Local-First Algorithms for Community Detection

Michele Amoretti, Alberto Ferrari, Paolo Fornacciari, Monica Mordonini, Francesco Rosi, and Michele Tomaiuolo

Università degli Studi di Parma, Dipartimento di Ingegneria dell'Informazione
Parco Area delle Scienze 181a, Parma 43124, Italy
michele.amoretti@unipr.it
aferrari@ce.unipr.it
paolo.fornacciari@unipr.it
monica.mordonini@unipr.it
rosi@ce.unipr.it
michele.tomaiuolo@unipr.it

Abstract. One of the most important problems in the field of social network analysis, and one of the most discussed ones, is community detection, aimed at clustering the nodes on the basis of their social relationships. Community detection is relevant in various fields, including: recommendation systems, link prediction and suggestion, epidemic spreading and information diffusion, sybil detection. In this paper, we discuss various ego-based community detection algorithms and propose a new one, named $PaNDEMON$, to exploit the parallelism of modern architectures. Comparing its performances with other algorithms, we show that $PaNDEMON$ demonstrates good scalability, while preserving the quality of results.

Keywords: Social Network Analysis, Ego-Based Community Detection, Parallel Algorithms

1 Introduction

Community detection is one of the most popular applications of social network analysis. A large number of algorithms and different approaches have been proposed and studied. In many algorithms, community detection is based on the concept of modularity. Social network analysis can be also applied at a local scale, as in the case of Ego-centered analysis. In this approach, the basic assumption is that, if two nodes are grouped together in the view of most of their neighbors, then they should be considered as actual members of the same community. Some algorithms use local scale analysis to infer communities also at a global scale.

As an interesting application, community detection can also be applied to the protection of social platforms, in particular those subject to so-called *sybil attacks*. In fact, existing algorithms for sybil defence are essentially community detection algorithms, distinguishing sybil nodes from legitimate ones. Apart from sybil detection, applications of community detection algorithms are many and

varied, including: recommendation systems, link prediction and suggestion, epidemic spreading and information diffusion.

This paper presents $PaNDEMON$ as a new parallel randomized algorithm for community detection and merging, derived from the $DEMON$ sequential randomized algorithm presented in [5]. The performance of these algorithms, both adopting a local-first approach, is compared in terms of computational complexity, community detection granularity, running time and result quality.

In particular, the rest of the paper is organized as follows. In Section 2, background information is provided. Namely, the problem of community detection in general, its application as a countermeasure against sybil attacks, and the importance of multi-agent systems in the field of social media are discussed. In Section 3, both $DEMON$ and $PaNDEMON$ are illustrated. In Section 4, their performance analysis is presented. Finally, concluding remarks and future research directions are provided in Section 5.

2 Background and Motivation

A large number of algorithms for community detection have been proposed and studied [8]. The traditional approach is to solve the problem of community detection at the global scale, based on analysis of the whole social graph. Many algorithms try to maximize modularity, a concept proposed by Newman [4, 14]. Other algorithms, such as Infomap [20] and Cross Associations [15], apply information theory to social graphs. Similarly to InfoMap, Walktrap [18] is also based on random walks. Label propagation [19] is another quite popular approach. In this approach, labels are spread through the edges of the graph. A label is attached to a node according to the majority of labels attached into its neighborhood, iteratively.

The idea of label propagation also paves the way for an alternative approach to community detection. In fact, global-scale algorithms for community detection may perform badly in the case of medium and large scale networks: since social graphs become too complex at the global level, it is very hard to produce a good analysis about the network topology. Consequently, these algorithms may produce low-quality results. Instead, social network analysis can be still applied at a local scale [5]. Intuitively, a social network user can quite easily distinguish some groups of friends, *e.g.*, schoolmates, colleagues, online acquaintances, family and other kinds of relatives. Quite obviously, the central user (*i.e.*, the ego node) participates himself in all these groups. Typically, a local community detection scheme works on the assumption that, if two nodes are grouped together in the view of most of their neighbors, then they should be considered actually members of the same community.

