DAOC: Stable Clustering of Large Networks

Artem Lutov
eXascale Infolab
University of Fribourg
Switzerland
artem.lutov@unifr.ch

Mourad Khayati
eXascale Infolab
University of Fribourg
Switzerland
mourad.khayati@unifr.ch

Philippe Cudré-Mauroux
eXascale Infolab
University of Fribourg
Switzerland
pcm@unifr.ch

***Abstract*—Clustering is a crucial component of many data mining systems involving the analysis and exploration of various data. Data diversity calls for clustering algorithms to be accurate while providing stable (i.e., deterministic and robust) results on arbitrary input networks. Moreover, modern systems often operate on large datasets, which implicitly constrains the complexity of the clustering algorithm. Existing clustering techniques are only partially stable, however, as they guarantee either determinism or robustness. To address this issue, we introduce DAOC, a Deterministic and Agglomerative Overlapping Clustering algorithm. DAOC leverages a new technique called Overlap Decomposition to identify fine-grained clusters in a deterministic way capturing multiple optima. In addition, it leverages a novel consensus approach, Mutual Maximal Gain, to ensure robustness and further improve the stability of the results while still being capable of identifying micro-scale clusters. Our empirical results on both synthetic and real-world networks show that DAOC yields stable clusters while being on average 25% more accurate than state-of-the-art deterministic algorithms without requiring any tuning. Our approach has the ambition to greatly simplify and speed up data analysis tasks involving iterative processing (need for determinism) as well as data fluctuations (need for robustness) and to provide accurate and reproducible results.**

***Index Terms*—stable clustering, deterministic overlapping clustering, community structure discovery, parameter-free community detection, cluster analysis.**

I. Introduction

Clustering is a fundamental part of data mining with a wide applicability to statistical analysis and exploration of physical, social, biological and information systems. Modeling and analyzing such systems often involves processing large complex networks [1]. Clustering large networks is intricate in practice, and should ideally provide *stable* results in an efficient way in order to make the process easier for the data scientist.

Stability is pivotal for many data mining tasks since it allows to better understand whether the results are caused by the evolving structure of the network, by evolving node ids (updated labels, coordinates shift or nodes reordering), or by some fluctuations in the application of non-deterministic algorithms. Stability of the results involves both determinism and robustness. We refer to the term *deterministic* in the strictest sense denoting algorithms that *a*) do not involve any stochastic operations and *b*) produce results invariant of the nodes processing order. *Robustness* ensures that clustering results gracefully evolve with small perturbations or changes in the input network [2]. It prevents sudden changes in the output for dynamic networks and provides the ability to tolerate noise and outliers in the input data [3].

Clustering a network is usually not a one-off project but an iterative process, where the results are visually explored and refined multiple times. The visual exploration of large networks requires to consider the specificities of human perception [4], [5] which is good at handling *fine-grained hierarchies* of clusters. In addition, those hierarchies should be stable across iterations such that the user can compare previous results with new results. This calls for results that are both stable and fine-grained.

In this paper, we introduce a novel clustering method called DAOC to address the aforementioned issues. To the best of our knowledge, DAOC[1] is the first parameter-free clustering algorithm that is simultaneously deterministic, robust and applicable to large weighted networks yielding a fine-grained hierarchy of overlapping clusters. More specifically, DAOC leverages *a*) a novel consensus technique we call *Mutual Maximal Gain (MMG)* to perform a robust and deterministic identification of node membership in the clusters, and *b*) a new technique for *overlap decomposition (OD)* to form fine-grained clusters in a deterministic way, even when the optimization function yields a set of structurally different but numerically equivalent optima (see *degeneracy* in Section III). We empirically evaluate the stability of the resulting clusters produced by our approach, as well as its efficiency and effectiveness on both synthetic and real-world networks. We show that DAOC yields stable clusters while being on average 25% more accurate than state-of-the-art deterministic clustering algorithms and more efficient than state-of-the-art overlapping clustering algorithms without requiring any manual tuning. In addition, we show that DAOC returns on average more accurate results than any state-of-the-art clustering algorithm on complex real-world networks (e.g., networks with overlapping and nested clusters). We foresee DAOC to represent an important step forward for clustering algorithms as: *a*) deterministic clustering algorithms are usually not robust and have a lower accuracy than their stochastic counterparts, and *b*) robust methods are typically not deterministic and do not provide fine-grained results as they are insensitive to micro-scale changes, as described in

[1] https://github.com/eXascaleInfolab/daoc

more detail in the following section.

