

High Quality, Scalable and Parallel Community Detection for Large Real Graphs

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

David Dominguez-Sal
Sparsity Technologies
david@sparsity-
technologies.com

Josep-LLuis Larriba-Pey
DAMA-UPC
Universitat Politècnica de
Catalunya
larri@ac.upc.edu

ABSTRACT

Community detection has arisen as one of the most relevant topics in the field of graph mining, principally for its applications in domains such as social or biological networks analysis. Different community detection algorithms have been proposed during the last decade, approaching the problem from different perspectives. However, existing algorithms are, in general, based on complex and expensive computations, making them unsuitable for large graphs with millions of vertices and edges such as those usually found in the real world.

In this paper, we propose a novel disjoint community detection algorithm called *Scalable Community Detection* (SCD). By combining different strategies, SCD partitions the graph by maximizing the *Weighted Community Clustering* (*WCC*), a recently proposed community detection metric based on triangle analysis. Using real graphs with ground truth overlapped communities, we show that SCD outperforms the current state of the art proposals (even those aimed at finding overlapping communities) in terms of quality and performance. SCD provides the speed of the fastest algorithms and the quality in terms of NMI and F1Score of the most accurate state of the art proposals. We show that SCD is able to run up to two orders of magnitude faster than practical existing solutions by exploiting the parallelism of current multi-core processors, enabling us to process graphs of unprecedented size in short execution times.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval; H.3.4 [Information Storage and Retrieval]: Systems and Software; G.2.2 [Discrete Mathematics]: Graph Theory

Keywords

Graph Algorithms; Community Detection; Clustering; Parallel; Social Networks; Graph Partition; Modularity; WCC

1. INTRODUCTION

During the last years, the analysis of complex networks has become a hot research topic in the field of data mining. Social, biological, information and collaboration networks are typical targets for such analysis, just to cite a few of them. Among all the tools used to analyze these networks, community detection is one of the most relevant [7, 22]. Communities, also known as clusters, are often referred to as vertices with a high density of connections among them and seldom connected with the rest of the graph [9]. Community detection provides valuable information about the structural properties of the network [5, 9], the interactions among the agents of a network [3] or the role the agents develop inside the network [21].

Community detection algorithms are often computationally expensive and are not scalable to large graphs with billions of edges. Recently, Yang and Leskovec provided a benchmark with real datasets and its corresponding ground truth communities [24]. In such work, they measure the time spent by several state of the art algorithms, such as clique percolation [16] or link clustering [1], and found that they did not scale to networks with more than hundreds of thousands of edges. Even their new proposal aimed at large networks, BigClam [24], was not able to process the largest graph in the benchmark, the Friendster graph, with roughly 2 billion edges. On the other hand, algorithms such as Louvain [4], which can locate communities in graphs with a scale similar to that of Friendster graph, does not scale in quality [2, 8, 10, 22].

In this paper, we present SCD, which is a new community detection algorithm that is much faster than the most accurate state of the art solutions, while maintaining or even improving their quality. SCD is able to compute the communities of the Friendster graph, yet using a modest 32GB RAM computer. Figure 1 illustrates schematically the two most important dimensions to evaluate community detection algorithms: quality and scalability. We observe that no algorithm in the state of the art excels both in scalability and quality.

SCD detects disjoint communities in undirected and unweighted networks by maximizing *WCC*, a recently proposed community metric [18]. *WCC* is a metric based on triangle structures in a community. In contrast to modularity, which is the most widely used metric and has resolution

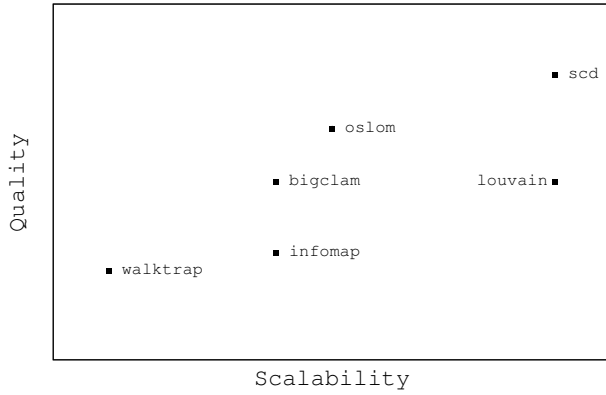


Figure 1: Scale vs quality of different community detection algorithms.

problems when optimized [2,8], *WCC* mathematically guarantees that the communities emerged from its optimization are cohesive and structured [18]. Furthermore, we show in this paper that the computation of *WCC* can be efficiently parallelized, allowing the design of algorithms that take advantage of current multi-core processors.

SCD implements a two-phase procedure that combines different strategies. In the first phase, SCD uses the *clustering coefficient* as an heuristic to obtain a preliminary partition of the graph. In the second phase, SCD refines the initial partition by moving vertices between communities while the *WCC* of the communities increases. In order to speed up this second phase, we propose a *WCC* estimator that approximates the original metric and it is faster to compute.

The evaluation of SCD indicates that the quality of the communities found by SCD is at least as good as the best state of the art algorithms though orders of magnitude faster. Although the communities found by SCD are disjoint, we evaluate our algorithm with overlapping community benchmarks. We note that this quality comparison is biased *against* our algorithm, because SCD is not able to detect overlaps, which are present in the ground truth. The reason of our good quality score despite this handicap is that our algorithm goes beyond edge counting metrics and accounts for triangle structures using *WCC*. The communities obtained using *WCC* are meaningful and precise, and thus have a very good match to real communities. It is beyond the scope of this paper to extend *WCC* and SCD to handle overlapping partitions, which are also known as *covers*.

We summarize the main contributions as follows:

1. We propose a very scalable parallel community detection algorithm that is able to handle graphs with billions of edges. Our algorithm is in the same order of magnitude as the fastest community detection techniques [4], which have, in general, a lower quality than the best state of the art algorithms, and thus far below SCD.
2. By using ground truth real world benchmarks, we show that the quality of the communities of SCD is as good as the best state of the art algorithms [12, 24], which are at least two orders of magnitude slower than SCD.
3. SCD shows that the topological and structural analysis of communities beyond edge counting metrics pro-

vides better quality communities. In particular *WCC*, which is based on triangle counting, is very effective locating meaningful communities.

4. According to our results, we observe that overlapping community detection metrics are still far from obtaining high quality results, since non-overlapping methods are able to obtain better communities using overlapping benchmarks.

The rest of the paper is structured as follows. In Section 2, we review the related work. In Section 3, we describe SCD, including a brief introduction to *WCC*, the proposal of the estimators and an analysis of the complexity of the algorithm. In Section 4, we describe the experimental setup. In Section 5, we show the results and discuss them. In Section 6 we introduce a case study where the use of SCD is proven successful and in Section 7, we conclude the paper and give guidelines for the future work.