In Section 3, we analyze in detail two ego-based algorithms, namely $DEMON$ and $PaNDEMON$. In the following sections, instead, we discuss some possible application fields of such algorithms. Practical applications of community detection algorithms are many and varied. For example, they include: recommendation systems, for suggesting specific resources to potentially interested users; link pre-

diction and suggestion, for enriching a social graph with missing links; epidemic spreading and information diffusion, which can adapt the patterns of messaging to the actual network topology; detection of groups of outliers or malicious users, characterized by different metrics. Among these various applications, the study of community detection is also at the core of many sybil defense schemes, aimed at distinguishing sybil nodes from legitimate ones in distributed social platforms. The problem is also relevant for multi-agent systems, whose model is often applied to the case of distributed social platforms.

Sybil attacks. Especially when lacking a centralized authority, distributed social systems may be subject to so-called sybil attacks [7]. In this case, the attacker manages to create a large number of identities, using them to subvert the basic functionalities of the system. Completely distributed systems — in particular peer-to-peer (P2P) systems and Distributed Hash Tables (DHT) — often rely on redundancy mechanisms. These mechanisms become ineffective when an attacker controls a large number of identities in the system. Sybil identities — *i.e.*, scam nodes in the network — can collude and create an artificial majority, under an attacker's control.

In a number of research works, different schemes are proposed to detect and isolate sybil identities. They exploit properties of the social graph among nodes [10], if such graph exists or can be inferred. Social relationships among nodes are interpreted as trust bounds, which can replace the role of a central authority within the scope of sybil detection.

In fact, these defense mechanisms do not prevent an attacker from creating sybil identities. Instead, they rely on the fact that an attacker can forge a large number of identities, while he/she cannot easily create social links with non-sybil identities. Consequently, non-sybil nodes tend to share only few connections with sybil nodes, thus forming two quite distinct groups. These groups can be distinguished using various graph analysis techniques. In fact, social-based sybil defense schemes highlight the topological fracture of the social network, by exploiting the limited ability of attackers to create links with the legitimate nodes of the system.

Among distributed sybil defense schemes, random walks are a common mechanism for estimating the ranking of a given suspect node, starting from a trusted local node. The ranking is then compared with a threshold to decide whether the node is a sybil or not. One of the most known scheme based on random walks is SybilGuard [29]. In this scheme, each node is associated with a public/private keypair generated locally. A suspect node is marked as a sybil if random walks starting from it and from the trusted local source do not intersect. Otherwise, the node is considered trustworthy. SybilLimit [28] was developed as an improvement of SybilGuard. In this case, multiple independent random walks are performed from each node. Trustworthy nodes have to satisfy two conditions: (i) an intersection condition, *i.e.*, two random nodes of the trusted node and the suspect node must intersect in their last edge, and (ii) a balance condition, *i.e.*, a limit on the variation of counts associated with each random walk of the trusted node. SumUp [24] is essentially a sybil defense mechanism for securing peer-to-

peer voting. In principle, such a system could be subverted by sybil identities, which can also outnumber legitimate ones. Differently from other sybil defense systems, SumUp does not rely on random walks. Instead, it uses an adaptive vote flow technique to classify nodes. If the votes of a node are accepted, then the node is classified as trustworthy. Otherwise, it is classified as a sybil node.

In [25], authors argue that sybil defense schemes, in distinguishing sybil nodes from legitimate ones, solve in essence a community detection problem. In fact, such algorithms assign a rank to nodes in the local community around a trusted node, filtering out nodes with a low rank, acknowledged as sybil identities. If compared with a well known generic community detection algorithm, such as Mislove's algorithm [13], sybil defense schemes perform similarly, *i.e.*, they show a similar accuracy. In [25], the authors focus in particular on local community detection algorithms, which do not require a global knowledge of the social graph, but instead operate on a local view. Mislove's algorithm starts with a single node in a local community. Then, nodes in the neighborhood are added iteratively, if they raise the conductance metric.