II. Related Work

A great diversity of clustering algorithms can be found in the literature. Below, we give an overview of prior methods achieving robust results, before describing deterministic approaches and outlining a few widely used algorithms that are neither robust nor deterministic but were inspirational for our method.

a) Robust clustering algorithms: typically leverage *consensus* or *ensemble* techniques [6]–[9]. They identify clusters using consensus functions (e.g., majority voting) by processing an input network multiple times and varying either the parameters of the algorithm, or the clustering algorithm itself. However, such algorithms typically *a*) are unable to detect fine-grained structures due to the lack of consensus therein, *b*) are stochastic and *c*) are inapplicable to large networks due to their high computational cost. We describe some prominent and scalable consensus clustering algorithms below.

- *Order Statistics Local Optimization Method (OSLOM)* [10] is one of the first widely used consensus clustering algorithms, which accounts for weights of the network links and yields overlapping clusters with a hierarchical structure. It is based on the local optimization of a fitness function expressing the statistical significance of clusters with respect to random fluctuations. OSLOM scales near linearly on sparse networks but has a relatively high computational complexity at each iteration, making it inapplicable to large real-world networks (as we show in Section V).

- *Core Groups Graph Clustering Randomized Greedy (CGGC[i]_RG)* [11] is a fast and accurate ensemble clustering algorithm. It applies a generic procedure of ensemble learning called Core Groups Graph Clustering (CGGC) to determine several weak graph (network) clusterings and then to form a strong clustering from their maximal overlap. The algorithm has a near linear computational complexity with the number of edges due to the sampling and local optimization strategies applied at each iteration. However, this algorithm is designed for unweighted graphs and produces flat and non-overlapping clusters only, which limits its applicability and yields low accuracy on large complex networks as we show in Section V.

- *Fast Consensus* technique was recently proposed and works on top of state-of-the-art clustering algorithms including Louvain (FCoLouv), Label Propagation (FCoLPM) and Infomap (FCoIMap) [12]. The technique initializes a consensus matrix and then iteratively refines it until convergence as follows. First, the input network is clustered by the original algorithm multiple times. The consensus values $D_{i,j} \in [0, 1]$ of the matrix are evaluated as the fraction of the runs in which nodes i and j belong to the same cluster. The consensus matrix is formed using pairs of co-clustered adjacent nodes and extended with closed triads instead of all nodes in the produced clusters, which significantly reduces the amount of computation. The formed matrix is filtered with a threshold τ and then clustered n_p times by the original clustering algorithm, producing a refined consensus matrix. This refinement process is repeated until all runs produce identical clusters (i.e.,until all values in the consensus matrix are either zero and one) with precision $1 - \delta$. The Fast Consensus technique however lacks a convergence guarantee and relies on three parameters having a strong impact on its computational complexity.

b) Deterministic clustering algorithms: and, in general, non-stochastic ones (i.e., algorithms relaxing the determinism constraint) are typically not robust and are sensitive to both *a*) initialization [13]–[16] (including the order in which the nodes are processed) and *b*) minor changes in the input network, which may significantly affect the clustering results [3], [8]. Non-stochastic algorithms also often yield less precise results getting stuck on the same local optimum until the input is updated. Multiple local optima often exist due to the *degeneracy* phenomenon, which is explained in Section III and has to be specifically addressed to create deterministic clustering algorithms that are both robust and accurate. We describe below some of the well-known deterministic algorithms.

- *Clique Percolation method (CPM)* [17] is probably the first deterministic clustering algorithm supporting overlapping clusters and capable of providing fine-grained results. *Sequential algorithm for fast clique percolation (SCP)* [18] is a CPM-based algorithm, which detects k-clique clusters in a single run and produces a dendrogram of clusters. SCP produces deterministic and overlapping clusters at various scales, and shows a linear dependency of the computational complexity with the number of k-cliques in the network. However, SCP relies on a number of parameters and has an exponential worst case complexity in dense networks, which significantly limits its practical applicability.

- *pSCAN* [19] is a fast overlapping clustering algorithm for "exact structural graph clustering" (i.e., it is deterministic and input-order independent). First, it identifies core graph vertices (network nodes), which always belong to exactly one cluster, forming initially disjoint clusters. The remaining nodes are then assigned to the initial clusters, yielding overlapping clusters. pSCAN relies on two input parameters, $0 < \epsilon \leq 1$ and $\mu \geq 2$. The results it produces are very sensitive to those parameters, whose optimal values are hard to guess for arbitrary input networks.

c) Inspirational algorithms for our method:

- *Louvain* [20] is a commonly used clustering algorithm that performs modularity optimization using a local search technique on multiple levels to coarsen clusters. It introduces modularity gain as an optimization function. The algorithm is parameter-free, returns a hierarchy of clusters, and has a near-linear runtime complexity with the number of network links. However, the resulting clusters are not stable and depend on the order in which the nodes are processed. Similarly to Louvain, our method is a greedy agglomerative clustering algorithm, which uses modularity gain as optimization function. However, the clusters formation process in DAOC differs a lot, addressing the aforementioned issues of the Louvain algorithm.

