Distributed Community Detection with the WCC Metric

Matthew Saltz
DAMA-UPC
Universitat Politécnica de Catalunya
msaltz@ac.upc.edu

Arnau Prat-Pérez
DAMA-UPC
Universitat Politécnica de Catalunya
aprat@ac.upc.edu

David Dominguez-Sal
DAMA-UPC
Universitat Politécnica de Catalunya
ddomings@ac.upc.edu

ABSTRACT

Community detection has become an extremely active area of research in recent years, with researchers proposing various new metrics and algorithms to address the problem. Recently, the Weighted Community Clustering (WCC) metric was proposed as a novel way to judge the quality of a community partitioning based on the distribution of triangles in the graph, and was demonstrated to yield superior results over other commonly used metrics like modularity. The same authors later presented a parallel algorithm for optimizing WCC on large graphs. In this paper, we propose a new distributed, vertex-centric algorithm for community detection using the WCC metric. Results are presented that demonstrate the algorithm's performance and scalability on up to 32 worker machines and real graphs of up to 1.8 billion edges. The algorithm scales best with the largest graphs, finishing in just over an hour for the largest graph, and to our knowledge, it is the first distributed algorithm for optimizing the WCC metric.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*clustering*

Keywords

Community detection; Distributed graph algorithms

1. INTRODUCTION

Due to the generality of the graph as a data structure, graphs correspond well to many different systems in the real world, like social networks, molecules, road maps, and more; and many problems can be expressed intuitively and solved using a graph representation. One such problem whose solution has many applications is that of *community detection* – automatically identifying groups of vertices that are tightly connected among themselves and loosely connected with the rest of the graph. In social networks, for example, the identification of communities can help with targeted marketing; or in a network of items that are frequently purchased together, community detection could be used to make recommendations.

As the graphs being operated on become larger and larger, the ability to process them in memory on one machine becomes infeasible due to both time and memory constraints. For these two reasons, complexity and size, distributed algorithms have become necessary to solve problems on large graphs. In this paper, we present a distributed algorithm for optimizing WCC [17], a recently proposed metric for judging the quality of community partitionings. The algorithm scales well on real graphs of up to 1.8 billion edges and outperforms a parallel, centralized algorithm that also seeks to optimize WCC [18]. The algorithm follows the vertex-centric paradigm introduced by the Pregel platform [12], and to the best of our knowledge, it is the first distributed algorithm for optimizing the WCC metric.

The structure of the paper is as follows. In Section 2, we begin by presenting an overview of related work in community detection and distributed community detection. Next, in Section 3, we introduce background material and the terminology used in the rest of the paper. Following this, the proposed algorithm is explained in Section 4, followed by experimentation in Section 5. We conclude with a discussion of future work.

2. RELATED WORK

Most of the research on community detection algorithms has focused on single threaded algorithms on SMP machines. The list of proposals is rich and diverse, with those based on modularity maximization forming the most prominent family of community detection algorithms [15]. Modularity is a community detection metric that rewards those partitions with communities with an internal edge density larger than that expected in a null model. Several strategies have been proposed for its optimization, such as agglomerative greedy [5] or simulated annealing [13]. One of the most famous and widely used community detection algorithms based on modularity maximization is the *Louvain* method [4], a multilevel approach that scales to graphs with hundreds of millions of objects. However, the quality of its results decreases considerably as the size of the graph increases [7]. More importantly, it has been reported that modularity has resolution limits [3, 6], which means that modularity is unable to detect small and well-defined communities when the graph is large. Related to this, recent studies have proven not only that modularity has detectability issues [14] (i.e. it is not able to identify communities even if they are well defined), but also that the identification of well-defined communities is more difficult than ill-defined ones [19]. Although it has not been studied whether or not WCC also suffers from these problems, properties presented in [17] suggest that algorithms based on WCC are able

WWW'15 Companion May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2744715.

to deliver cohesive and structured communities regardless of the size of the graph.

There also exist several proposals based on random walks. The intuition is that in a random walk, the probability of remaining inside of a community is higher than going outside, due to the higher density of internal edges. This strategy is the main idea exploited in Walktrap [16]. Another algorithm based on random walks that is highly adopted in the literature is Infomap [22], which searches for a codification for describing random walks based on communities. The codification that requires the least amount of memory (attains the highest compression rates) is selected. According to the comparison performed by Lancichinetti et al. [7], Infomap stands as one of the best community detection algorithms in the literature.