Software Agents and Social Network Analysis. In the field of social media, multi-agent systems have been used as (i) an underlying layer or a middleware for developing social networking platforms, (ii) a technology to increase the autonomous and intelligent behaviour of existing systems and (iii) a tool to develop simulation environments for studying both online and offline human social networks.

For the first type of solution, an example is MAgNet [3], a multi-agent system built using JADE and FOAF. Poggi *et al.* [16] discuss some existing issues, mainly in terms of overlay infrastructure, management of the social graph, existence of specific ontologies. In particular, if global knowledge of the social graph is not centralized, community detection and social network analysis in general are particularly difficult.

In the second case, a number of research works propose multi-agent technology for augmenting existing social platforms [9, 11, 26, 27]. In particular, the model of multi-agent systems has been often used to study the problem of trust and reputation [2, 17, 21, 23], which is another aspect of social network analysis. In [6], authors extend the concept of Coalition Logic to deal with hidden coalitions among autonomous agents. This approach is orthogonal to specific sybil detection mechanisms, allowing to define various policies to block or defer individual actions bringing the system to a state identified in general as insecure.

Finally, multi-agent systems are a powerful tool for simulating the behaviour of social networks, either based on direct or online relationships [12, 22]. Ascape, NetLogo, MASON, Repast and Swarm are among the best known platforms for agent simulation, often used to study emerging behaviours and features of social networks.

3 Algorithms

3.1 Ego-based Community Detection

Regarding local community detection, we selected the $DEMON$ algorithm recently proposed by Coscia *et al.* [5], because of its highly appealing properties (correctness, completeness, determinacy, order insensitivity, compositionality and incrementality). $DEMON$'s pseudo-code is illustrated in Algorithm 1. Initially, the set of discovered communities is empty. For each node v, the $EgoMinusEgo$ function is applied, obtaining a graph e. Such a graph is then passed to the *LabelPropagation* function, which returns a set of v-related communities $\mathcal{C}(v)$. The union of every community $C \in \mathcal{C}(v)$ with v itself is then performed. At the end, set $\mathcal{C}$ is filled with communities.

Algorithm 1 Pseudo-code of $DEMON$'s core algorithm [5].

```
1: C ← ∅
2: for all v ∈ V do
3:     e ← EgoMinusEgo(v, G)
4:     C(v) ← LabelPropagation(e)
5:     for all C ∈ C(v) do
6:         C ← C ∪ {v}
7:     end for
8: end for
```

The time complexity of $DEMON$'s core algorithm is $O(nK^{3-\alpha})$, in case of scale free networks with n nodes, degree distribution $p_k = k^{-\alpha}$ and maximum node degree K [5]. After this phase, the $DEMON$ complete algorithm still requires to merge overlapping communities in $\mathcal{C}$. This task is performed by the $DEMON$'s *FlatOverlap* algorithm, described in Subsection 3.2.

Starting from the original code kindly provided by the authors, we designed and implemented a parallel version of the complete $DEMON$ algorithm, including both phases of label propagation and community merge. Our algorithm is called $PaNDEMON$ ($PArallel\ Non-deterministic\ DEMON$). In fact, it is derived from $DEMON$, but with significant differences in parallelism and non-determinism. In the core part of $PaNDEMON$, the **for all** $v \in V$ loop is split over n concurrent tasks. Thus, the theoretical speedup, with respect to the sequential version, is equal to the number of available processing units p.

3.2 Community Merging

The result of the community detection algorithm is a set of local communities $\mathcal{C}$, from the perspective of all network nodes. Such communities are potentially overlapping and do not represent the actual community coverage of the network. Further processing is needed, for merging the communities in $\mathcal{C}$. Coscia *et al.* [5] proposed a merging function, namely *FlatOverlap*, whose pseudo-code is

reported in Algorithm 2. In *FlatOverlap*, two communities are merged if and only if the smaller one is partially overlapping the largest one for a fraction at least equal to ϵ.