2. RELATED WORK

In the literature, we find a wide range of different community detection algorithms which follow different strategies. The biggest family of community detection algorithms is formed by those based on maximizing modularity [14]. Modularity scores high those partitions containing communities with an internal edge density larger than that expected in a given graph model, which is almost always an Erdős-Rényi model. Several strategies have been proposed for its optimization, such as agglomerative greedy [6] or simulated annealing [13]. A multilevel approach has been proposed which scales to graphs with hundreds of millions of objects [4], but the quality of its results decreases considerably as long as the size of the graph increases [10]. Moreover, it has been reported that modularity has resolution limits [2,8]. Modularity is unable to detect small and well defined communities when the graph is large, and its maximization delivers sets with a tree-like structure, which cannot be considered communities. SCD does not suffer from these problems because it is based on *WCC*, whose maximization is proven to deliver cohesive and structured communities [18].

Random walks is a tool on which several community detection algorithms rely. The intuition behind this 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 [17]. Another algorithm based on random walks is Infomap [20]. In this case, a codification for describing random walks based on communities is searched. The codification that requires less memory space (attains the highest compression rates) is selected. According to the comparison performed by Lancichinetti et al. [10], Infomap stands as one of the best community detection algorithms.

Another category of algorithms is that formed by those capable of finding overlapping communities. An example of such an algorithm is Osloom, which uses the significance as a fitness measure in order to assess the quality of a cluster. 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 Link Clustering Algorithm (LCA) [1]. This algorithm is based on the idea of taking edges instead of vertices to form a community. By means of an iterative process, the similarity of adjacent edges (i.e. those edges that share a vertex, forming

an opened triad) is assessed by using the Jaccard coefficient of the adjacency lists of the two other vertices of the edges. In this case, thanks to taking the edges instead of the vertices, the overlapped communities emerge naturally. Label propagation is another family of iterative techniques that initially sets labels to nodes. Then, it defines rules that simulate the spread of these labels in the network similarly to infections [19, 23]. Finally, a recently proposed algorithm is BigClam by Yang et al. [24]. 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 existing an edge between two vertices increases with the number of communities the vertices share (i.e. the number of communities in which the vertices overlap).

Finally, the exploitation of parallelism has been an evasive topic in the field of community detection with few remarkable exceptions [24]. SCD differs from existing solutions by being designed with parallelism in mind, and hence can take advantage of current multi-core architectures.

3. SCALABLE COMMUNITY DETECTION

Scalable Community Detection (SCD) finds disjoint communities in an undirected and unweighted graph by maximizing WCC . First, we briefly introduce WCC , and then we describe the proposed algorithm. For a more detailed description of WCC , please refer to [18].

3.1 WCC

WCC is a community metric based on the fact that real networks contain a large number of triangles due to their community structure. Since communities are groups of highly connected vertices, the probability of these vertices to close triangles among them is larger than the expected between vertices of different communities. WCC uses this feature to quantify the quality of a partition of vertices.

Given a graph $G(V, E)$, a vertex x and a community C , $t(x, C)$ denotes the number of triangles that vertex x closes with the vertices in C and $vt(x, C)$ denotes the number of vertices in C that close at least one triangle with x and another vertex in G . Then, the level of cohesion of a vertex x with respect to community C is denoted by $WCC(x, C)$, which is defined as follows:

$$WCC(x, C) = \begin{cases} \frac{t(x, C)}{t(x, V)} \cdot \frac{vt(x, V)}{|C \setminus \{x\}| + vt(x, V \setminus C)} & \text{if } t(x, V) \neq 0; \\ 0 & \text{if } t(x, V) = 0. \end{cases} \quad (1)$$

Using Equation 1, the WCC of a community C is computed as follows:

$$WCC(S) = \frac{1}{|C|} \sum_{x \in S} WCC(x, C). \quad (2)$$

Given a set C , $WCC(C)$ is the average WCC of all the vertices in C . Finally, the WCC of a graph partition $\mathcal{P} = \{C_1, \dots, C_n\}$ such that $(C_1 \cap \dots \cap C_k) = \emptyset$ is:

$$WCC(\mathcal{P}) = \frac{1}{|V|} \sum_{i=1}^n (|C_i| \cdot WCC(C_i)). \quad (3)$$

The WCC of a partition is the weighted average of the WCC of all the communities in the partition. In other

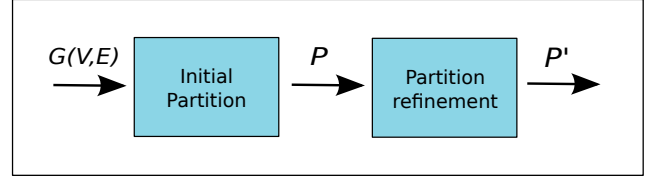


Figure 2: The algorithm's general schema.

words, it is the average WCC of every vertex x in the graph with respect to the community C in P where x belongs.

3.2 Algorithm description

SCD takes a graph G as input, and generates a partition of G resulting from a WCC optimization process. Before starting the execution of the algorithm, we perform a pre-processing step where unnecessary edges of the graph are removed. Figure 2 shows the general structure of SCD. The algorithm is divided into two phases. The first phase consists in finding an initial partition. The second phase is fed with this initial partition and refines it.

3.2.1 Graph loading and preprocessing

During the loading of the graph, we perform the following cleanup of the graph. We compute the number of triangles where each edge in the graph belongs to. Then, we remove from the graph those edges that do not close any triangle. Those edges that do not close any triangle, are irrelevant from the point of view of WCC . Hence, by removing these edges we reduce the memory consumption and improve the performance of SCD. Furthermore, removing these edges simplifies the estimator proposed in Section 3.3, since we can assume that each edge closes at least one triangle.

3.2.2 Initial partition

The initial partition is computed by a fast heuristic process, described in Algorithm 1. First, we compute the clustering coefficient of each vertex (Line 2). This information is needed by this step as well as the refinement step. Then, we sort the vertices of the graph by their clustering coefficient decreasingly. For those vertices with equal clustering coefficient, we use the degree as a second sorting criterion (Line 3). Then, the vertices are iterated and, for each vertex v not previously visited, we create a new community C that contains v and all the neighbors of v that were not visited previously (Line 7 to 13). Finally, community C is added to partition P (Line 14) and all the vertices of the community are marked as visited. The process finishes when all the vertices in the graph have been visited.

The intuition behind this heuristic is the following: the larger the clustering coefficient of a vertex, the larger the number of triangles the vertex closes with its neighbors, and the larger the probability that its neighbors form triangles among them. Hence, considering Equation 1, the larger the clustering coefficient of a vertex, the larger is the probability that the WCC of its neighbors is large if we include them in the same community.

Data: Given a graph $G(V,E)$
Result: Computes a partition of G

```

1 Let  $P$  be a set of sets of vertices;
2 ComputeCC( $G$ );
3  $S \leftarrow \text{sortByCC}(V)$ ;
4 foreach  $v$  in  $S$  do
5   if not visited( $v$ ) then
6     markAsVisited( $v$ );
7      $C \leftarrow v$ ;
8     foreach  $u$  in neighbors( $v$ ) do
9       if not visited( $u$ ) then
10         markAsVisited( $u$ );
11          $C.add(u)$ ;
12       end
13     end
14      $S.add(C)$ ;
15   end
16 end
17 return  $P$ ;

```

Algorithm 1: Phase 1, initial partition.