Another category of algorithms is formed by those capable of finding overlapping communities, which have gained significant interest during the last years. We find several proposals, such as Oslom [8], which uses the *significance* as a fitness measure in order to assess the quality of a community. Similar to modularity, the significance is defined as the probability of finding a given cluster in a random null model. Another algorithm that falls into this category is the Link Clustering Algorithm (LCA) [1]. This algorithm is based on the idea of taking edges instead of vertices to form a community. The similarity of adjacent vertices is assessed by looking at the Jaccard coefficient of the adjacency lists of the two vertices of the edges. Those edges connecting vertices with high similarity are assigned to the same community, and so overlapping communities emerge naturally. Finally, a recently proposed algorithm is BigClam by Yang et al. [25]. This algorithm is based on computing an affiliation of vertices to communities that maximizes an objective function using non-negative matrix factorization. The objective function is based on the intuition that the probability of an edge existing between two vertices increases with the number of communities the vertices share (i.e. the number of communities in which the vertices overlap).

Most of the work regarding the exploitation of parallelism for community detection has the form of multithreaded algorithms for SMP machines. In [11], authors propose a parallel version of the *Louvain* method, which achieves an speedup of 16x using 32 threads. Similarly, in [21] Riedy et al. propose an agglomerative modularity optimization algorithm for the Cray XMT and Intel based machines, capable of analyzing a graph with 100 million nodes and 3.3 billion edges in 500 seconds. Finally, in [2] the authors propose a parallel version of Infomap, called RelaxMap that relaxes concurrency assumptions of the original method, achieving a parallel efficiency of about 70%.

There has been little work regarding distributed algorithms for community detection. One family of algorithms that fit well into the vertex-centric model are those based on label propagation [20, 23]. In label propagation, each vertex is initialized with a unique label, and then, they define rules that simulate the spread of these labels in the network similarly to infections. Label propagation has the advantage of being asymptotically efficient, but no theoretical guarantees are given regarding the quality of the results, especially in networks where communities are not well-defined.

3. BACKGROUND & TERMINOLOGY

Informally stated, the goal of community detection is, given a graph, to divide the graph into groups (communities) of vertices such that, within a group, vertices are tightly connected, and between groups, there are few connections. For non-overlapping community detection, which is the focus of this paper, no two communities contain the same vertex. There are two primary aspects of this problem. First, it is necessary to give a formal definition of a metric that defines the quality of a given grouping, or *partitioning*, of a graph. The next step is to create an algorithm to find one or more partitionings of the graph that optimize this metric.

In this paper, we address the second part of this problem by proposing a scalable, distributed algorithm for the optimization of the WCC metric proposed in [17]. This metric is defined on unweighted, undirected graphs. Inspired by properties of real-life networks, the basic idea behind the metric is that within a community, vertices should have a high concentration of triangles among themselves, and they should close more triangles with other vertices in the community than with vertices outside of the community. Using this idea, given an undirected, unweighted graph $G(V, E)$, the quality of a community may then be defined as the average cohesion of each of its member vertices to the other vertices in the community, where the cohesion of a vertex x to a set of vertices S is defined as

$$WCC(x,S) = \begin{cases} \frac{t(x,S)}{t(x,V)} \cdot \frac{vt(x,V)}{|S\setminus\{x\}|+vt(x,V\setminus S)} & \text{if } t(x,V) \neq 0 \\ 0 & \text{if } t(x,V) = 0 \end{cases}$$

The function $t(x, S)$ here gives the number of triangles closed by x with other vertices in S, and the function $vt(x, S)$ gives the number of unique vertices contained in all such triangles. This cohesion metric therefore rewards a high ratio of triangles closed within the community versus triangles closed outside of the community (the left-hand term) and punishes vertices that have a high number of vertices in its community with which it does not close any triangle (the right-hand term). In other words, the left term promotes that the communities are well defined and isolated from the rest of the graph; and the right term promotes that all nodes in the community are interconnected and form triangles.

