Locality Analysis Against Graph Partitioning

CHEN Hongxu
XU Zhengzi
NIE Xiao

October 23, 2015

1 Introduction

Social media has gained its popularity over the years. According to Pew research center[1], 74% of all the Internet users have used the social network sites; and the number keeps increasing.

Due to the need from the customers, in recent years, many startup companies begin to develop their business in this area. Based on the report of Gust[2], a company aims to build the connection between startup and investor, in 2015 second quarter, the number of startups has increased by 20% compared to data at the same time in 2014 and 26% for the first quarter of 2015. Among all the startups, 13% of them mainly focus on providing Internet web services and others, like entertainment applications startups may also include social networking functionalities.

One of the problems in developing software with social network capability is to divide the user data into groups and store them distributedly. A good way of storing the data will improve the availability by providing replication of the data into different server, and the scalability by maintaining the data locality within the server. Many researchers have developed various complexed partition algorithms to improve the performance of the large networking site. However, few of them have optimised their solution to suit the

[1] `http://www.pewinternet.org/data-trend/social-media/social-media-use-all-users`

[2] `https://gust.com/startup-trends`

case where a relatively small network is presented and little resource is available. Therefore, this study is going to evaluate a few well-known partition algorithms' performance for the small startup companies.

The rest of the report is organised in the following way: In section 2, a real world problem is defined with high motivation stated. The next section presents a detailed description of the methodology used in the study. Then the experiment setup and implementation are shown. The results and limitation of the experiment are discussed in section 4 and 5 respectively. The last section concludes this report by highlight the findings and insights.

2 Problem Description

2.1 Motivation

Nowadays, more and more small companies have developed software and mobile applications with networking services. Some of them provide networking function as their main business; while others use it as a method to increase the interaction among their users, and thus attracting more potential users of their products. Those small companies need to store their user profiles into the server. However, many startups may encounter problems where they may be lack of budget to deploy adequate number of servers and other infrastructures. Therefore, they must design their storage algorithm wisely in order to fully utilise their limited resource and achieve a relatively good performance.

One of the aspects which needs considering is to choose an efficient and effective graph partitioning algorithm to separate the user data and deploy them into different servers. It is important because a good deployment of user data at different server can speed up the query process when user would like to obtain it. Small companies need to provide better user experience in order to survive and thrive. Thus, providing user experience with a fast access rate can help the company to maintain and develop its user base.

2.2 Problem Definition

Our experiment aims to find a suitable graph partition solution for small business which runs networking services. More specifically, we are testing the performance (locality) of different partition algorithms under the same

condition, in which we treat any replication of data as overhead. The number of servers is set to a range of small values in order to accord with the real world companies with limited resources.

3 Methodology

3.1 Algorithms

In the existing studies of online social networks scalability, we found researchers have developed various partition algorithms to preserve the data locality while reducing the overhead. In this project, we decide to leverage the four graph partition algorithms, namely, random partitioning, METIS graph partitioning, community detection algorithm, and SPAR to test the partition performance in the case where different number of server is available.

The social network which needs to be partition can be represented by node and edges, where one node specifies one active user in the network, and the edge, which can be directed or indirected, stands for the existence of relationship between two connected nodes.

3.1.1 Random Partition

As the name suggests, random partition algorithm will cluster all the node in a random manner. The possibility for a user node to be stored in a particular server is equal to the possibility for it to be stored in another ones. Since there is no optimisation involved in the algorithm, we expected that it will have the highest overhead among all algorithm. Therefore, it can server as the baseline for our experiment.

Another reason for choosing this algorithm is that, since we are focusing on small business with relatively small number of server available, some of the advanced techniques may not function well. Using the random partition algorithm in this situation may reduce the the effort in implementation while achieves similar performance with those advanced algorithm. Even for some large corporation such as Facebook, random partition is widely used for simplicity.