3.2.3 Partition refinement

The partition refinement phase is described in Algorithm 2. The goal of this phase is to refine the partition received from the previous phase. This phase follows a hill climbing strategy, that is, in each iteration, a new partition is computed from the previous one by performing a set of modifications (movements of vertices between communities). The algorithm repeats the process until the WCC of the new partition does not percentually improve over the previous one more than a given threshold. In our tests, this threshold is set at 1%, which provided a good tradeoff between performance and quality.

In each iteration, for each vertex v of the graph, we use the **bestMovement** function to compute the movement of v that improves the WCC of the partition most (Line 8). There are three types of possible movements:

- No Action: leave the vertex in the community where it currently is.
- Remove: remove the vertex from its current community and place it as a singleton (a community formed by a single vertex).
- Transfer: remove the vertex from its current community and insert it into another one.

Note that **bestMovement** does not modify the current partition, and that the best movement of each vertex is computed independently from the others. This feature allows the computation of the best movements for all the vertices in the graph in parallel. Once each vertex of the graph has a best movement, we apply it *simultaneously* to all the vertices (**applyMovements** Line 10). Finally, we update the WCC of the new partition (Line 11) and check whether it improved since the last iteration.

Before describing function **bestMovement**, we introduce some auxiliary functions that are used in its computation. The proofs of the theorems introduced are Appendix A.

- $WCC_I(v, C)$ computes the improvement of the WCC of a partition when vertex v (which belongs to a singleton community) is inserted into community C .

Data: Given a graph $G(V,E)$ and a partition P
Result: A refined partition P'

```

1 newP  $\leftarrow P$ ;
2 newWCC  $\leftarrow \text{computeWCC}(P)$ ;
3 repeat
4   WCC'  $\leftarrow \text{newWCC}$ ;
5   P'  $\leftarrow \text{newP}$ ;
6   M  $\leftarrow \emptyset$ ;
7   foreach  $v$  in  $V$  do
8     M.add( bestMovement( $v, P'$ ) );
9   end
10  newP  $\leftarrow \text{applyMovements}(M, P')$ ;
11  newWCC  $\leftarrow \text{computeWCC}(\text{newP})$ ;
12 until  $(\text{newWCC} - \text{WCC}') / \text{WCC}' \geq t$ ;
13 return P';

```

Algorithm 2: Phase 2, refinement.

Theorem 1. Let $P = \{C_1, \dots, C_k, \{v\}\}$ and $P' = \{C'_1, \dots, C_k\}$ be partitions of a graph $G = (V, E)$ where $C'_1 = C_1 \cup \{v\}$. Then,

$$\begin{aligned}
WCC(P') - WCC(P) &= WCC_I(v, C_1) \\
&= 1/|V| \cdot \sum_{x \in C_1} [WCC(x, C'_1) - WCC(x, C_1)] + \\
&\quad 1/|V| \cdot WCC(v, C'_1).
\end{aligned}$$

- $WCC_R(v, C)$ computes the improvement of the WCC of a partition when vertex v is removed from community C and placed as a singleton community.

Theorem 2. Let partitions $P = \{C_1, \dots, C_k\}$ and $P' = \{C'_1, \dots, C_k, \{v\}\}$ of a graph $G = (V, E)$ where $C_1 = C'_1 \cup \{v\}$. Then,

$$WCC(P') - WCC(P) = WCC_R(v, C_1) = -WCC_I(v, C'_1).$$

- $WCC_T(v, C_1, C_2)$ computes the improvement of the WCC of a partition when vertex v is transferred from community C_1 and to C_2 .

Theorem 3. Let $P = \{C_1, C_2, \dots, C_{k-1}, C_k\}$ and $P' = \{C'_1, C_2, \dots, C_{k-1}, C'_k\}$ be partitions of a graph $G = (V, E)$ where $C_1 = C'_1 \cup \{v\}$ and $C'_k = C_k \cup \{v\}$. Then,

$$\begin{aligned}
WCC(P') - WCC(P) &= WCC_T(v, C_1, C_k) \\
&= -WCC_I(v, C'_1) + WCC_I(v, C_k).
\end{aligned}$$

From Theorem 1, we conclude that in order to compute the improvement of WCC derived from inserting a vertex v (i.e. a singleton community) into a community C , only the WCC of vertex v and those vertices in C has to be computed. This limits the number of computations to perform a movement, because only a very local portion of the graph needs to be accessed for each vertex. Theorems 2 and 3 show that we can express any of the movements needed by the algorithm, in terms of function $WCC_I()$, simplifying the implementation of the algorithm.

Algorithm 3 describes the **bestMovement** function. First, we compute the improvement of removing vertex v from its

Data: Given a graph $G(V,E)$ a partition P and a vertex v

Result: Computes the best movement of v .

```

1  $m \leftarrow [\text{NO\_ACTION}];$ 
2  $\text{sourceC} \leftarrow \text{GetCommunity}(v,P);$ 
3  $\text{wcc}_r \leftarrow \text{WCC}_R(v,\text{sourceC});$ 
4  $\text{wcc}_t \leftarrow 0.0;$ 
5  $\text{bestC} \leftarrow \emptyset;$ 
6  $\text{Candidates} \leftarrow \text{candidateCommunities}(v,P);$ 
7 for  $c$  in  $\text{Candidates}$  do
8    $\text{aux} \leftarrow \text{WCC}_T(v,\text{sourceC},c);$ 
9   if  $\text{aux} > \text{wcc}_t$  then
10      $\text{wcc}_t \leftarrow \text{aux};$ 
11      $\text{bestC} \leftarrow c;$ 
12   end
13 end
14 if  $\text{wcc}_r > \text{wcc}_t$  and  $\text{wcc}_r > 0.0$  then
15    $m \leftarrow [\text{REMOVE}];$ 
16 else if  $\text{wcc}_t > 0.0$  then
17    $m \leftarrow [\text{TRANSFER}, \text{bestC}];$ 
18 end
19 return  $m;$ 

```

Algorithm 3: bestMovement.

current community (Line 3). Then, we obtain the set of candidate communities, formed by those communities containing the neighbors of v (Line 6). After that, we calculate which is the candidate community where transferring vertex v improves the WCC most (Lines 7 to 13). Finally, we select whether the best improvement is obtained from removing the vertex from its current community (REMOVE) or transferring it into a new community (TRANSFER) (Lines 14 to 18). If neither of the two movements improves the WCC of the partition, we keep the vertex in the current community (NO_ACTION) (Line 1).

3.3 WCC_I Estimation

The exact computation of WCC needs the computation of the triangles of the target vertex with the rest of its neighbors. For a vertex of degree d , the complexity of this operation is $O(d^2)$. And, since real graphs typically have power law distributions, this cost is large for the highest degree vertices in the graph. Furthermore, considering that the functions that compute the WCC improvements are called several times per vertex and iteration, they become the most time consuming part of the algorithm. As shown in the previous section, all auxiliary functions can be computed with only the $WCC_I()$ function. In this section, we propose a model to estimate $WCC_I()$ with a constant time complexity function (given some easy to compute statistics) that we call $WCC'_I()$.