The quality of a partitioning is the average quality of each vertex in its assigned community. So, for a set S, $WCC(S)$ is defined as the average $\forall x \in S$ of $WCC(x, S)$, and the final WCC of a partitioning $\mathcal{P} = \{C_1, \ldots, C_n\}$ of V is then defined as

$$WCC(\mathcal{P}) = \frac{1}{|V|} \sum_{S\in\mathcal{P}} \sum_{x\in S} WCC(x,S).$$

In practice, the optimization of this metric results in high quality partitionings that correspond well to ground-truth communities, and it satisfies a number of desirable theoretical properties. For more information on the metric itself see [17].

4. PRESENTATION OF ALGORITHM

Our algorithm for optimizing this metric consists of three basic phases: preprocessing, community initialization, and WCC iteration. In the first phase, the values of $t(x, V)$ and $vt(x, V)$ are computed for every vertex, and all edges that do not belong to any triangles are removed from the graph. Next, the local clustering coefficient of each vertex is computed, and an initial partitioning of the graph is determined based on these coefficients. From this initial partitioning, the WCC iteration process is repeatedly applied, where each vertex chooses a new community simultaneously based on a heuristic and the global WCC value is computed. The algorithm halts when the WCC value converges. An overview of the algorithm can be seen in Figure 1.

4.1 Preprocessing

The preprocessing portion of the algorithm is responsible for two things: counting, for each vertex, the total of number of triangles it belongs to in the graph ($t(x, V)$), and removing all edges which do not belong to any triangles. After removing all such edges, $vt(x, V)$ is simply the degree of the vertex x. This filtering step

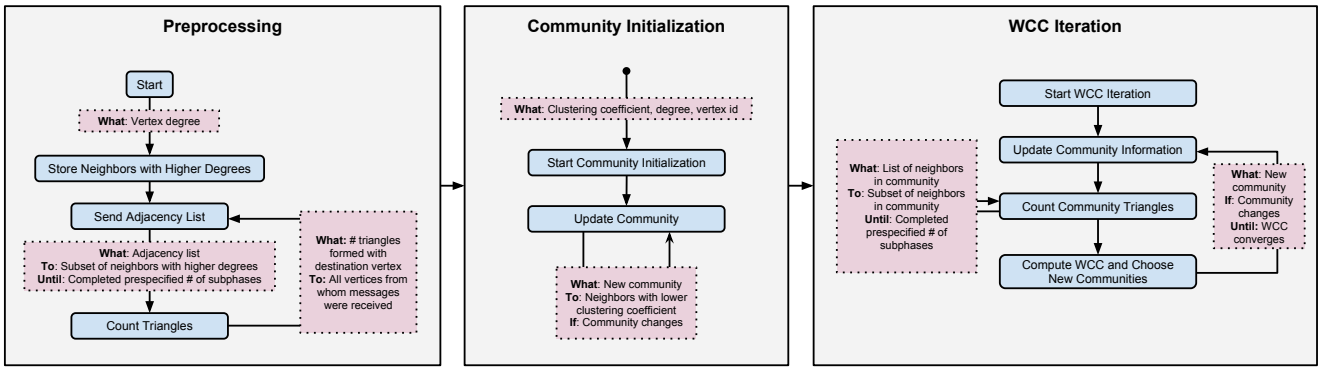

Figure 1: Algorithm Overview. The boxes with dotted edges represent messages sent by vertices. Where unspecified, assume that a message is sent to all neighbors of the vertex.

improves performance and allows simplifying assumptions later when deciding whether to transfer a vertex from one community to another. Note that these two values are constant throughout computation and therefore only need to be calculated once.

Given two vertices u and v, a standard way to compute the number of triangles they form together (the number of triangles in which the edge (u, v) is included) is to intersect their adjacency lists in order to count the number of their common neighbors. If the two vertices have no common neighbors, the edge (u, v) is removed from the graph, because it does not affect the computation of WCC. To count all of the triangles in the graph in which node u is contained, one must do this process for every neighbor v of u. In a centralized setting, this is relatively straightforward to implement. However, in a vertex-centric distributed setting, vertices do not have access to the adjacency lists of their neighbors, and therefore adjacency lists must be sent between vertices via message passing. With a large graph, if every vertex sends its adjacency list to every one of its neighbors in one superstep, this may lead to an excessive amount of time being spent in communication, or in the worst case, to memory problems that cause worker failures.