Algorithm 2 Pseudo-code of *DEMON*'s *FlatOverlap* function [5].

```
1: for all C ∈ 𝒞 do
2:     for all I ∈ 𝒞 do
3:         if C ⊆_ε I then
4:             𝒞 ← 𝒞 − {C}; 𝒞 ← 𝒞 − {I}
5:             U ← C ∪ I; 𝒞 ← 𝒞 ∪ {U}
6:         end if
7:     end for
8: end for
9: return 𝒞
```

Evaluating the time complexity of *DEMON*'s *FlatOverlap* is a challenging problem, as $|\mathcal{C}|$ changes over time. In particular, $|\mathcal{C}|$ reduces by 1 iff there is a pair (C, I), with $C, I \in \mathcal{C}$, such that $C \subseteq_\epsilon I$. The probability of such an event decreases over time, until $|\mathcal{C}|$ converges. How this happens, it does depend on network topology. We consider the worst case, in which the algorithm converges after n steps, each one being characterized by only 1 merge operation:

$$\begin{aligned} \#\text{ operations} &= \sum_{i=1}^{n} \frac{1}{2} \left(|\mathcal{C}| - i + 1\right) \left(|\mathcal{C}| - i\right) \\ &= \frac{1}{2} \left[\frac{n^3}{3} - |\mathcal{C}| n^2 + \left(|\mathcal{C}|^2 - \frac{1}{3} \right) n \right] \end{aligned}$$

As n depends on the network topology, it is impossible to find a general rule for the upper bound of the time complexity. If we assume that $n \sim |\mathcal{C}|^{1/2}$, then time complexity is $O(|\mathcal{C}|^{5/2})$.

To parallelize *DEMON*'s *FlatOverlap* is almost impossible, because of its incremental nature and the two nested **for all** loops for the comparison of every community pairs in $\mathcal{C}$. Thus, in *PaNDEMON* we designed and implemented an alternative function, denoted as *ParallelOverlap*, whose pseudo-code is illustrated in Algorithm 3. In *ParallelOverlap*, $\mathcal{C}$ is split into p subsets, where p is the number of available processing units. For each subset $\mathcal{C}_i$ ($i \in \{1, .., p\}$), its communities are picked one by one and placed into another set $\mathcal{L}_i$ (which is initially empty), either alone or merged with the "oldest" community $I \in \mathcal{L}_i$ such that $C \subseteq_\epsilon I$, where ϵ is the same parameter that characterizes *FlatOverlap*. Then, $\mathcal{L}_i$ becomes the new $\mathcal{C}_i$, for all $i \in \{1, .., p\}$. The union of all $\mathcal{C}_i$ results in a set $\mathcal{R}$. If the cardinalities of $\mathcal{R}$ and $\mathcal{C}$ are equal, then $\mathcal{C}$ is shuffled, split into p subset, etc. After k_{max} attempts, if the cardinalities of $\mathcal{R}$ and $\mathcal{C}$ are still equal, the algorithm stops and $\mathcal{C}$ is returned. Otherwise, when the cardinality of $\mathcal{R}$ is different (*i.e.*, lower) than $\mathcal{C}$'s one, $\mathcal{R}$ becomes the new $\mathcal{C}$, then $\mathcal{C}$ gets shuffled and

the process continues. Thus, in $PaNDEMON$, repeated randomized shuffling introduces a level of non-determinism over the original $DEMON$ algorithm.