$WCC'_I()$ stands as the approximated increment of WCC when vertex v is inserted into a community C . We depict the simplified schema in Figure 3. For the considered vertex v , we only record the number of edges that connect it to community C . For each community C , we keep the following statistics: the size of the community r ; the edge density of the community δ ; and the number of edges b that are in the boundary of the community. We also record the clustering coefficient of the graph ω , which is constant along all the community detection process. These statistics homoge-

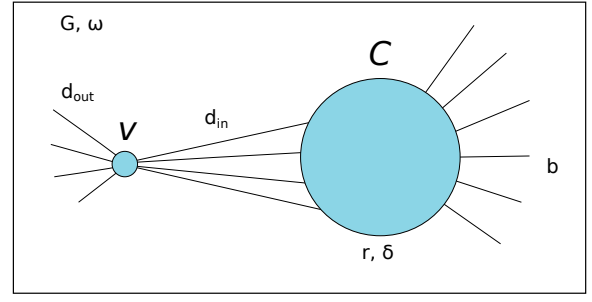


Figure 3: Model for estimating the WCC improvement.

nize the community members and allow the computation of $WCC'_I()$ as follows:

Theorem 4. Consider the situation depicted in Figure 3, with the following assumptions:

- Every edge in the graph closes at least one triangle.
- The edge density inside community C is homogeneous and equal to δ .
- The clustering coefficient of the graph is homogeneous for all nodes outside C and equals to ω .

Then,

$$WCC(P') - WCC(P) = WCC'_I(v, C) = \frac{1}{V} \cdot (d_{in} \cdot \Theta_1 + (r - d_{in}) \cdot \Theta_2 + \Theta_3), \quad (4)$$

where,

$$\begin{aligned} \Theta_1 &= \frac{(r-1)\delta+1+q}{(r+q) \cdot ((r-1)(r-2)\delta^3 + (d_{in}-1)\delta + q(q-1)\delta\omega + q(q-1)\omega + d_{out}\omega)} \cdot (d_{in}-1)\delta; \\ \Theta_2 &= -\frac{(r-1)(r-2)\delta^3}{(r-1)(r-2)\delta^3 + q(q-1)\omega + q(r-1)\delta\omega} \cdot \frac{(r-1)\delta+q}{(r+q)(r-1+q)}; \\ \Theta_3 &= \frac{d_{in}(d_{in}-1)\delta}{d_{in}(d_{in}-1)\delta + d_{out}(d_{out}-1)\omega + d_{out}d_{in}\omega} \cdot \frac{d_{in}+d_{out}}{r+d_{out}} \end{aligned}$$

and $q = (b - d_{in})/r$.

Conceptually, Θ_1 , Θ_2 and Θ_3 are the WCC improvements of those vertices in C connected to v , those vertices in C not connected to v , and vertex v respectively, when v is added to community C . The evaluation of Equation 4 is $O(1)$ given all the statistics. And, the update of all statistics is only performed when all communities are updated, with a cost $O(m)$ for the whole graph. Note that we use aggregated statistics to estimate the number of triangles, and thus we are *not* computing the triangles when we compute $WCC'_I()$.

3.4 Complexity of the Algorithm

Let n be the number of vertices and m the number of edges in the graph. We assume that the average degree of the graph is $d = m/n$ and that real graphs have a quasi-linear relation between vertices and edges $O(m) = O(n \cdot \log n)$.

In the initial partition phase, for each edge in the graph, we compute the triangles that each edge participates in. The triangles are found by intersecting the adjacency lists of the two connected vertices. Since we assume sorted adjacency lists, the complexity of computing the intersection is $O(d)$. Since the average degree is m/n , we have that the cost of the first phase is $O(m \cdot d) = O(m \cdot \log n)$.

	Vertices	Edges	Communities	% non-overlap	% two-overlap	% three-plus-overlap
Amazon	334,863	925,872	151,037	3.9	3.6	92.4
Dblp	317,080	1,049,866	13,477	57.5	16.6	25.9
Youtube	1,134,890	2,987,624	8,385	62.4	16.1	21.5
LiveJournal	3,997,962	34,681,189	287,512	35.8	17.2	47.0
Orkut	3,072,441	117,185,083	6,288,363	6.2	5.7	88.1
Friendster	65,608,366	1,806,067,135	957,154	45.5	20.8	33.6

Table 1: Characteristics of the test graphs.

Regarding the second phase, let α be the number of iterations required to find the best partition P' , which in our experiments is between 3 and 7. In each iteration, for each vertex v of the graph, we compute, in the worst case, $d + 1$ movements of type $WCC'(I)$ that have a cost $O(1)$. Then, the computation of the best movement for all vertices in the graph in an iteration is $O(n \cdot (d + 1)) = O(m)$. The application of the all the movements is linear with respect to the number of vertices $O(n)$. We also need to update, for each iteration of the second phase, the statistics δ , c_{out} , d_{in} and d_{out} for each vertex and community, which has a cost of $O(m)$. Finally, the computation of the WCC for the current partition is performed by computing for each edge the triangles, which is $O(m \cdot \log n)$ as already stated. Hence, the cost of the refinement phase becomes $O(\alpha \cdot (m + n + m + m \cdot \log n))$, which after simplification, becomes $O(m \cdot \log n)$ assuming α as constant.

The final cost of the algorithm is the sum of the two phases: $O(m \cdot \log n + m \cdot \log n) = O(m \cdot \log n)$

4. EXPERIMENTAL SETUP

For the experimental evaluation of SCD, we used all the community benchmark datasets provided by SNAP¹ that include a community ground truth, which we use to verify the quality of the algorithms. To our knowledge, these are the largest datasets that include such a gold standard verification, which range from a million edges to almost two billion edges, as reported in Table 1. The communities field indicates the number of ground truth communities. Note that, for all the graphs, a vertex can belong to more than one community. In Table 1, we show the percentage of vertices that belong to one, two, or three or more ground truth communities. We see that the number of overlaps is not homogeneous and in some graphs a vertex tends to participate in more communities than others.

In this paper, we do not make any special distinction between disjoint and overlapping community detection algorithms. We are interested in determining how *meaningful* are the output communities. Therefore, we simply analyze the matching between the output of the algorithms and the real communities specified in the ground truth. We consider better, in terms of quality, those algorithms that have better matching with the gold standard, independently of the type of algorithm under consideration.

Amazon: This graph represents a network of products, where each vertex is a product and an edge exists between two products if they have been co-purchased frequently. The product categories provided by Amazon define the ground truth communities.

DBLP: This graph represents a network of coauthorships, where each vertex is an author and two authors are connected if they have written a paper together. Each journal or conference defines a ground truth community formed by those authors which published in that journal or conference.

Youtube: This graph represents the Youtube social network, where each vertex is a user and two users are linked if they have established a friendship relation. Communities are defined by the groups created by the users, which are formed by those users that joined that group.

LiveJournal: This graph represents the social network around LiveJournal. Similar to the Youtube network, the vertices are the users, which establish friendship relationships with other users. Users can create groups, which define the ground truth communities and which are formed by those users that joined that group.

Orkut: This graph represents the Orkut social network. Similar to the Youtube and LiveJournal networks, the vertices are the users of the social network and the edges represent the friendship relationships between the users. The groups created by the users define the ground truth communities, which are formed by those users that joined the group.

