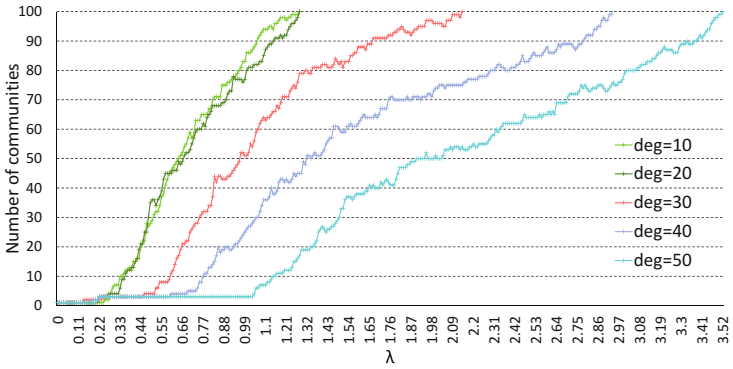



**Fig. 1.** The influence of $\lambda$

The theoretical analysis also demonstrates our study. The parameter $\lambda$ controls the direction of the influence of exclusive adjacent nodes. The higher $\lambda$, the more repulsive forces and the less attractive forces. So the algorithm yields more communities if $\lambda$ is higher and larger communities if $\lambda$ is lower.

### 3.2 The Two Constraints

Because $\lambda$ impacts the result, we introduce two constraints to regulate the result as well as $\lambda$. Firstly we define the undirected graph, which is the algorithm input.

**Definition 1 (The Undirected Graph).** *Let $G = (V, E)$ donate an undirected graph, where $V = \{v_1, v_2 \dots v_n\}$ is the set of nodes, $E = \{e_1, e_2 \dots e_m\}$ is the set of edges and $e = \{v_i, v_j\} \in E$ indicates an edge between $v_i$ and $v_j$.*

After community detection, the graph is divided into parts and each node is mapping to its community.

**Definition 2 (The Community Affiliation).** *Given a graph $G = (V, E)$ and the set of community labels $L = \{l_1, l_2 \dots l_k\}$, the community affiliation $C : V \rightarrow L$ maps the given node $v \in V$ to its community $C(v) \in L$.*

Then we can analyse the result. In case of fragments, we claim that the smallest community has three nodes. So smaller communities are viewed as fragments, whose nodes are outliers. The percentage of outliers reflects whether the result is generally fragmentary. We also study the largest community in case of a monster.

**Definition 3 (The Fragment).** *Given a graph $G = (V, E)$ and a community affiliation $C$, if the community $l_f \in L$ has less than 3 nodes, then it is called a fragment or a fragmentary community, and its nodes are called outliers.*

**Definition 4 (The Fragment Rate).** *Given a graph $G = (V, E)$ and a community affiliation $C$, suppose $V_f$ is the set of all the outliers, then the fragment rate $R_f(G, C)$ is the percentage of outliers.*

$$R_f(G, C) = \frac{|V_f|}{|V|} \tag{1}$$

**Definition 5 (The Monster Rate).** *Given a graph $G = (V, E)$ and a community affiliation $C$, if the largest community is $l_m \in L$, then the monster rate $R_m(G, C)$ is the ratio of its size to the total number of nodes.*

$$R_m(G, C) = \frac{|\{v \in V | C(v) = l_m\}|}{|V|} \tag{2}$$

We introduce two constraints to limit the fragments and the monster.

1. Given a graph $G = (V, E)$ and a community affiliation $C$, the fragment rate $R_f(G, C)$ should be less than 25%.
2. Given a graph $G = (V, E)$ and a community affiliation $C$, the monster rate $R_m(G, C)$ should be less than 70%.

Then we can adjust $\lambda$ by the constraints. Communities are generally fragmentary if Constraint 1 is violated, so $\lambda$ is supposed to decrease. If Constraint 2 is violated, there is a monster community and we need to increase $\lambda$ for more communities. In fact, the constraints regulate the upper and lower bound of $\lambda$, respectively. Each time one constraint is violated, mAttractor narrows the range of $\lambda$. We keep adjusting $\lambda$ until both constraints are satisfied, or they are judged never to be satisfied at the same time.

### 3.3 The Elimination of Fragments

After we limit the fragments, we still need to eliminate the remaining fragments.