In order to address this problem, we propose two optimizations. First, we observe that in real life graphs, there tend to be a few 'hub' vertices with a very high degree and many vertices with a much lower degree [9]. This means that when these hub vertices send out their adjacency sets, it incurs a high communication cost in comparison with the messages sent by non-hub vertices. For this reason, in the first superstep, each vertex sends its degree to all of its neighbors, and following this, vertices only send their adjacency sets to neighbor vertices with a higher degree. The higher degree vertex in an edge then counts the triangles formed with the lower degree vertex, and responds with a message containing the triangle count.

Secondly, to reduce the occurrence of memory problems, this phase may be split into several subphases, where each vertex only sends its adjacency list to a subset of its neighbors in each subphase. The number of subphases is chosen by counting the total number of vertices that will be sent in messages during preprocessing and using this to estimate the approximate overhead required to send these messages, yielding the model

$$nPrepPhases = \left\lceil \frac{vertexSize \cdot \sum_{v \in V} |adj(v)||hdn(v)|}{nWorkers \cdot availWorkerMemory} \right\rceil,$$

where $adj(v)$ is the adjacency set of vertex v (the contents of a preprocessing message), $hdn(v)$ is the set of neighbors of v that have a higher degree than it (the destinations of the message), and $vertexSize$ is an estimate of the amount of memory taken to send one vertex id. For a given vertex v, $|adj(v)||hdn(v)|$ gives the total number of elements (vertex ids) that will be sent during preprocessing. The numerator therefore estimates the total amount of memory that will be taken by all messages sent across all preprocessing phases. This sum is computed with aggregators just after the computation of hdn for each vertex. The denominator estimates the total amount of memory available for preprocessing overhead in the cluster, assuming an even degree distribution across workers. The value for $availWorkerMemory$ is chosen based on the resources available for preprocessing, and the number of preprocessing phases is thus chosen such that each subphase operates with an overhead less than this value.

Together, these two optimizations together greatly reduce the cost of communication during preprocessing.

4.2 Community Initialization

Following preprocessing, the graph consists only of edges that are part of at least one triangle. The next step is to create an initial partitioning of the graph from which to begin the process of WCC optimization, meaning that each vertex must decide its initial community. We make the assumption that the higher the clustering coefficient of a vertex, the more likely its neighbors are to belong to its community, because a high clustering coefficient indicates that these vertices are tightly connected. This assumption is also applied in [18], but the computation method presented there is not adapted to the vertex centric processing model.

We require that the initial communities fulfill the following properties:

1. Every community contains a single center vertex and a set of border vertices connected to the center vertex.
2. The center vertex has the highest clustering coefficient of any vertex in the community.
3. Given a center vertex y and a border vertex x in a community, the clustering coefficient of y must be higher than the clustering coefficient of any neighbor z of x that is the center of its own community.

The process for obtaining such initial communities is shown in Figure 2. First, each vertex sends a message with its id, its clustering coefficient, its degree[1], and its initial community (its vertex id) to

[1]The degree is only used in the case that two neighbors of a vertex have identical clustering coefficients. In this case, the vertex with

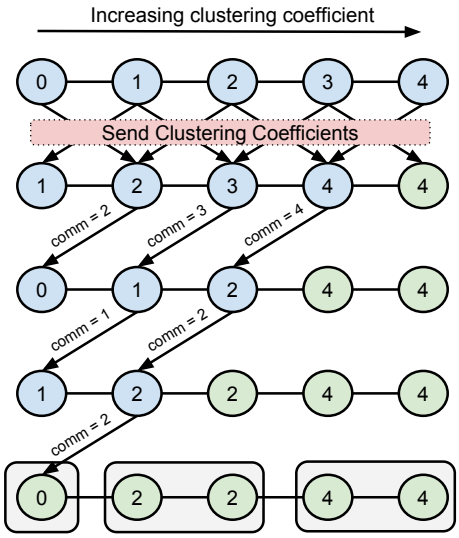

Figure 2: Community Initialization Process. The number inside each vertex indicates its current community, and arrows represent messages being passed from one vertex to another. Execution proceeds downward in the diagram, with each level being the resulting configuration after the incoming messages are processed. When a vertex receives a message from a vertex with id x and community y, it checks whether x is currently a center vertex by checking if $x = y$ and whether the clustering coefficient of x is greater than that of its current center and changes its community accordingly.

all of its neighbors. Each vertex then saves its incoming messages for use in future steps. Following this step, a vertex chooses its new community to be the id of the neighbor who has the highest clustering coefficient, considering as candidates only the neighbors that are currently centers. If its own clustering coefficient is higher then that of any neighbor or if none of its neighbors are currently centers, it chooses to be the center of its own community.