Friendster: This graph represents the Friendster gaming social network, where each vertex is a user of the network and two users are connected if they established a friendship relation. In this network, users create groups, which define the ground truth communities. Then, each ground truth community is formed by those vertices in that group.

We select the most relevant community detection algorithms in the state of the art: Infomap [20], Louvain [4], Walktrap [17], BigClam [24] and Oslo [12]. Infomap, Louvain and Walktrap are considered the best algorithms for disjoint community detection, according to [10, 15]. On the other hand, we take BigClam and Oslo as the state of the art of overlapping community detection. We use the implementations provided in the website of the authors and are all implemented in C++. SCD was implemented in C++ and can be found in the website of the authors².

We use two metrics to evaluate the quality of the different algorithms. The first metric is the **Average F1Score**, \bar{F}_1 , following the approach taken by the authors of the community benchmark [24]. The F1Score of a set A with respect to a set B is defined as the harmonic mean (H) of the precision

¹<http://snap.stanford.edu>

²<http://www.dama.upc.edu>

and the recall of A with respect to B :

$$\text{precision}(A, B) = \frac{|A \cap B|}{|A|}, \text{recall}(A, B) = \frac{|A \cap B|}{|B|}.$$

$$H(a, b) = \frac{2 \cdot a \cdot b}{a + b}$$

$$F_1(A, B) = H(\text{precision}(A, B), \text{recall}(A, B))$$

Then, the average F1Score of two sets of communities C_1 and C_2 is given by:

$$F_1(A, C) = \arg \max_i F_1(A, C_i), c_i \in C = \{C_1, \dots, C_n\}$$

$$\bar{F}_1(C_1, C_2) = \frac{1}{2|C|} \sum_{c_i \in C} F_1(c_i, C') + \frac{1}{2|C'|} \sum_{c_i \in C'} F_1(c_i, C)$$

The second metric is the **Normalized Mutual Information (NMI)** [11], which is based on information theory concepts. The NMI provides a real number between zero and one that gives the similarity between two sets of sets of objects. An exact match between the two inputs obtains a value of one.

The computer used in the experiments has: a CPU Intel Xeon E5530 at 2.4 GHz, 32 GB of RAM, 1TB of disk space and Debian Linux with kernel 2.6.32-5-amd64.

5. EXPERIMENTAL RESULTS

5.1 Quality and Performance

Figure 4 shows the execution time of the different tested algorithms. Missing columns indicate that the algorithm was not able to process that graph due to memory consumption, or because it took more than a week of computing. All the state of the art algorithms are executed using their default parameters. In BigClam, parameter k , which indicates the number of communities, is set to the actual number of ground truth communities. In this experiment, all the algorithms were set to run as single threaded, including ours, because most implementations provided were single threaded. Note that since the differences between the fast and the slow algorithms are in the orders of magnitude, multi-threading does not alter the conclusions.

When we look at the execution times, we see that there are mainly two groups of algorithms in terms of scalability: SCD and Louvain on one side, and the rest of algorithms on the other. In general, the group formed by SCD and Louvain, runs about two orders of magnitude faster than the rest. For practical purposes, this computational complexity creates a barrier to analyze large networks by the group of slow algorithms. Oslom takes several days to analyze the Orkut graph whereas SCD finds the communities in a few minutes. Even assuming that these slow algorithms scale linearly with the problem size, which is not true for most of them, the analysis of large graphs may require unaffordable times. We see that SCD's performance is better than for Louvain, the fastest algorithm in the state of the art.

Figures 5 and 6 show the \bar{F}_1 and NMI quality scores of the tested algorithms, respectively. We observe that both metrics are correlated, though some small deviations exist among them. From these results, we conclude that SCD obtains the best quality, followed by Oslom and Louvain. In the case of \bar{F}_1 , SCD obtains the best quality in all the test graphs except Livejournal, where it is close to BigClam. On the other hand, Oslom stands as the second best option

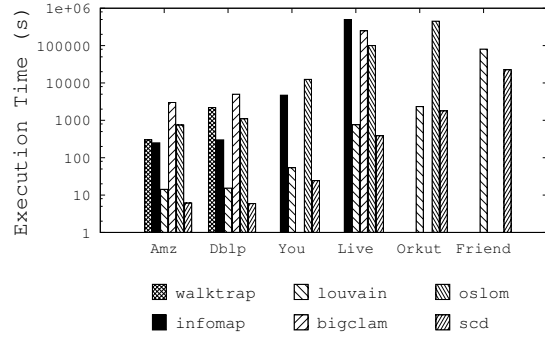


Figure 4: Execution times in seconds.

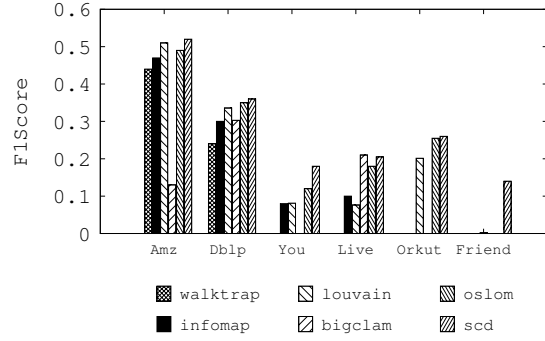


Figure 5: \bar{F}_1 score using ground truth.

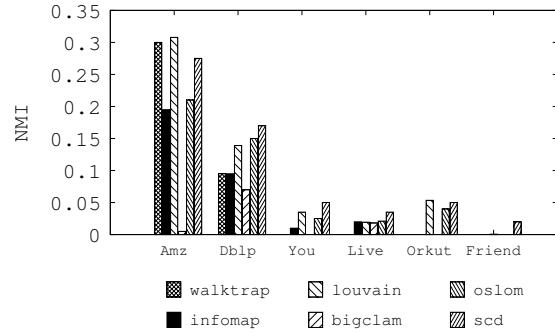


Figure 6: NMI using ground truth.

except for Amazon and Livejournal. Note that the \bar{F}_1 of Louvain for the Friendster graph cannot be appreciated in the figure due to its small value. When we turn into NMI , we see that SCD obtains the best quality in four out of six of the tested graphs. For the rest of the graphs, SCD obtains a score close to the best one. In terms of NMI , Louvain and Oslom are the closest ones to SCD on average, but with less quality in most of the graphs. Broadly speaking, SCD improves the quality of the best community detection algorithms but running much faster.

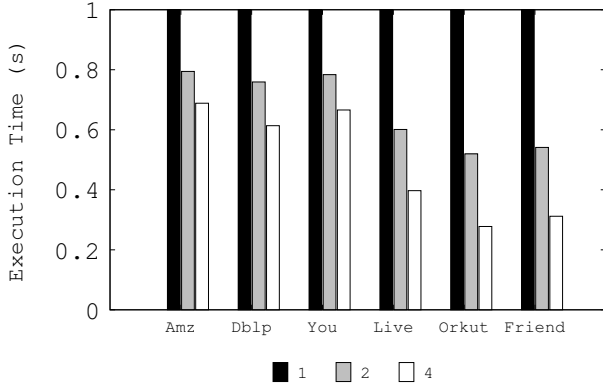


Figure 7: SCD normalized execution time for different number of threads.