The key is to maintain the scalability of the algorithm. Inspired by the Label Propagation Algorithm (LPA) [20] which is linear to the graph size, we eliminate the outliers in descending order by the degree. In Algorithm 1, each outlier

is appointed to the majority community in its neighbourhood. The majority community is the one with the most votes after each neighbour node votes for its community by the weight of degree.

We keep running Algorithm 1 until there are only isolated nodes remaining, which cannot be eliminated because they have no edge at all.

**Algorithm 1.** Elimination

**Input:** $G = (V, E)$, $C : V \to L$, fragment node list $Flist$;
Reorder the nodes in $Flist$ in descending order by the degree;
**for** each $v \in Flist$ **do**
    Find the neighbour set $\Gamma(v) = \{v' \in V | \{v, v'\} \in E\}$;
    Initialize the vote function $f(l) = 0, \forall l \in L$;
    **for** each $v' \in \Gamma(v)$ **do**
        Update the vote function $f(C(v')) = f(C(v')) + deg(C(v'))$;
    **end for**
    Find $l_{max} \in L$ with the maximum votes, randomly choose one if many;
    Update the affiliation $C(v) = l_{max}$;
**end for**
**Return:** the modified community affiliation $C$;

### 3.4 The Algorithm

Algorithm 2 shows the parameter adjustment, whose step halves if the direction toggles or $\lambda$ steps out of the regulated range. Algorithm 3 shows mAttractor, which is designed to call Attractor and analyse the result repeatedly. After calling Attractor, we run Algorithm 1 and study $R_f(G, C)$ and $R_m(G, C)$. We consider the fragments before elimination when studying $R_f(G, C)$. We adjust $\lambda$, call Attractor and restart the analysis until the constraints are satisfied or they are discovered impossible to be met.

Then we adjust $\lambda$ so that the community number is close to an optional parameter, $target$, which is the targeted number of communities. The algorithm will adjust $\lambda$ until the number is within 10% more or less than $target$. Also, $target$ should satisfy the two constraints. The constraints regulate the range of the reasonable community number. If $target$ becomes out of the range, it violates the constraints and then becomes illegal. But as long as it is legal, it is influential. We do not encourage users to appoint $target$ unless they know the exact number. We set $target = 0$ if it is not appointed.

After the two constraints are satisfied, the algorithm terminates if the community number is close to the parameter $target$, or $target$ is illegal or not appointed. It also terminates when the adjustment step or the range of $\lambda$ is smaller than the arbitrarily small positive quantity $\epsilon$.

**Algorithm 2.** Change

**Input:** $\lambda$, $step$, $dir$, $dir_{old}$, $ubound_\lambda$, $lbound_\lambda$;
**if** $dir_{old} \neq dir$ **then**
  $step = step/2$;
**end if**
**if** $dir == `up'$ **then**
  $\lambda = \lambda + step$;
  **if** $\lambda \geq ubound_\lambda$ **then**
    $\lambda = (\lambda - step + ubound_\lambda)/2$;
    $step = step/2$;
  **end if**
**else**
  $\lambda = \lambda - step$;
  **if** $\lambda \leq lbound_\lambda$ **then**
    $\lambda = (\lambda + step + lbound_\lambda)/2$;
    $step = step/2$;
  **end if**
**end if**
**Return:** $\lambda$, $step$, $dir$;

## 4 Experiments

### 4.1 Experiment Setup

We choose ten networks for experiments, as shown in Table 1. Six networks have ground truth, including the Karate [25], the Dolphins [16], the Lesmis [11], the Mexican [3,6], the Polblogs [1] and the Company [2]. The ground truth can give the correct result and the number *target*. There are also four graphs without ground truth, including the Email [9], the Jazz [8], the Ca-GrQc [14] and the Ca-HepTh [14]. Circles are removed from those graphs.

We use Rand Index (RI) [10,21] and Adjusted Rand Index (ARI) [10,21] to study the accuracy performance on networks with ground truth. Then we use all networks to study Modularity [18], an influential quality measurement of community detection. At last, we study the scalability of the algorithm.

In Attractor, $\lambda \in [0, 1]$, so we set $\lambda = 0.3, 0.5, 0.7$ to represent the blind parameter determination and the best performance of the three is viewed as Attractor's performance. Our mAttractor runs in two modes. In the targeted mode, *target* equals to the real number of communities; in the untargeted mode, *target* is not appointed and equals to 0. For graphs without ground truth, mAttractor only runs in the untargeted mode. The arbitrarily small positive quantity $\epsilon = 0.01$.