- *DBSCAN* [21] is a density-based clustering algorithms suitable to process data with noise. It regroups points that are close in space given the maximal distance between the points ε and the minimal number of points $MinPts$ within an area.

DBSCAN is limited in discovering a large variety of clusters because of its reliance to a density parameter. It has a strong dependency on input parameters, and lacks a principled way to determine optimal values for these parameters [22]. We adopt however the DBSCAN idea of clusters formation based on the extension of the densest region to prevent early coarsening and to produce a fine-grained hierarchical structure.

III. Preliminaries

A clustering algorithm is applied to a network to produce groups of nodes that are called *clusters* (also known as communities, modules, partitions or covers). Clusters represent groups of tightly-coupled nodes with loosely inter-group connections [23], where the group structure is defined by the clustering optimization function. The resulting clusters can be overlapping, which happens in case they share some common nodes, the *overlap*. The input network (graph) can be weighted and directed, where a node (vertex) weight is represented as a weighted link (edge) to the node itself (a self-loop). The main notations used in this paper are listed in Table I.

Clustering algorithms can be classified by the kind of input data they operate on: *a*) graphs specified by pairwise relations (networks) or *b*) attributed graphs (e.g., vertices specified by coordinates in a multi-dimensional space). These two types of input data cannot be unambiguously converted into each other, at least unless one agrees on some customized and specific conversion function. Hence, their respective clustering algorithms are not (directly) comparable. In this paper, we focus on clustering algorithms working on graphs specified by pairwise relations (networks), which are also known as community structure discovery algorithms.

a) Modularity (Q) [24]: is a standard measure of clustering quality that is equal to the difference between the density of the links in the clusters and the expected density:

$$Q = \frac{1}{2w} \sum_{i,j} \left(w_{i,j} - \frac{w_i w_j}{2w} \right) \delta(C_i, C_j) \tag{1}$$

where $w_{i,j}$ is the accumulated weight of the arcs between nodes `#i` and `#j`, w_i is the accumulated weight of all arcs of `#i`, w is the total weight of the network, C_i is the cluster to which `#i` is assigned, and Kronecker delta $\delta(C_i, C_j)$ is a function, which is equal to 1 when `#i` and `#j` belong to the same cluster (i.e., $C_i = C_j$), and 0 otherwise.

Modularity is applicable to non-overlapping cases only. However, there exist modularity extensions that handle overlaps [25], [26]. The main intuition behind such modularity extensions is to quantify the degree of a node membership among multiple clusters either by replacing a Kronecker δ (see (1)) with a similarity measure s_{ij} [27], [28] or by integrating a belonging coefficient [29]–[31] directly into the definition of modularity. Although both old and new measures are named modularity, they generally have different values even when applied to the same clusters [32], resulting in incompatible outcomes. Some modularity extensions are equivalent to the original modularity when applied to non-overlapping clusters [27], [28], [31]. However, the implementations of these extensions introduce an excessive number of additional parameters [27], [28] and/or boost the computational time by orders of magnitude [31], which significantly complicates their application to large networks.

Modularity gain (ΔQ) [20] captures the difference in modularity when merging two nodes `#i` and `#j` into the same cluster, providing a computationally efficient way to optimize Modularity:

$$\Delta Q_{i,j} = \frac{1}{2w} \left(w_{i,j} - \frac{w_i w_j}{w} \right) \tag{2}$$

We use *modularity gain* (ΔQ) as an underlying optimization function for our meta-optimization function MMG (introduced in Section IV-A1).

b) Degeneracy: is a phenomenon linked to the clustering optimization function appearing when multiple distinct clusterings (i.e., results of the clustering process) share the same globally maximal value of the optimization function while being structurally different [33]. This phenomenon is inherent to any optimization function and implies that a network node might yield the maximal value of the optimization function while being a member of multiple clusters, which is the case when an *overlap* occurs. This prevents the derivation of accurate results by deterministic clustering algorithms without considering overlaps. To cope with degeneracy, typically multiple stochastic clusterings are produced and combined, which is called an *ensemble* or *consensus* clustering and provides robust but coarse-grained results [33], [34]. Degeneracy of the optimization function, together with the aforementioned computational drawback of modularity extensions, motivated us to introduce a new overlap decomposition technique, OD (see Section IV-B1). OD allows to consider and process overlaps efficiently using algorithms having an optimization function designed for the non-overlapping case. It produces accurate, robust and fined-grained results in a deterministic way as we show in our experimental evaluation (see Section V).