5.2 Scalability

We parallelized the two most time consuming parts of the algorithm: the computation of the clustering coefficient of the vertices and the refinement part. In the former, we parallelized the loop that computes, for each edge, the number of triangles that the edge closes. In the later, we parallelized the loop in Line 7 of Algorithm 2, which calls the function `bestMovement` for each of the vertices in the graph, as well as the computation of *WCC* for the partition at the end of the iteration. Since all the parallel code is in the form of loops, we used OpenMP with dynamic scheduling, using a chunk size of 32. Figure 7 shows the normalized execution times of SCD with different number of threads. In this experiment, we have excluded the time spent in I/O, which includes reading the graph file and printing the results.

Broadly speaking, we see that SCD is able to achieve very good scalability, specially for the larger graphs which are also the graphs with the largest degree. The larger the degree of the graph, the larger the cost of those parts that have been parallelized, which become dominant over the sequential ones. This translates into a better scalability due to a direct application of Ahmdal’s Law.

For large graphs, the implementation of SCD is able to exploit all the processor’s resources available. The configuration with four threads of SCD keeps the four cores of the processor active, and hence obtains about a four fold improvement over the single threaded version. These results show that SCD is an algorithm capable of exploiting multi-core architectures efficiently, especially on large graphs where this feature is more appreciated. More specifically, SCD processes the Friendster graph using four threads in just 4.3 hours.

Figure 8 shows the execution time of SCD with respect to the number of edges of the graph. Each point represents the time spent by the four thread version of SCD in the six datasets. We observe that SCD scales approximately as a line with respect to the number of edges of the graph.

5.3 Memory Consumption

In Table 2, we show the memory consumption of SCD for each of the graphs. We split the memory consumption into three different concepts:

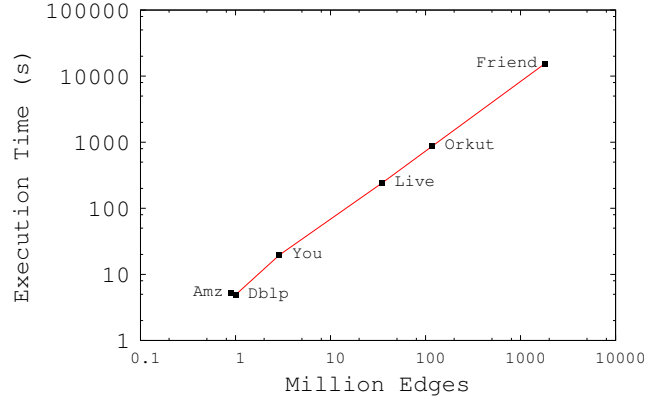


Figure 8: SCD execution time with four threads vs number of edges.

	Graph	Triangles	Partitions	Total
Amazon	11.4	1.3	16.0	28.7
Dblp	12.2	1.3	14.9	28.4
Youtube	37.5	4.5	68.9	110.9
Livejournal	325.4	16.0	197.7	539.1
Orkut	974.3	12.3	124.4	1111.0
Friendster	15235.8	262.4	3317.6	18815.8

Table 2: SCD Memory consumption in MB.

- **Graph:** the size of the data structure that stores the graph as a list of adjacencies. We relabel the indices of the vertices to the range from 1 to n , and hence, we also account an array containing a mapping between our vertex identifier and the original identifier.
- **Triangles:** an array with size equals to the number of vertices, which contains the number of triangles each vertex belongs to.
- **Partitions:** accounts the statistics for the partitions. We report the iteration with the largest memory consumption.

The auxiliary data structures (triangles and partitions) built by SCD scale linearly with the number of vertices of the graph, and not with the number of edges. Since the number of statistics stored per node is small, the amount of memory consumed by SCD is often dominated by the graph representation itself. Among the tested benchmarks, the only exception is Youtube, because of its very small average degree of 2.6. For the largest graphs, SCD allocates only data structures for an additional 23% (Friendster) and 14% (Orkut) of the original graph. The amount of memory consumed for the Friendster graph is roughly 18GB, indicating that even larger graphs could be processed with the 32 GB of memory of the test machine.

6. CASE STUDY

In this section, we present results showing the behavior of SCD in a real world application, such as product purchasing recommendation. We have downloaded, from the SNAP website, the metadata associated with the products of the co-purchasing Amazon graph described above. This

Modern Information Retrieval	Finding Out About: A Cognitive Perspective on Search Engine Technology and the WWW (Richard K. Belew); Understanding Search Engines: Mathematical Modeling and Text Retrieval (Michael W. Berry); Managing Gigabytes: Compressing and Indexing Documents and Images (Ian H. Witten); Foundations of Statistical Natural Language Processing (Christopher D. Manning);	Information Retrieval: Data Structures and Algorithms (William B. Frakes and Ricardo Baeza-Yates); Modern Information Retrieval (Ricardo Baeza-Yates); Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization (P. Jackson and I. Moulinier); Readings in Information Retrieval (Karen Spark Jones and Peter Willett);
Harry Potter	Harry Potter and the Sorcerer's Stone (Book 1); Harry Potter and the Chamber of Secrets (Book 2); Harry Potter and the Prisoner of Azkaban (Book 3); Harry Potter and the Goblet of Fire (Book 4); Harry Potter and the Order of the Phoenix (Book 5); Harry Potter and the Sorcerer's Stone (Hardcover); Harry Potter and the Prisoner of Azkaban (Hardcover);	Harry Potter and the Order of the Phoenix (Book 5, Deluxe Edition); Harry Potter and the Sorcerer's Stone (Book 1, Large Print); Harry Potter and the Sorcerer's Stone (Book 1, Audio CD); Harry Potter and the Prisoner of Azkaban (Book 3, Audio CD); Harry Potter and the Goblet of Fire (Book 4, Audio CD); Harry Potter and the Order of the Phoenix (Book 5, Audio);
Lonely Planet Barcelona	Eyewitness Top 10 Travel Guides: Barcelona; Barcelona and Catalonia (Eyewitness Travel Guides); Eyewitness Travel Guide to Barcelona and Catalonia; Lonely Planet Barcelona;	The National Geographic Traveler: Barcelona; Lonely Planet Barcelona City Map; Streetwise Barcelona;

Table 3: Examples of communities of co-purchased products of WCC.

metadata includes information such as the title of the product, the type and the categories that it belongs to, and was collected in 2006. As stated above, a ground truth community in the Amazon graph is formed by those products belonging to the same groups and form a connected component. Therefore, the communities in the Amazon graph should contain similar products than have been usually co-purchased. Hence, once a new buyer purchases a product, it could be recommended with those products of the same communities of that he bought. We select three profiles of well known products: the technical computer science book titled “Modern Information Retrieval”, by Ricardo Baeza; the first book of a popular novel series “Harry Potter and the Sorcerer’s Stone”; and a travel book “Lonely Planet Barcelona”.