In the example in the figure, this means that after the communication of clustering coefficients, each vertex chooses the id of the vertex to its right as its community. However, after this step, the third desired property above is violated; the first three nodes belong to the communities of the vertices to their right, none of which are center nodes. So, it is then necessary for any vertex x that has changed communities to communicate its new community to all of its neighbors with lower clustering coefficients. These neighbors are the only ones that need to be notified because only vertices with a lower coefficient can become border nodes of x. After receiving the new communities of their neighbors, vertices redetermine their communities based on which neighbors have become borders and centers in the previous step. This iterative process continues until no vertices change communities, in which case all three properties above are satisfied.

the higher degree is considered to be 'higher'. If the degrees are also equal, the vertex with the higher id is considered to be higher.

4.3 WCC Iteration

The main idea behind WCC iteration is to have each vertex repeatedly update its community based on an improvement heuristic and to evaluate the overall WCC between each update, and after a prespecified number of steps where the WCC does not improve more than a certain amount, the computation halts.

4.3.1 Choosing a new community

When updating its community, the vertex has three options:

- **Transfer:** The vertex moves from its community to the community of a neighboring vertex.
- **Remove:** The vertex removes itself from its current community and becomes the sole member of its own isolated community.
- **Stay:** The vertex remains in its current community.

In order to choose which of these actions to perform, the vertex must decide which of the actions will most likely lead to the biggest improvement in the global WCC value. In [18], the authors present a heuristic for the WCC improvement induced by each action, using aggregate community statistics for a vertex's current and neighbor communities (the size and edge density of the community and the number of edges leaving the community), the graph's clustering coefficient, and a vertex's knowledge of its neighbors' communities. The heuristic is an approximation of the WCC that does not require the computation of the internal triangles, and thus is computationally more efficient. Due to its effectiveness, we use this heuristic as well. More details on the heuristic can be found in [18]. Because this update process occurs independently within each vertex, every vertex may perform the update simultaneously, meaning that this portion of the algorithm very effectively exploits parallelism.

4.3.2 WCC Computation

To compute the actual global WCC, the values $t(x, C_x)$ and $vt(x, C_x)$ must be calculated for each vertex x and its community C_x. This follows the same distributed triangle-counting process as in preprocessing, except that messages are only sent between vertices in the same community, and thus this step is less computationally expensive than global triangle counting. These local WCC values are then aggregated and averaged to obtain the global WCC. If a new best WCC has been obtained, vertices save their current communities, and when the WCC value converges, vertices output their saved community that led to the best overall WCC.

5. EXPERIMENTATION

For experimentation, we chose to perform tests on a variety of real life graphs, taken from the SNAP graph repository[2]. Information about each graph can be found in Table 1. Experiments were performed on a 40 node cluster with 2.40GHz Xeon E5-2630L processors and 128G of RAM each, and a 1 Gigabit Ethernet connection between nodes. In terms of software, we use Giraph release 1.1.0 and Hadoop version 0.20.203 (the default for Giraph). All experiments were repeated 5 times and their results were averaged to obtain the numbers shown. [3].

The goal of the experiments is to demonstrate the scalability of the method as the number of workers grows, as well as to compare performance benefits and result quality as compared to the parallel

[2] `http://snap.stanford.edu`

[3] Code for the distributed WCC algorithm can be found at `http://www.github.com/saltzm/distributed_wcc`

	Vertices	Edges	Communities
Youtube	1,134,890	2,987,624	8,385
LiveJournal	3,997,962	34,681,189	287,512
Orkut	3,072,441	117,185,083	6,288,363
Friendster	65,608,366	1,806,067,135	957,154