### 4.2 Accuracy Performance

The accuracy performance is shown in Fig. 2. In the Karate and the Dolphins, Attractor produces a monster when $\lambda = 0.3, 0.5$. ARI is sensitive to such terrible

**Algorithm 3.** mAttractor

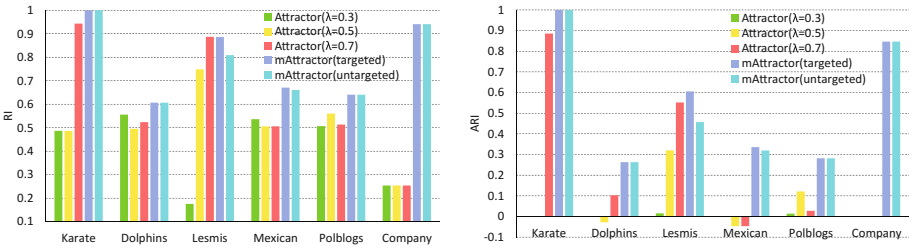
**Input:** $G = (V, E)$, $target$, $\epsilon$;
$dir_{old} = `up', \lambda = 0.5, step = 0.25$;
$lbound_\lambda = 0, ubound_\lambda = \infty, lbound_c = 1, ubound_c = |V|/3$;
**while** ($step \geq \epsilon$ and $(ubound_\lambda - lbound_\lambda) \geq \epsilon$) **do**
    Call $C = Attractor(G, \lambda)$;
    Store outliers into $Flist$ and $Flist_{original}$, $Flist_{old} = V$;
    **while** $|Flist| < |Flist_{old}|$ **do**
        $C = Elimination(G, C, Flist)$;
        $Flist_{old} = Flist$, store all outliers into $Flist$;
    **end while**
    $n =$ the number of communities in $C$;
    **if** $|Flist_{original}| \geq 0.25|V|$ **then**
        $ubound_\lambda = min(ubound_\lambda, \lambda), ubound_c = min(ubound_c, n)$;
        $\lambda, step, dir_{old} = Change(\lambda, step, `down', dir_{old}, ubound_\lambda, lbound_\lambda)$;
        **continue**;
    **end if**
    **if** $R_m(G, C) \geq 0.7$ **then**
        $lbound_\lambda = max(lbound_\lambda, \lambda), lbound_c = max(lbound_c, n)$;
        $\lambda, step, dir_{old} = Change(\lambda, step, `up', dir_{old}, ubound_\lambda, lbound_\lambda)$;
        **continue**;
    **end if**
    **if** (not $(lbound_c < target < ubound_c)$ or $(0.9target < n < 1.1target)$) **then**
        **break**;
    **end if**
    **if** $target > n$ **then**
        $\lambda, step, dir_{old} = Change(\lambda, step, `up', dir_{old}, ubound_\lambda, lbound_\lambda)$;
    **else**
        $\lambda, step, dir_{old} = Change(\lambda, step, `down', dir_{old}, ubound_\lambda, lbound_\lambda)$;
    **end if**
**end while**
**Output:** the community affiliation $C$;

results, and it is either zero or minus. It works well when $\lambda = 0.7$. But mAttractor outperforms Attractor by 6% in RI and 13% in ARI on the Karate, and 9% in RI and 156% in ARI on the Dolphins. Particularly, mAttractor achieves a perfectly correct result on the Karate. For the Lesmis, Attractor works well when $\lambda = 0.5, 0.7$, and mAttractor is better in the targeted mode. For the Polblogs, Attractor yields a monster when $\lambda = 0.3$ and too many fragments when $\lambda = 0.7$. It is satisfying when $\lambda = 0.5$, but mAttractor still outperforms Attractor by 132% in ARI and 14% in RI. Things are similar for the Mexican and the Company. Attractor always produces a huge monster, while mAttractor performs 25% better in RI on the Mexican and 270% better in RI on the Company.

Our mAttractor runs in two modes, but the result is usually the same. Sometimes the targeted mode can be better (the Mexican and the Lesmis). The algorithm can work well without *target*, the parameter can produce better results.