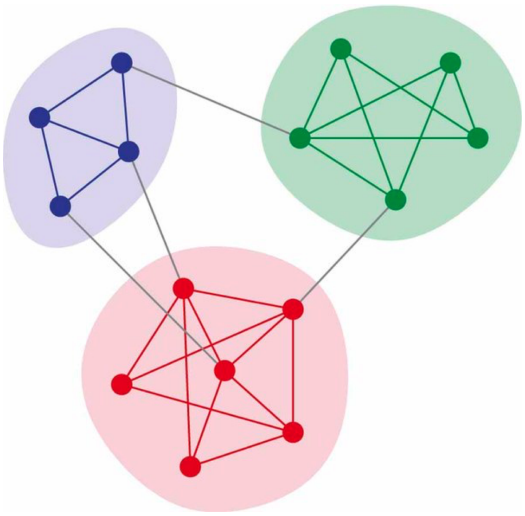


Figure 1: Community Structure Network

3.2 Community Detection

Community detection algorithm bases on the assumption that the input network involves the community structure [3] which can be detected and used as the criteria to perform the partition. The community structure is a network whose nodes can be easily divided into groups/communities, in which nodes are densely connected to each other. Whereas the inter-connection between nodes from different group is relatively sparse compared to the intra-connection of nodes within the group.

Figure 1 has depicted a typical community structure network with three groups. The community detection algorithm tries to identify the structural information in the network and leverage the group information to partition the node. More specifically, take Vincent's approach as an example [1], it first assigns all individual node to a different community. Then, it iteratively remove the node in its community and assigns it to other communities. It calculates the gain of all the assignment, then keeps the one with maximum gain and discards others. The iteration continues until no more assignment can be made; and the community partition reaches its local maximum. In the second phase, it build a new graph where the community of original graph become a node. The edge weight is the sum of the weight of the links between nodes in the corresponding two communities. Then it iteratively performs the algorithm to produce a hierarchical community structure in the graph.

The pseudocode for the algorithm is summarised in Algorithm 1.

Algorithm 1 Community Detection Algorithm: Vincent's Approach

```
1: procedure INIT
2:     for each Node i ∈ Graph do
3:         i ← community C_i
4:     end for
5: end procedure
6: procedure PHASE1
7:     loop:
8:     for each Node i ∈ Graph do
9:         Remove i from its Community
10:        for each Community c ∈ Graph do
11:            Add i to c
12:            Calculate and store gain
13:            Remove i in c
14:        end for
15:        gain ← Max(gain)
16:        i ← c with gain
17:    end for
18: end procedure
19: procedure PHASE2
20:    for each Community c ∈ Graph do
21:        Convert c to node
22:        edge weight ← Sum(number of connections)
23:    end for
24:    goto loop.
25: end procedure
```

There are various approaches to identify the communities in the network. Each of them has its own advantages in some circumstances. According to Andrea [2], among various community detection algorithm, Infomap [8] has the best performance over the others. In this experiment, we have chosen [1] to do the testing.

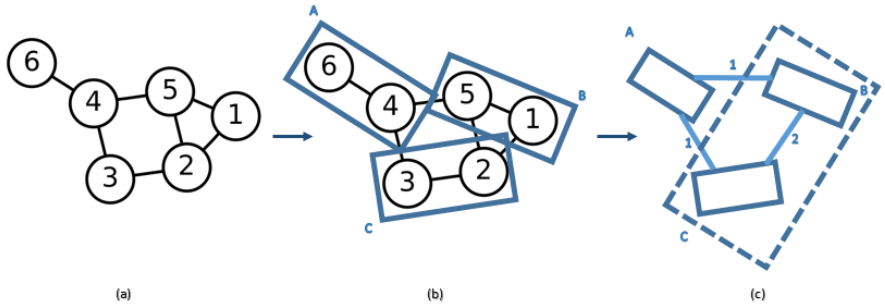


Figure 2: METIS Graph Reduce Algorithm

3.3 METIS

METIS is a multilevel graph partitioning algorithm [4], which consists several stages to partition a large graph. First, it tries to reduce the size of the group by collapsing the nodes and edges. Two similar nodes are grouped together and be represented by a coarser mesh. The coarser mesh then serves as a new large node with weighted edges specify the number of connection between the two meshes. After that, the process is repeated until the graph is small enough for later processing (usually less than 3k $\sim$ 5k nodes). Since the graph is small, METIS then performs a high-cost but high-quality spectral partitioning algorithm on the graph. After the partitioning, a refinement or un-coarsening phase is applied to the graph to retrieve the node back from the mesh recursively. By using the Kernighan-Lin heuristic algorithm [5], METIS can construct back the original graph with an accurate graph partitioning solution.