Algorithm 3 Pseudo-code of $PaNDEMON$'s $ParallelOverlap$ function.

```
1: R ← ∅
2: l ← |C|; k_max ← log l; k ← 0
3: while l ≠ |R| or k < k_max do
4:     C → {C_1, .., C_p} s.t. ⋃_i C_i = C and ⋂_i C_i = ∅
5:     l ← |C|
6:     for all C_i ∈ {C_1, .., C_p} do
7:         L_i ← ∅
8:         for all C ∈ C_i do
9:             merged ← FALSE
10:            for all I ∈ L_i do
11:                if C ⊆_ε I then
12:                    U ← C ∪ I; L_i ← L_i − {I}; L_i ← L_i ∪ {U}
13:                    merged ← TRUE
14:                    break
15:                end if
16:            end for
17:            if merged = FALSE then
18:                L_i ← L_i ∪ {C}
19:            end if
20:        end for
21:        C_i ← L_i
22:    end for
23:    R ← ⋃_i C_i
24:    if |R| = l then
25:        shuffle(C); k ← k + 1
26:        continue
27:    else
28:        k ← 0; C ← R; shuffle(C)
29:    end if
30: end while
31: return C
```

To estimate the time complexity of $PaNDEMON$'s $ParallelOverlap$, we consider the worst case of convergence that is reached after n cycles, each cycle lasting $k_{max} - 1$ attempts to merge the $|\mathcal{C}| - (i-1)$ clusters, where $i \in \{1, .., n\}$, split over p processing units:

$$\begin{aligned} \#\text{ operations} &= k_{max} \sum_{i=1}^{n} \frac{1}{2} \left(\frac{|\mathcal{C}| - i + 1}{p} \right) \left(\frac{|\mathcal{C}| - i + 1}{p} - 1 \right) \\ &= \frac{k_{max}}{2p^2} \left[\frac{n^3}{3} + (p - 1 - 2|\mathcal{C}|) \frac{n^2}{2} + \left(\frac{1}{6} - \frac{p}{2} + (1-p)|\mathcal{C}| + |\mathcal{C}|^2 \right) n \right] \end{aligned}$$

As n depends on the network topology, it is impossible to find a general rule for the upper bound of the time complexity. If we assume that $n \sim |\mathcal{C}|^{1/2}$ and $k_{max} = \log|\mathcal{C}|$, then time complexity is $O(\log|\mathcal{C}|p^{-2}|\mathcal{C}|^{5/2})$, which is better than $FlatOverlap$'s when $p > \sqrt{\log|\mathcal{C}|}$.

4 Performance Evaluation

For evaluating the performance of the algorithms, we have adopted a test case derived from the IMDb online movie database. In particular, we have inferred a social network of actors, following the methodology described in [1], but using updated data as in [5]. The nodes of the graph are actors who starred in at least two movies, in the period 2001-2010. Other kinds of shows are not considered.

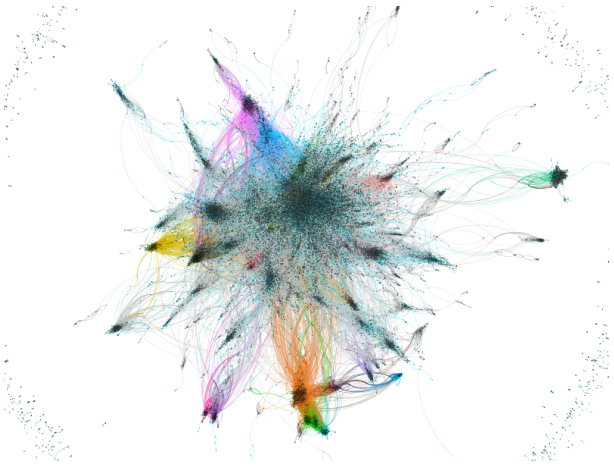

Fig. 1. IMDb social graph, with colors associated with the main communities found by our $PaNDEMON$ algorithm.

Consistently with [1], we have used an old definition of IMDb which considers the first 15 cast members of a movie as its stars; this distinction is not apperent anymore, neither in the IMDb website, nor in the available raw data. Actors are connected in the social graph if they starred together in at least two movies, in the reference period. Consequently, edges are undirected. As a result, we have obtained a social graph with $N = 58984$ nodes and 295099 edges, illustrated in Figure 1. The social graph is represented according to the ForceAtlas2 layout algorithm, and from a qualitative analysis it is apparent that many visible regions regroup actors with the same nationality. Colors are attributed to nodes according to the largest community they belong to, as detected by our $PaNDEMON$ algorithm.