We run the SCD algorithm on the whole network and we report the communities where these books were assigned in Table 3. In the case of “Modern Information Retrieval”, we see that the community is formed by relevant books in the field of information retrieval and text analysis. In the case of Harry Potter, we see that the community contains different books of the Harry Potter series (for the curious reader, the two last books of the series were published after the crawl and are not present in the dataset), as well as Harry Potter audio books and some special editions. Finally, for the “Lonely Travel Barcelona” guide, the community found Barcelona travel guides from other publishers. We observe that SCD is able to perform a good selection of the relations in the graphs in order to give meaningful communities.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed SCD, a novel algorithm for disjoint community detection based on optimizing WCC . We proposed a mechanism to estimate WCC , allowing the computation of WCC faster. We compared SCD with the best community detection algorithms in the state of the art, by means of a methodology for overlapping community detection using ground truth data. SCD is able to detect communities as meaningful (and in the most of the cases better) as the most high-profile algorithms in the literature. More-

over, the results show that SCD is able to run faster than these highest quality existing solutions, matching the speed of those algorithms aimed at large scale graphs. This translates into SCD being able to process graphs of an unprecedented size, such the Friendster which has roughly 2 billion edges in just 4.3 hours on off the shelf computer hardware. The design of SCD also allows a remarkable scalability that is close to four fold improvements in a four core processor. Also, we showed that SCD is able to deliver meaningful communities, by means of a case study consisting of a product recommendation application. Finally, we can conclude that going beyond edge counting, i.e. focusing on richer structures such as triangles for community detection, provides better results.

The fact that SCD, being a disjoint community detection algorithm, performs better than pure overlapping community detection algorithms, gives us the hint that overlapping community detection is a problem still far from being solved. Hence, one of the future research lines is to extend the ideas behind the topological analysis of the graph performed by SCD to overlapped communities. On the other hand, we have seen that SCD is able to scale on current multi-core architectures. Another interesting research line to explore in the future is how to adapt SCD to a vertex centric large graph processing model such as GraphLab or Pregel.

8. ACKNOWLEDGEMENTS

The members of DAMA-UPC thank the Ministry of Science and Innovation of Spain and Generalitat de Catalunya, for grant numbers TIN2009-14560-C03-03 and GRC-1187 respectively. The members of DAMA-UPC thank the EU FP7 project LDBC (FP7-ICT2011-8-317548) for funding the LDBC project. David Dominguez-Sal thanks the Ministry of Science and Innovation of Spain for the grant Torres Quevedo PTQ-11-04970. The authors would like to thank the reviewers for their useful comments.

APPENDIX

A. PROOF OF THEOREM 1

Proof.

$$\begin{aligned}
& WCC(P') - WCC(P) = \\
& = \frac{1}{|V|} \left(|C_1 \cup \{v\}| \cdot WCC(C_1 \cup \{v\}) + \sum_{i=2}^k |C_i| \cdot WCC(C_i) \right) - \\
& \quad \frac{1}{|V|} \left(|C_1| \cdot WCC(C_1) + \sum_{i=2}^k |C_i| \cdot WCC(C_i) + WCC(\{v\}) \right) \\
& = \frac{1}{|V|} (|C'_1| \cdot WCC(C'_1)) - \frac{1}{|V|} (|C_1| \cdot WCC(C_1) + 0) \\
& = \frac{1}{|V|} \left(\sum_{x \in C'_1} WCC(x, C'_1) + \sum_{x \in C_1} WCC(x, C_1) \right) \\
& = \frac{1}{|V|} \left(\sum_{x \in C_1} WCC(x, C'_1) + WCC(v, C'_1) - \right. \\
& \quad \left. \sum_{x \in C_1} WCC(x, C_1) \right) \quad \square
\end{aligned}$$

B. PROOF OF THEOREM 2

Proof. As stated in the theorem assumptions, the partition P' is build by removing v from C_1 . Alternatively, the partition P can be build by removing vertex v to C'_1 in P' . Then, the two following equalities hold:

$$\begin{aligned}
WCC(P) + WCC_R(v, C_1) &= WCC(P'), \\
WCC(P) &= WCC(P') + WCC_I(v, C'_1) \\
\text{and thus: } WCC_R(v, C_1) &= -WCC_I(v, C'_1) \quad \square
\end{aligned}$$

C. PROOF OF THEOREM 3

Proof. Since WCC is a state function, all paths from P to P' have the same differential. Then, we express the transfer operation as a combination of remove and insert:

$$\begin{aligned}
WCC(P) + WCC_T(v, C_1, C_k) &= WCC(P') \\
WCC(P) + WCC_R(v, C_1) + WCC_I(v, C_k) &= WCC(P') \\
WCC(P') - WCC(P) &= -WCC_I(v, C'_1) + WCC_I(v, C_k) \quad \square
\end{aligned}$$

D. PROOF OF THEOREM 4

Proof. Consider the situation depicted in Figure 3. Let $N(x)$ be the set of neighbors of x . Given that, we define sets $F = N(v) \cap C$ which contains those vertices in C that are actual neighbors of v , and $G = (C \setminus N(x))$, which contains those vertices in C that are not neighbors of v . Therefore, from Theorem 1 we have:

$$\begin{aligned}
WCC_I(v, C) &= \\
& = \frac{1}{|V|} \sum_{x \in C} (WCC(x, C \cup \{v\}) - WCC(x, C)) + \\
& \quad \frac{1}{|V|} WCC(v, C \cup \{v\}) \\
& = \frac{1}{|V|} \sum_{x \in F} (WCC(x, C \cup \{v\}) - WCC(x, C)) + \\
& \quad \frac{1}{|V|} \sum_{x \in G} (WCC(x, C \cup \{v\}) - WCC(x, C)) + \\
& \quad \frac{1}{|V|} WCC(v, C \cup \{v\})
\end{aligned}$$

We know that $|F| = d_{in}$ and $|G| = r - d_{in}$, then we can define $WCC'_I(v, C)$ with respect to three variables Θ_1 , Θ_2

and Θ_3 , which represent the WCC improvement of a vertex of F , a vertex of G and v respectively. Then,

$$WCC'_I(v, C) = \frac{1}{|V|} (|F| \cdot \Theta_1 + |G| \cdot \Theta_2 + \Theta_3).$$

We define $q = (b - d_{in})/r$ as the number of edges connecting each vertex in C with the rest of the graph excluding v . Then,

(i) If $x \in F$, we have

$$\begin{aligned}
t(x, C) &= (r-1)(r-2)\delta^3; \\
t(x, C \cup \{v\}) &= (r-1)(r-2)\delta^3 + (d_{in} - 1)\delta; \\
t(x, V) &= (r-1)(r-2)\delta^3 + (d_{in} - 1)\delta + q(r-1)\delta\omega + \\
& \quad q(q-1)\omega + d_{out}\omega; \\
vt(x, V) &= (r-1)\delta + 1 + q; \\
|C \cup \{v\} \setminus \{x\}| + vt(x, V \setminus \{C \cup \{v\}\}) &= r + q; \\
|C \setminus \{x\}| + vt(x, V \setminus C) &= r - 1 + q + 1 = r + q;
\end{aligned}$$