Figure 2 has given the basic steps of METIS. With the original graph shown in 2(a), METIS will group two nodes together to form a mesh. Figure 2(b) shows the grouped nodes represented by three meshes A,B, and C. Then, METIS treats the meshes as new nodes and repeats the process to group those large nodes. The criteria for group the node is based on the similarity of the two node. In this example, the similarity is proportion to the edge weight between two nodes. In Figure 2(c), a mesh, indicated in dash line, has formed, since the weight between mesh B and C is large than others. In this algorithm, each time the number of node/mesh is reduced by half. Unlike the community detection algorithm, which considers node context

globally, METIS only groups the node/mesh based on local information such as the connection weight between neighbour. Therefore, it can merge the graph much faster than the previous methods. After the graph is reduced to small size, METIS will partition the graph and reverse the procedure to get individual node with accurate partitioning assigned.

3.3.1 SPAR

SPAR (Social Partitioning and Replication) is a dynamic graph partitioning algorithm, which guarantees data locality by adding or removing nodes and edges at run time [7]. It makes a balance between the random partition algorithm which costs a lot when it reads the user profile and the full replication algorithm which require more writing operation to the server. SPAR only performs the replication/deletion when a change of the network takes place. In other words, SPAR regards the user's action as heuristics to help to build up the node partition layout in the server.

In our experiment, since we use only one replication for every data, whereas SPAR dynamically creates replicas for nodes neighbour in different server, we excluded this method in the experiment. However, the algorithm itself has a good performance and may become a choice for the small companies.

4 Implementation and Evaluation

4.1 Data description and Pre-processing

We built our experiment based on networkx-1.10[3], a well-known Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. During the experiment, we applied random partition algorithm (a naive implementation), community detection (provided by community module[4]) and metis partition (provided by nxmetis module[5]). We ran our experiment on a Ubuntu 15.04 workstation with 16G RAM and an Intel Xeon 12-core processor. The

[3] `http://networkx.github.io/`

[4] `http://perso.crans.org/aynaud/communities/`

[5] `http://networkx-metis.readthedocs.org/en/latest/`

Table 1: Basic Graph Information

	# node	# edge	clustering	# triangles	type
Facebook	4039	88234	0.6055	1612010	social
DBLP	317080	1049866	0.6324	2224385	ground-truth communities
YouTube	1134890	2987624	0.0808	3056386	ground-truth communities

Python version is 2.7.9. Our implementation code and the resultant data is available at GitHub[6].

The input data is collected from the SNAP page[7] provided by J. McAuley and J. Leskovec. [6]. We selected 3 datasets: Facebook, DBLP, and YouTube. All of our chosen raw graph data is represented as lines of edge pairs in a gzip file. In order to load the data fast, we did a pre-processing work that pickled each graph object in a data stream file.

We provided an elementary analysis (graph_info.py) on the graphs to get an overview of these data, including the node size, edge size, average clustering coefficient, number of triangles, etc. The results are shown in Table 1 and can be confirmed in their SNAP pages.

Additionally, we pre-computed the node degree list and sorted them in a descending order $D = \langle nd_1, nd_2, \ldots, nd_n \rangle$, where each element $nd_i, i \in \{1, 2, \ldots, n\}$ is a pair (n_i, d_i) which represents the node and its degree respectively.

4.2 Comparison of Partitioning Algorithms

This experiment compares the locality for different partition strategies without replication or temporary caching.

Our experiments are conducted with such an assumption: all nodes in the graph are accessed with equal opportunities, and accessing each node requires to retrieve *all* of the neighbor nodes $neighbor(n)$.

As we do not consider the data replication when a node data is stored remotely and do not care about the data fetch mechanism (e.g., caching frequently accessed data), the locality problem is only relevant to the static node

[6] `https://github.com/HongxuChen/ds_assign1`

[7] `https://snap.stanford.edu/data/`

distribution against a particular partitioning approach. A generated partitioning divides the underlying node set N into m parts: $\mathcal{N} = \{N_1, N_2, \ldots, N_m\}$. For each $n \in N_i, i \in \{1, 2, \ldots, m\}$, we compute the number of its neighbor nodes that are also in N_i, i.e.,

$$Local(n) = |neighbor(n) \cap N_i|, n \in N_i$$

And the graph locality is measured as the ratio of summation of node localities against the summation of all nodes' degrees.

$$\mathcal{L}(G) = \frac{\Sigma_{n \in N} Local(n)}{\Sigma_{n \in N} |neighbor(n)|}$$