**Table 1.** Basic information of datasets

| Dataset | #Vertices | #Edges | #Communities |
|---|---|---|---|
| Karate | 34 | 78 | 2 |
| Dolphins | 62 | 159 | 2 |
| Lesmis | 77 | 254 | 11 |
| Mexican | 35 | 117 | 2 |
| Polblogs | 1224 | 19087 | 2 |
| Company | 77 | 2326 | 4 |
| Email | 1133 | 10902 | / |
| Jazz | 198 | 5484 | / |
| Ca-GrQc | 5241 | 28968 | / |
| Ca-Hepth | 9875 | 51946 | / |




(a) RI Performance (b) ARI Performance

**Fig. 2.** The accuracy performance

### 4.3 Modularity Performance

The Modularity performance is shown in Fig. 3. For the first three graphs, Attractor performs well when $\lambda = 0.7$, but mAttractor outperforms Attractor by 2% on the Karate, 17% on the Dolphins, 7% (targeted) and 3% (untargeted) on the Lesmis. As for the Mexican, mAttractor performs much better than Attractor. Attractor performs well on the Polblogs when $\lambda = 0.5$, while mAttractor performs a little better. In the Company, although Attractor has higher Modularity, such performance is meaningless because of the monster it produces.

For the networks without ground truth, we cannot appoint *target*, so we only consider the untargeted mode. In the Email, mAttractor performs 6% better. In the Jazz, Attractor produces a huge monster when $\lambda = 0.3$ and 0.5 but performs well when $\lambda = 0.7$. Our mAttractor performs 3% better by eliminating the fragments. As for the Ca-GrQc and the Ca-Hepth, Attractor performs well most of the time, but mAttractor still outperforms Attractor by 3%–4%.

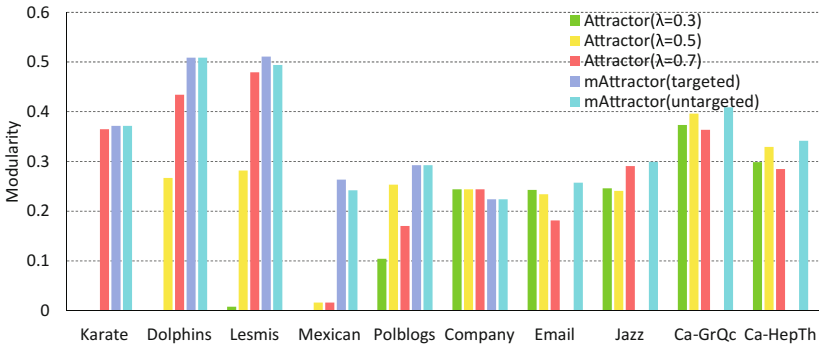



**Fig. 3.** The modularity performance

### 4.4 Scalability Study

We use LFR benchmark to generate networks for scalability study. The node size ranges from 2000 to 30000, and the average degree is set to 10. For Attractor, $\lambda = 0.5$; for mAttractor, it works in the untargeted mode.

In Fig. 4, we can see that mAttractor is scalable with its linear time complexity. It calls Attractor repeatedly, so it is slower. However, mAttractor achieves a better result than the three results of Attractor ($\lambda = 0.3, 0.5, 0.7$), so we claim such cost is acceptable.

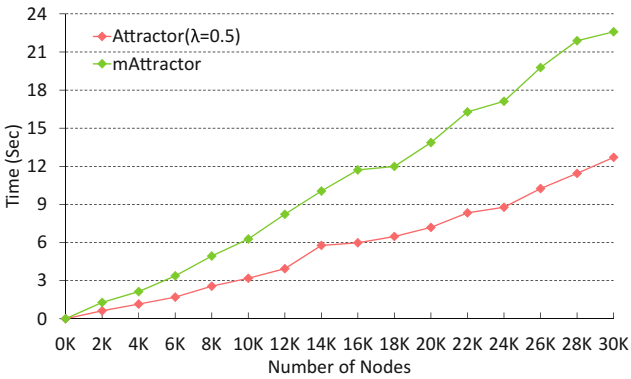



**Fig. 4.** The running time

## 5 Conclusion

We propose mAttractor to adjust the parameter based on the current result. Introduced constraints can guarantee the quality and narrow the range of $\lambda$. It also eliminates the fragments to optimize the result. It can also choose to utilize

the user-specified parameter by judging whether it is reasonable. Experiments demonstrate mAttractor can achieve higher accuracy and Modularity, with a linear time complexity.