Then, we have compared $DEMON$ and $PaNDEMON$, against the aforementioned IMDb social graph. All algorithms have been implemented in Python

(*DEMON* code is the original one [5], kindly provided by Coscia *et al.*) and executed on a Linux server equipped with Python 2.7, four Intel Xeon dual-core, 16 GB of RAM.

PaNDEMON has been executed using $p \in \{1, 2, 4, 8\}$ cores. Firstly, we have set $\epsilon = 0.75$ and tested different values of k_{max}. Then, we have chosen $k_{max} = \log N$ and tested different values of ϵ. Some results of the two test sets are reported in Figure 2, showing the number of detected communities, and Figure 3, showing the running time.

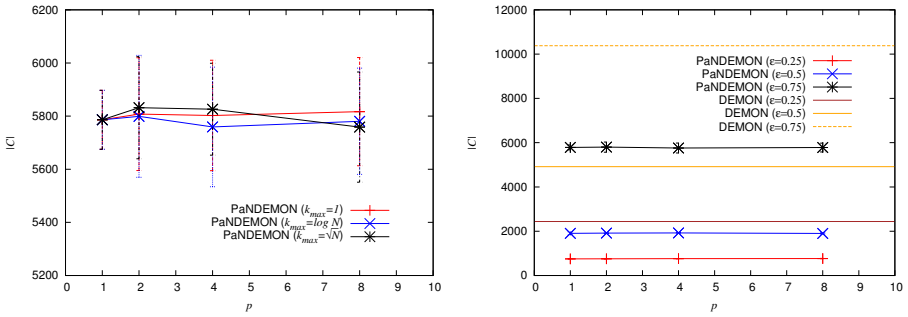

Fig. 2. Number of communities versus p, for increasing k_{max} (left) and ϵ (right). In the first plot, $\epsilon = 0.75$. In the second plot, $k_{max} = \log N$ is assumed.

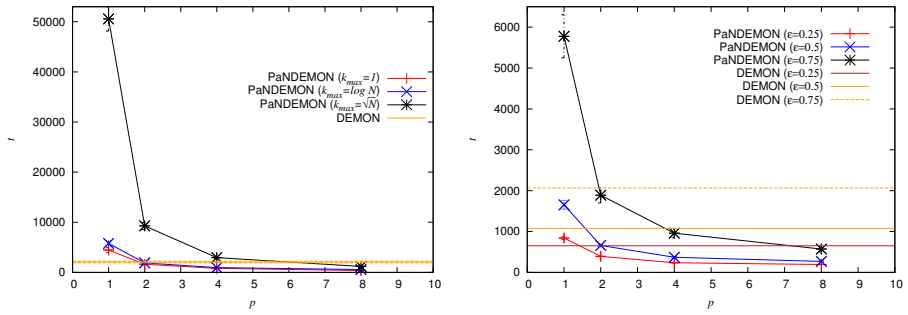

Fig. 3. Running time versus p, for increasing k_{max} (left) and ϵ (right). In the first plot, $\epsilon = 0.75$. In the second plot, $k_{max} = \log N$ is assumed.

It is evident that *PaNDEMON*'s *ParallelOverlap* merges communities much more than the original *DEMON*'s *FlatOverlap*, resulting in $|\mathcal{C}|_{FlatOverlap} \simeq 2|\mathcal{C}|_{ParallelOverlap}$, when $k_{max} > 1$. This result is due to the higher amount of merging attempts performed by *ParallelOverlap*. A less surprising result is that the higher k_{max}, the smaller the number of resulting communities, with

ParallelOverlap. Finally, the expected and most desired result is the one related to running time. Clearly, $PaNDEMON$ outperforms $DEMON$, when $p > 1$ and k_{max} is reasonably small ($k_{max} = \log N$ is a suitable choice).