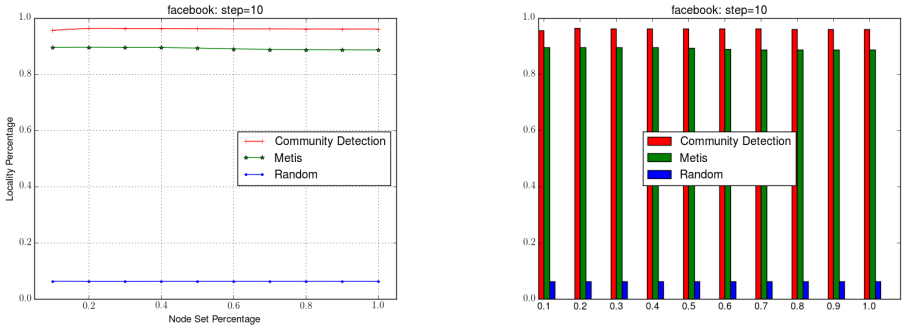
An interesting question is that: can we use a partial graph to approximate the result of the full graph locality information?

Intuitively, if we collect a node set whose element node has larger degree and get a subgraph G' from the edge relationship, we may get a locality that is similar to $\mathcal{L}(G)$. To get the node set, we slice the node order list D with a percentage thredshold $r \in (0, 1]$, that is to say $D_r = \langle nd_1, nd_2, \ldots, nd_\nabla \rangle$ is a prefix of D where $\nabla = r * |N|$. The node set N_r consists of all nodes corresponding to the first element of nd_i in each D_r, and G_r's edge set is defined as $\{e \in neighbor(n) | n \in N_r\}$.

Since community detection algorithm partitions the graph according to the graph structure, it does not require the number of group as the input. However for METIS and random partition need that. In this sense, we use the same part number parts as the former for the other two algorithms.

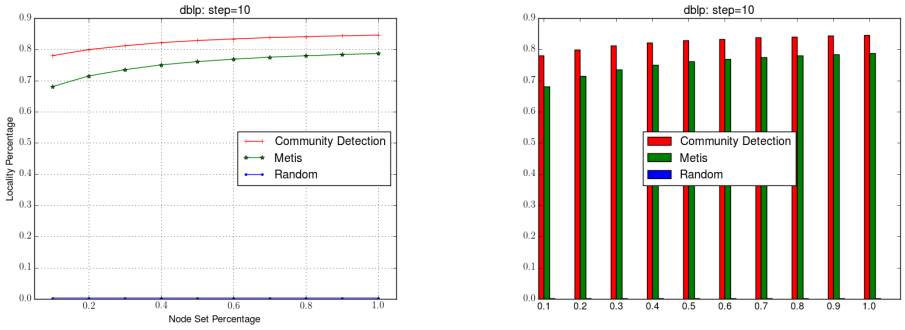
Figure 3 is the result for the Facebook dataset result, we assign r to be $\{0.1, 0.2, \ldots, 1.0\}$ (for YouTube it is $\{0.2, 0.4, \ldots, 1.0\}$ since otherwise it is quite costly) separately. Community Detection determines that parts $= 16$. It is obvious that Community Detection has the largest locality and the random partition approach has the lowest average locality. Similar results also occur for DBLP 4 and YouTube 5. By comprising Figure 3, 4 and 5, we can also discover that on a larger network such as the YouTube data, the locality of local clustering based algorithm METIS decreases heavily.

It is also interesting to notice that those nodes that have more neighbors have a greater impact on the locality: $\mathcal{L}(G_r)$ can somewhat represent the whole graph locality $\mathcal{L}(G)$ ($r = 1.0$).