Table 1: Characteristics of the test graphs

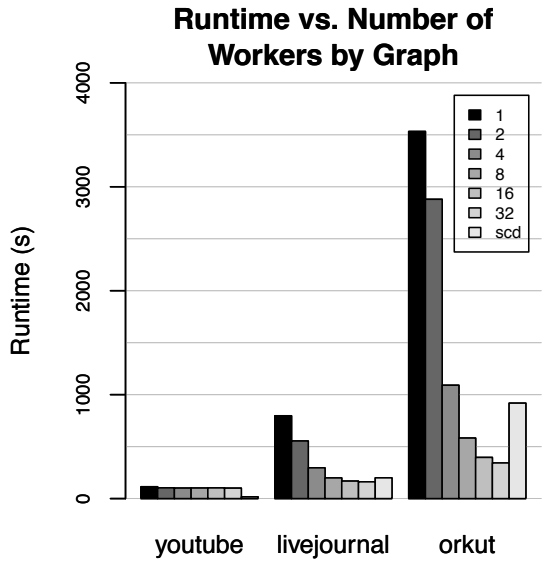

Figure 3: The runtime of the algorithm on each graph, varying the number of workers. The last bar in each group indicates the runtime of the centralized Scalable Community Detection algorithm from [18].

centralized version reported in [18]. In Figure 3, we see that in all cases except for the smallest graph, our distributed version eventually outperforms the centralized version. The final WCC values obtained by both methods (not shown) are very similar as well, within 1% of each other for every graph, indicating that there is no decrease in result quality incurred by the distributed algorithm. In addition, from looking at Figures 4 and 6, the speedup of the algorithm with the addition of workers improves with the size of the graph; the larger the graph, the better the scalability. For the largest graph, Friendster, we were not able to run the algorithm on fewer than 24 machines due to memory issues. This could be ameliorated by using a more efficient data structure for storing the graph, since the graph alone used a large amount of memory, which could be a topic of future work. In the largest graphs, we measured that the two main bottlenecks are triangle counting during preprocessing and in the computation of the next best community for each vertex. Because the phase for choosing new communities is much more computation heavy than communication heavy, it is to be expected that additional parallelism would continue to boost performance especially in this phase as the number of workers increases.

6. CONCLUSION & FUTURE WORK

In this paper, we presented a scalable algorithm for distributed community detection using the WCC metric that performs well on graphs of over one billion edges. In particular, in the Friendster

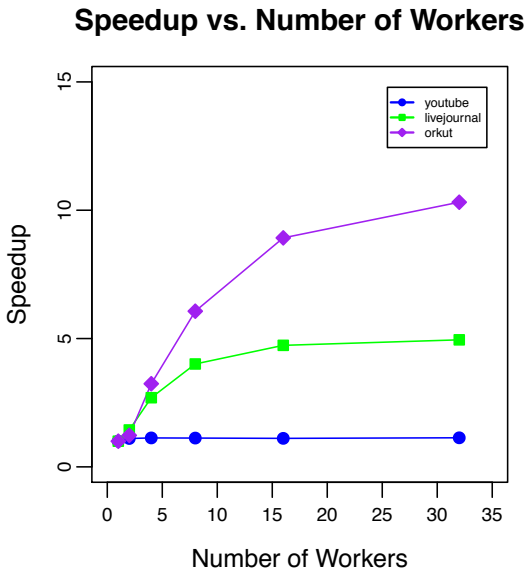

Figure 4: The speedup of the algorithm on each graph as compared to the execution time with 1 worker, varying the number of workers. Note that speedup improves as the size of the graph increases.

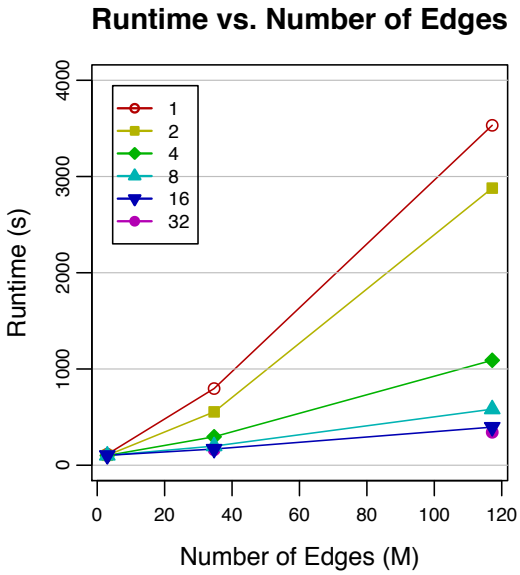

Figure 5: The runtime of the algorithm increases with the number of edges in the graph, varying the number of workers. When the number of workers is higher, the runtime increases more slowly as edges are added.