Which clustering is the most realistic is a matter of further investigation, we will present in a future work. However, we can already state that $PaNDEMON$'s scalability does not affect the quality of the produced communities, with respect to $DEMON$ and to HLC (*Hierarchical Link Clustering*), this latter algorithm being the one that provides the highest quality results, in the context of overlapping community detection [1]. A first result has been obtained by training the Orange BRL multilabel classifier,[1] which we have chosen as a reference, on the different sets of communitites produced by the three algorithms under analysis. As its output, the classifier (which essentially is a set of binary Naive Bayes classifiers) has to predict the film genres and nationality of an actor, given his membership status in each community. The comparison shows no significant advantage of any one of the algorithms. However, this does not mean that detected communities are similar. On the contrary, also the cardinality of the set of detected communities is very different, as shown in table 1 (communities with fewer than 15 nodes are filtered out, in this kind of analysis). In our future investigation, we are going to compare the three sets of communities, *e.g.*, to see if they respect some sort of hierarchical containment. Moreover, we are going to experiment with additional social graphs of different nature, to have a more varies set of case studies.

Table 1. Quality of communities detected by different algorithms.

Algorithm	$\lvert\mathcal{C}_{15}\rvert$	**Accuracy**	**Precision**	**Recall**	**F-Measure**
HLC	793	0.298	0.549	0.362	0.436
DEMON	3820	0.275	0.440	0.410	0.424
PaNDEMON	1368	0.284	0.509	0.347	0.413

5 Conclusion

In this paper, we have introduced $PaNDEMON$, a parallel randomized algorithm for community detection in social graphs. Community detection is important for sybil detection, recommendation systems, link prediction and suggestion, epidemic spreading and information diffusion. $PaNDEMON$ was inspired by $DEMON$, a sequential randomized algorithm with highly appealing properties. Both $DEMON$ and $PaNDEMON$ are local-first algorithms, which originate at the local node, thus mimicking the flow of trust in a peer-to-peer

[1] http://orange.biolab.si/

network. Comparing the performance of the two algorithms, we have found that $PaNDEMON$ shows good scalability, without degrading the quality of results.

Future work will focus on further comparing sequential and parallel algorithms in terms of computational complexity and precision. Furthermore, we plan to study fully decentralized algorithms, where nodes collaborate to detect communities, using that knowledge to improve system resilience and user experience.

References

1. Ahn, Y.Y., Bagrow, J.P., Lehmann, S.: Link communities reveal multiscale complexity in networks. Nature 466(7307), 761–764 (2010)
2. Amoretti, M., Bisi, M., Laghi, M.C., Zanichelli, F., Conte, G.: Reputation management service for peer-to-peer enterprise architectures. In: International Conference on E-Business and Telecommunication Networks. pp. 52–63. Springer (2006)
3. Basuga, M., Belavic, R., Slipcevic, A., Podobnik, V., Petric, A., Lovrek, I.: The magnet: Agent-based middleware enabling social networking for mobile users. In: 10th International Conference on Telecommunications (ConTEL 2009). pp. 89–96. IEEE (2009)
4. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. Physical Review E 70(6), 066111 (2004)
5. Coscia, M., Rossetti, G., Giannotti, F., Pedreschi, D.: Uncovering hierarchical and overlapping communities with a local-first approach. ACM Transactions on Knowledge Discovery from Data (TKDD) 9(1), 6 (2014)
6. Cristani, M., Karafili, E., Vigano, L.: Blocking Underhand Attacks by Hidden Coalitions. In: 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011). pp. 311–320 (2011)
7. Douceur, J.R.: The sybil attack. In: International Workshop on Peer-to-Peer Systems. pp. 251–260. Springer (2002)
8. Fortunato, S.: Community detection in graphs. Physics Reports 486(3), 75–174 (2010)
9. Franchi, E., Poggi, A., Tomaiuolo, M.: Blogracy: A peer-to-peer social network. International Journal of Distributed Systems and Technologies (IJDST) 7(2), 37–56 (2016)
10. Franchi, E., Tomaiuolo, M.: Distributed Social Platforms for Confidentiality and Resilience. Social Network Engineering for Secure Web Data and Services p. 114 (2013)
11. Gursel, A., Sen, S.: Improving search in social networks by agent based mining. In: 21st international joint conference on Artifical intelligence (IJCAI'09). pp. 2034–2039 (2009)
12. Hamill, L., Gilbert, N.: Simulating large social networks in agent-based models: A social circle model. Emergence: Complexity and Organization 12(4), 78 (2010)
13. Mislove, A., Viswanath, B., Gummadi, K.P., Druschel, P.: You are who you know: inferring user profiles in online social networks. In: Third ACM international conference on Web search and data mining. pp. 251–260. ACM (2010)
14. Newman, M.E.: Modularity and community structure in networks. Proceedings of the National Academy of Sciences 103(23), 8577–8582 (2006)