(a) Line Chart (b) Bar Chart

Figure 3: Partitioning Approaches in Facebook



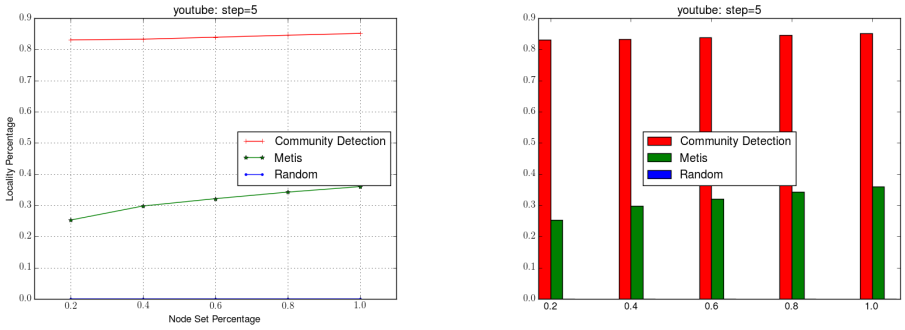
(a) Line Chart (b) Bar Chart

Figure 4: Partitioning Approaches in DBLP

4.3 Impact of Partition Number

It is widely accepted that partition is important for distributed networks. Surely, with more data distributed in different network nodes, data will be more likely to be stored remotely. However it remains an open question how partition number affects the locality in the absence of replication.

This experiment demonstrates that how the partition number parts affects the locality, without considering the workload of each network node (i.e., the partition group). We only use METIS and random partition in our experiment. We range parts in $\{2, 3, \ldots, 16\}$ (for YouTube it is $\{2, 3, \ldots, 16\}$) and experiment on each graph.



(a) Line Chart (b) Bar Chart

Figure 5: Partitioning Approaches in YouTube

As to the locality measurement, we use a similar way in Section 4.2 to compute different G_r (where r is set to be 0.2, 0.4, 0.6, 0.8, 1.0 separately). The final Locality $\mathcal{L}$ is computed as the average of the computed values.

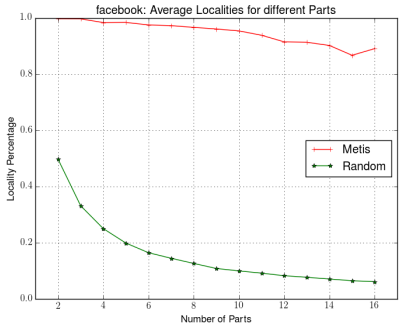
The result is depicted in Figure 6. It can be seen that the locality is decreased with the increase of partition groups. However the decrease of METIS is much slower than the random partition (for Facebook, DBLP and YouTube the lowest locality is still above 80%.

From the curve shown in Figure 6 and the results in Section 4.2, we may conjecture that the random partition approach is too coarse that the locality coefficient $\mathcal{L}$ is in proportion to $1/\mathsf{parts}$. In order to prove that, we redraw the plot where the $y-$axis to be the value of $\mathsf{parts} * \mathcal{L}$ (Figure 7).

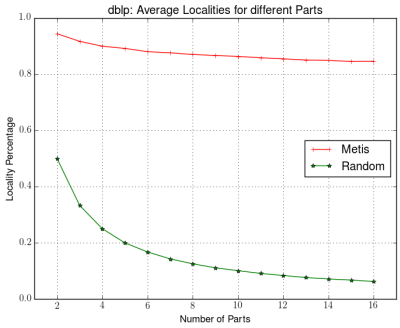
We found that the $y-$axis value of the newly generated curve is around 1.0 for our dataset. This convinced us that that the locality of random partition approach should suffer much since the value is in inverse proportion to the partition number parts.

4.4 Discussions

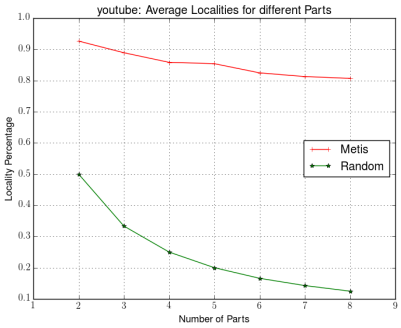
It is no wonder that community detection preserves the largest locality among the three applied algorithms since community detection exhaustively computes the connection gain throughout the full graph. Meanwhile, METIS also shows a relatively high locality. The time cost for community detection can be a big issue as the graph complexity increases. For example, it takes 1.46s for Facebook, while for DBLP and YouTube, the cost is 70.14s and 180.74s.