**Acknowledgement.** This work is partially supported by The National Key Research and Development Program of China (2016YFB0200401), by program for New Century Excellent Talents in University, by National Science Foundation (NSF) China 61402492, 61402486, 61379146, by the laboratory pre-research fund (9140C810106150C81001).

## References

1. Adamic, L.A., Glance, N.: The political blogosphere and the 2004 US election: divided they blog. In: Proceedings of the 3rd International Workshop on Link Discovery, pp. 36–43. ACM (2005)
2. Cross, R., Parker, A., Christensen, C.M., Anthony, S.D., Roth, E.A.: The hidden power of social networks. J. Appl. Manag. Entrepreneurship **9** (2004)
3. De Nooy, W., Mrvar, A., Batagelj, V.: Exploratory Social Network Analysis with Pajek, vol. 27. Cambridge University Press, Cambridge (2011)
4. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
5. Fortunato, S., Barthelemy, M.: Resolution limit in community detection. Proc. Nat. Acad. Sci. U.S.A. **104**(1), 36–41 (2007)
6. Gil-Mendieta, J., Schmidt, S.: The political network in Mexico. Soc. Netw. **18**(4), 355–381 (1996)
7. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Nat. Acad. Sci. U.S.A. **99**(12), 7821–7826 (2002)
8. Gleiser, P.M., Danon, L.: Community structure in jazz. Adv. Complex Syst. **6**(04), 565–573 (2003)
9. Guimera, R., Danon, L., Diaz-Guilera, A., Giralt, F., Arenas, A.: Self-similar community structure in a network of human interactions. Phys. Rev. E **68**(6), 065103 (2003)
10. Hubert, L., Arabie, P.: Comparing partitions. J. Classif. **2**(1), 193–218 (1985)
11. Knuth, D.E.: The Stanford GraphBase: A Platform for Combinatorial Computing, vol. 37. Addison-Wesley, Reading (1993)
12. Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **80**(1), 016118 (2009)
13. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **78**(4), 046110 (2008)
14. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: densification and shrinking diameters. ACM Trans. Knowl. Discov. Data (TKDD) **1**(1), 2 (2007)
15. Leung, I.X.Y., Hui, P., Lio, P., Crowcroft, J.: Towards real-time community detection in large networks. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **79**(6), 066107 (2009)
16. Lusseau, D., Schneider, K., Boisseau, O.J., Haase, P., Slooten, E., Dawson, S.M.: The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. Behav. Ecol. Sociobiol. **54**(4), 396–405 (2003)
17. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp. 29–42. ACM (2007)

18. Newman, M.E.J.: Modularity and community structure in networks. Proc. Nat. Acad. Sci. U.S.A. **103**(23), 8577–8582 (2006)
19. Porter, M.A., Onnela, J.P., Mucha, P.J.: Communities in networks. Not. Am. Math. Soc. **56**(9), 4294–4303 (2009)
20. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **76**(3), 036106 (2007)
21. Rand, W.M.: Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc. **66**(336), 846–850 (1971)
22. Shao, J., Han, Z., Yang, Q., Zhou, T.: Community detection based on distance dynamics. In: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1075–1084. ACM (2015)
23. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000)
24. Van Dongen, S.M.: Graph clustering by flow simulation. Ph.D. thesis, University of Utrecht (2001)
25. Zachary, W.W.: An information flow model for conflict and fission in small groups. J. Anthropol. Res. **33**(4), 452–473 (1977)
26. Zhang, X.-K., Fei, S., Song, C., Tian, X., Ao, Y.-Y.: Label propagation algorithm based on local cycles for community detection. Int. J. Mod. Phys. B **29**(05), 1550029 (2015)

http://www.springer.com/978-3-319-68934-0

Intelligent Data Engineering and Automated Learning – IDEAL 2017
18th International Conference, Guilin, China, October 30 – November 1, 2017, Proceedings
Yin, H.; Gao, Y.; Chen, S.; Wen, Y.; Cai, G.; Gu, T.; Du, J.; Tallón-Ballesteros, A.J.; Zhang, M. (Eds.)
2017, XVI, 609 p. 198 illus., Softcover
ISBN: 978-3-319-68934-0