15. Papadimitriou, S., Sun, J., Faloutsos, C., Philip, S.Y.: Hierarchical, parameter-free community discovery. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 170–187. Springer (2008)
16. Poggi, A., Tomaiuolo, M.: Integrating Peer-to-Peer and Multi-agent Technologies for the Realization of Content Sharing Applications. In: Information Retrieval and Mining in Distributed Environments, pp. 93–107. Springer (2010)
17. Poggi, A., Tomaiuolo, M., Vitaglione, G.: Security and trust in agent-oriented middleware. In: OTM Confederated International Conferences — On the Move to Meaningful Internet Systems. pp. 989–1003. Springer (2003)
18. Pons, P., Latapy, M.: Computing communities in large networks using random walks. Journal of Graph Algorithms and Applications 10(2), 191–218 (2006)
19. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical Review E 76(3), 036106 (2007)
20. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. Proceedings of the National Academy of Sciences 105(4), 1118–1123 (2008)
21. Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: First international joint conference on Autonomous agents and multiagent systems: part 1. pp. 475–482. ACM (2002)
22. Sebastio, S., Amoretti, M., Lafuente, A.L.: AVOCLOUDY: a simulator of volunteer clouds. Software: Practice and Experience 46(1), 3–30 (2016)
23. Tomaiuolo, M.: Trust management and delegation for the administration of web services. Organizational, Legal, and Technological Dimensions of Information System Administration pp. 18–37 (2014)
24. Tran, N., Min, B., Li, J., Subramanian, L.: Sybil-resilient online content voting. In: 6th USENIX Symposium on Networked Systems Design and Implementation. pp. 15–28. NSDI'09, USENIX Association, Berkeley, CA, USA (2009), http://dl.acm.org/citation.cfm?id=1558977.1558979
25. Viswanath, B., Post, A., Gummadi, K.P., Mislove, A.: An analysis of social network-based sybil defenses. ACM SIGCOMM Computer Communication Review 40(4), 363–374 (2010)
26. Walter, F.E., Battiston, S., Schweitzer, F.: A model of a trust-based recommendation system on a social network. Autonomous Agents and Multi-Agent Systems 16(1), 57–74 (2008)
27. Yu, B., Singh, M.P.: Searching social networks. In: Second international joint conference on Autonomous agents and multiagent systems. pp. 65–72. ACM (2003)
28. Yu, H., Gibbons, P.B., Kaminsky, M., Xiao, F.: Sybillimit: A near-optimal social network defense against sybil attacks. In: 2008 IEEE Symposium on Security and Privacy (SP 2008). pp. 3–17. IEEE (2008)
29. Yu, H., Kaminsky, M., Gibbons, P.B., Flaxman, A.D.: Sybilguard: defending against sybil attacks via social networks. IEEE/ACM Transactions on Networking 16(3), 576–589 (2008)