(a) Facebook



(b) DBLP

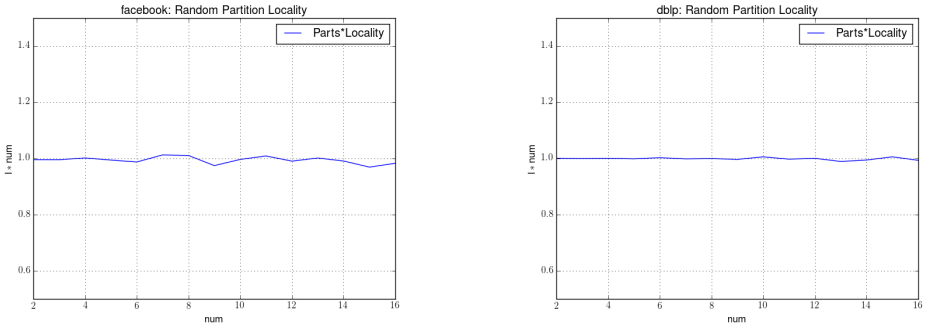


(c) YouTube

Figure 6: Partition Number's Effect on Locality

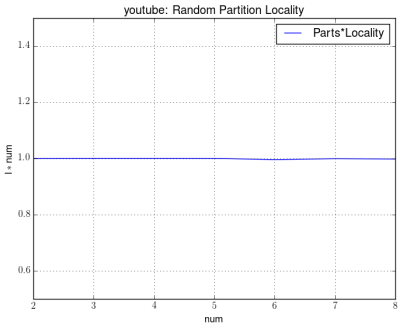
This indicates that community detection may be overkill to be applied for a static network. In our experiment, METIS leverages the locality and the time cost well.

It is worth noting that in reality the network may update frequently; this is especially true for social networks such as Facebook and Twitter. This means that partition algorithms such as community detection, METIS should be recomputed each time and the existing node should be rearranged. Therefore, random partitioning may still have the advantage to be widely used; in our experiment, even for YouTube, the partitioning is finished within 0.58s.



(a) Facebook

(b) DBLP



(c) YouTube

Figure 7: Random Partitions and Localities

5 Limitations

Our partition algorithm performance experiment bears several limitations.

First, the dataset used in the experiment is retrieved from the SNAP project page, which represents a small portion of the social network starting from one node and traverse through all of its neighbours. Even the combination of several networks cannot form a real social network. A real social network map may consist of many isolated nodes due to inactive accounts.

Second, unlike large networking sites, small companies' user social network may not exhibit some features such as the community structure among the users. An example is that a mobile online game application company may provide social networking services where the player can make friends in the game. However, the player-to-player interaction is limited to between

two players rather than in a group, so that one would like to make friends with someone he or she does not know and all his friends do not know each other. Therefore, the overall network map will be sparse and even distributed among all the nodes. In this case it is difficult to find a community structure in the networks. In our experiment, the dataset is retrieved from Facebook, DBLP, and YouTube, it may be favourable for some algorithms while make others in disadvantages; as a result, DBLP and YouTube data is labeled as network containing ground-truth communities.

Finally, our experiment only considers to store one copy of the user profile into the server. It is plausible since we are focusing on small business, where the availability of networking services may not be a key issue. However, in the real world, some of the companies would like to store the data in several copies. The difference in the number of replication will affect the performance results, for some of the overhead of an algorithm may be considered as a replication stored in a different server. Therefore, the overall overhead decreases when more replication are needed. Future works may take the replication into consideration and get a better performance comparison.

6 Conclusion

In this project, we examined the locality results of a few partitioning algorithms for an existing static network. The experiment has shown that community detection partition achieves the best locality while random partitioning approach has the least computation overhead. If the underlying network is infrequently changed or its size is relatively small, we suggest implementing the METIS algorithm since it leverages the computation and the data locality well. For some companies whose resource is limited, they can opt to choose to combine the random partitioning approach with replication, or some caching mechanisms. Our work also provides a reference for these companies to evaluate the partitioning approaches by themselves.

References

[1] V. Blondel, J. Guillaume, R. Lambiotte, and E. Mech. Fast unfolding of communities in large networks. *J. Stat. Mech*, page P10008, 2008.

[2] S. Fortunato and A. Lancichinetti. Community detection algorithms: a comparative analysis: invited presentation, extended abstract. In *4th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '09, Pisa, Italy, October 20-22, 2009*, page 27, 2009.

[3] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[4] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

[5] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell Systems Technical Journal*, 49(2), 1970.

[6] J. Mcauley and J. Leskovec. Discovering social circles in ego networks. *ACM Trans. Knowl. Discov. Data*, 8(1):4:1–4:28, Feb. 2014.

[7] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine(s) that could: Scaling online social networks. *IEEE/ACM Trans. Netw.*, 20(4):1162–1175, 2012.

[8] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.