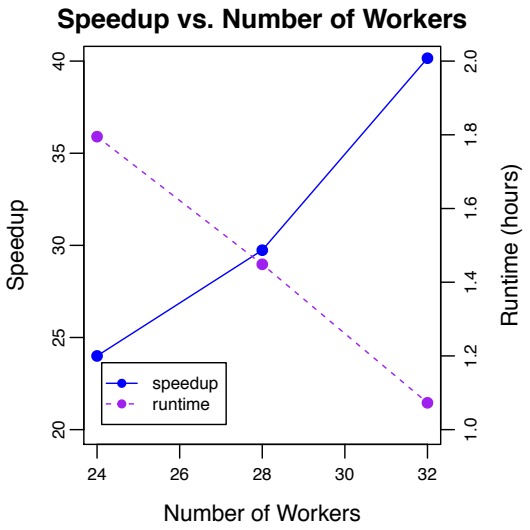

Figure 6: The speedup of the Friendster graph, varying the number of workers. Because the smallest number of machines used was 24, the runtime with one worker is extrapolated to be 24 times the runtime with 24 workers, and the speedups for the rest are calculated from that value. For comparison, the average runtime of the centralized Scalable Community Detection code was 5.56 hours.

graph, with over 1.8 billion edges, the algorithm scales well from 24 to 32 workers and at its fastest finds all communities in just over an hour. Current bottlenecks include memory requirements for triangle counting and runtime for the computation of new communities for each vertex. In the future, we may consider using alternative triangle counting algorithms, including ones that only approximate the number of triangles. Furthermore, implementing the algorithm in alternate frameworks such as GraphX [24] and GraphLab [10] would be worthwhile, as well as comparing to other community detection algorithms like label propagation.

7. ACKNOWLEDGMENTS

The members of DAMA-UPC thank the Ministry of Science and Innovation of Spain and Generalitat de Catalunya, for grant numbers TIN2013-47008-R and GRC-2014-890 respectively. Arnau Prat-Perez thanks the EU FP7 project LDBC (FP7- ICT2011-8-317548) for funding the LDBC project.

8. REFERENCES

[1] Y. Ahn, J. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

[2] S.-H. Bae, D. Halperin, J. West, M. Rosvall, and B. Howe. Scalable flow-based community detection for large-scale network analysis. In *Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on*, pages 303–310. IEEE, 2013.

[3] J. P. Bagrow. Are communities just bottlenecks? trees and treelike networks have high modularity. *CoRR*, abs/1201.0745, 2012.

[4] V. Blondel et al. Fast unfolding of communities in large networks. *JSTAT*, 2008:P10008, 2008.

[5] A. Clauset et al. Finding community structure in very large networks. *Phy. Rev. E*, 70(6):066111, 2004.

[6] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *PNAS*, 104(1):36, 2007.

[7] A. Lancichinetti. Community detection algorithms: a comparative analysis. *Phy. Rev. E*, 80(5):056117, 2009.

[8] A. Lancichinetti, F. Radicchi, J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.

[9] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*, pages 177–187, 2005.

[10] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[11] H. Lu, M. Halappanavar, and A. Kalyanaraman. Parallel heuristics for scalable community detection. *arXiv preprint arXiv:1410.1237*, 2014.

[12] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[13] A. Medus et al. Detection of community structures in networks via global optimization. *Physica A*, 358(2-4):593–604, 2005.

[14] R. R. Nadakuditi and M. E. Newman. Graph spectra and the detectability of community structure in networks. *Physical review letters*, 108(18):188701, 2012.

[15] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phy. Rev. E*, 69(2):026113, 2004.

[16] P. Pons and M. Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.

[17] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. In *CIKM*, pages 1677–1681, 2012.

[18] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 225–236, New York, NY, USA, 2014. ACM.

[19] F. Radicchi. A paradox in community detection. *EPL (Europhysics Letters)*, 106(3):38001, 2014.

[20] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007.

[21] J. Riedy, D. A. Bader, and H. Meyerhenke. Scalable multi-threaded community detection in social networks. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1619–1628. IEEE, 2012.

[22] M. Rosvall and C. Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4):1118, 2008.

[23] J. Xie and B. Szymanski. Towards linear time overlapping community detection in social networks. In *PAKDD*, pages 25–36, 2012.

[24] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

[25] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*, pages 587–596, 2013.