In $t(x, C)$, we account for those triangles that x closes with two other vertices in C . Similarly, in $t(x, C \cup \{v\})$ we account for those triangles that x closes with two other vertices in C , and those triangles that x closes with v and another vertex in C . $t(x, V)$ accounts for all triangles that vertex x closes with the graph, which are: $t(x, C \cup \{v\})$ plus those triangles that vertex x closes with another vertex of C and a vertex of $V \setminus C$, plus those triangles that vertex x closes with two other vertices in $V \setminus C$, plus those triangles vertex x closes with v and another vertex of $V \setminus C$. Since we assume that every edge in the graph closes at least one triangle, $vt(x, V)$ accounts for the number of vertices in C that are actual neighbors of x plus 1 (for vertex v) and q vertices that are connected to x . Finally, we have that the union of vertices in C and those vertices in V with whom x closes at least one triangle is $r + q$. Therefore,

$$\begin{aligned}
\Theta_1 &= WCC(x, C \cup \{v\}) - WCC(x, C) \\
&= \frac{t(x, C \cup \{v\})}{t(x, V)} \cdot \frac{vt(x, V)}{|C \cup \{v\} \setminus \{x\}| + vt(x, V \setminus \{C \cup \{v\}\})} - \\
& \quad \frac{t(x, C)}{t(x, V)} \cdot \frac{vt(x, V)}{|C \setminus \{x\}| + vt(x, V \setminus C)} \\
&= \frac{vt(x, V)}{(r+q) \cdot t(x, V)} \cdot (t(x, C \cup \{v\}) - t(x, C)) \\
&= \frac{(r-1)\delta + 1 + q}{(r+q) \cdot ((r-1)(r-2)\delta^3 + (d_{in} - 1)\delta + q(r-1)\delta\omega + q(q-1)\omega + d_{out}\omega)} \cdot (d_{in} - 1)\delta.
\end{aligned}$$

(ii) If $x \in B$, we have

$$\begin{aligned}
t(x, C) &= (r-1)(r-2)\delta^3; \\
t(x, C \cup \{v\}) &= (r-1)(r-2)\delta^3; \\
t(x, V) &= (r-1)(r-2)\delta^3 + q(q-1)\omega + q(r-1)\delta\omega; \\
vt(x, V) &= (r-1)\delta + q; \\
|C \cup \{v\} \setminus \{x\}| + vt(x, V \setminus \{C \cup \{v\}\}) &= r + q; \\
|C \setminus \{x\}| + vt(x, V \setminus C) &= r - 1 + q;
\end{aligned}$$

$t(x, C)$ accounts for those triangles that x closes with two other vertices in C . Since, x is not connected to v , we have that $t(x, C) = t(x, C \cup \{v\})$. $t(x, V)$ accounts for the number of triangles that x closes with the rest of vertices in V . These are $t(x, C)$ plus those triangles that vertex x closes with

another vertex of C and a vertex of $V \setminus C$, plus those triangles that vertex x closes with two other vertices in $V \setminus C$. $vt(x, V)$ accounts for the number of vertices in V with whom x closes at least one triangle, which are the neighbors of x in C and those t vertices with whom x is connected. Finally, we have that the union of vertices in $C \cup \{v\}$ and vertices in V with whom x closes at least one triangle is $r + q$, and the union of vertices in C and vertices in V with whom x closes at least one triangle is $r + q - 1$. Therefore,

$$\begin{aligned}\Theta_2 &= \frac{WCC(x, C \cup \{v\}) - WCC(x, C)}{t(x, V)} \cdot \frac{vt(x, V)}{|C \cup \{v\} \setminus \{x\}| + vt(x, V \setminus \{C \cup \{v\}\})} - \\ &= \frac{t(x, C \cup \{v\})}{t(x, V)} \cdot \frac{vt(x, V)}{|C \setminus \{x\}| + vt(x, V \setminus C)} = \\ &= -\frac{(r-1)(r-2)\delta^3}{(r-1)(r-2)\delta^3 + q(q-1)\omega + q(r-1)\delta\omega} \cdot \frac{(r-1)\delta + q}{(r+q)(r-1+q)}.\end{aligned}$$

(iii) If $x = v$ we have

$$\begin{aligned}t(x, C \cup \{v\}) &= d_{in}(d_{in} - 1)\delta; \\ t(x, V) &= d_{in}(d_{in} - 1)\delta + d_{out}(d_{out} - 1)\omega + d_{out}d_{in}\omega; \\ vt(x, V) &= d_{in} + d_{out}; \\ |C| + vt(x, V \setminus C) &= r + d_{out};\end{aligned}$$

In this case, $t(x, C \cup \{v\})$ accounts for those triangles that x closes with C , with whom it is connected to d_{in} vertices. $t(x, V)$ are those vertices vertex x closes with V , which are those x closes with C plus those x closes with other two vertices in $V \setminus C$. $vt(x, V)$ accounts for the number of vertices in V with whom x closes at least one triangle, which are d_{in} plus d_{out} since we assume that every edge closes at least one triangle. Finally, the union between the vertices in C and those vertices in V with whom x closes at least one triangle is $r + d_{out}$. Therefore,

$$\begin{aligned}\Theta_3 &= \frac{WCC(v, C \cup \{v\})}{t(x, V)} \cdot \frac{vt(x, V)}{|C| + vt(x, V \setminus C)} = \\ &= \frac{d_{in}(d_{in}-1)\delta}{d_{in}(d_{in}-1)\delta + d_{out}(d_{out}-1)\omega + d_{out}d_{in}\omega} \cdot \frac{d_{in} + d_{out}}{r + d_{out}}. \quad \square\end{aligned}$$

E. REFERENCES

- [1] Y. Ahn, J. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.
- [2] J. P. Bagrow. Are communities just bottlenecks? trees and treelike networks have high modularity. *CoRR*, abs/1201.0745, 2012.
- [3] A.-L. Barabási and Z. N. Oltvai. Network biology: understanding the cell’s functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.
- [4] V. Blondel et al. Fast unfolding of communities in large networks. *JSTAT*, 2008:P10008, 2008.
- [5] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4):175–308, 2006.
- [6] A. Clauset et al. Finding community structure in very large networks. *Phy. Rev. E*, 70(6):066111, 2004.
- [7] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.
- [8] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *PNAS*, 104(1):36, 2007.
- [9] M. Girvan and M. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821, 2002.
- [10] A. Lancichinetti. Community detection algorithms: a comparative analysis. *Phy. Rev. E*, 80(5):056117, 2009.
- [11] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.
- [12] A. Lancichinetti, F. Radicchi, J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.
- [13] A. Medus et al. Detection of community structures in networks via global optimization. *Physica A*, 358(2-4):593–604, 2005.
- [14] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phy. Rev. E*, 69(2):026113, 2004.
- [15] G. K. Orman, V. Labatut, and H. Cherifi. Qualitative comparison of community detection algorithms. In *Digital Information and Communication Technology and Its Applications*, pages 265–279. Springer, 2011.
- [16] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.
- [17] P. Pons and M. Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.
- [18] 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.
- [19] 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.
- [20] M. Rosvall and C. Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4):1118, 2008.
- [21] Y. Wang, G. Cong, G. Song, and K. Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *SIGKDD*, pages 1039–1048. ACM, 2010.
- [22] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: the state of the art and comparative study. *ACM Computing Surveys*, 45(4), 2013.
- [23] J. Xie and B. Szymanski. Towards linear time overlapping community detection in social networks. In *PAKDD*, pages 25–36, 2012.
- [24] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*, pages 587–596, 2013.