TABLE I
Notations

`#i`	Node <i> of the network (graph) $\mathcal{G}$
Q	*Modularity*
$\Delta Q_{i,j}$	*Modularity Gain* between `#i` and `#j`
$j \sim i$	Items i and j (nodes or clusters) are neighbors (adjacent, linked)
ΔQ_i	Maximal *Modularity Gain* for `#i`: $\Delta Q_i = \{\max \Delta Q_{i,j} \mid \texttt{\#j} \sim \texttt{\#i}\}$
$MMG_{i,j}$	*Mutual Maximal Gain*: $MMG_{i,j} = \{\Delta Q_{i,j} \mid \Delta Q_i = \Delta Q_j,\ \texttt{\#j} \sim \texttt{\#i}\}$
ccs_i	Mutual clustering candidates of `#i` (by MMG_i)

IV. METHOD

We introduce a novel clustering algorithm, DAOC, to perform a stable (i.e., both *robust* and *deterministic*) clustering of the input network, producing a fine-grained hierarchy of overlapping clusters. DAOC is a greedy algorithm that uses an agglomerative clustering approach with a local search technique (inspired by Louvain [20]) and extended with two novel techniques. Namely, we first propose a novel (micro) consensus technique called *Mutual Maximal Gain (MMG)* for the robust identification of nodes membership in the clusters, which is performed in a deterministic and fine-grained manner. In addition to MMG, we also propose a new *overlap decomposition (OD) technique* to cope with the degeneracy of the optimization function. OD forms stable and fine-grained clusters in a deterministic way from the nodes preselected by MMG.

Algorithm 1 gives a high-level description of our method. It takes as input a directed and weighted network with self-loops specifying node weighs. The resulting hierarchy of clusters is built iteratively starting from the bottom level (the most fine-grained level). One level of the hierarchy is generated at each iteration of our clustering algorithm. A clustering iteration consists of the following steps listed on lines 4–5:

1) Identification of the clustering candidates ccs_i for each node `#i` using the proposed consensus approach, MMG, described in Section IV-A2 and
2) Cluster formation considering overlaps, described in Section IV-B.

Algorithm 1 DAOC Clustering Algorithm.

```
1: function CLUSTER(nodes)
2:     hier ← []                                  ▷ List of the hierarchy levels
3:     while nodes do                             ▷ Stop if the nodes list is empty
4:         identifyCands(nodes)                   ▷ Initialize nd.ccs
5:         cls ← formClusters(nodes)
6:         if cls then                            ▷ Initialize the next-level nodes
7:             for all cl ∈ cls do
8:                 initCluster(cl)
9:             end for
10:            for all nd ∈ nodes do              ▷ Consider propagates
11:                if nd.propagated then
12:                    initNode(nd)
13:                    cls.append(nd)
14:                end if
15:            end for
16:            hier.append(cls)                   ▷ Extend the hierarchy
17:        end if
18:        nodes ← cls                            ▷ Update the processing nodes
19:    end while
20:    return hier                                ▷ The resulting hierarchy of clusters
21: end function
```

At the end of each iteration, links are (re)computed for the formed clusters (`initCluster` procedure) and for the non-clustered nodes (*propagated* nodes in the `initNode` procedure). Both the non-clustered nodes and the formed clusters are treated as input nodes for the following iteration. The algorithm terminates when the iteration does not produce any new cluster.

The clustering process yields a hierarchy of overlapping clusters in a deterministic way independent of the nodes processing order, since all clustering operations *a*) consist solely of non-stochastic, uniform and local operations, and *b*) process each node independently, relying on immutable data evaluated on previous steps only. The algorithm is guaranteed to converge since *a*) the optimization function is bounded (as outlined in Section IV-A1) and monotonically increasing during the clustering process, and *b*) the number of formed clusters does not exceed the number of clustered nodes at each iteration (as explained in Section IV-B2).

A. Identification of the Clustering Candidates

The *clustering candidates* are the nodes that are likely to be grouped into clusters in the current iteration. The clustering candidates are identified for each node ($nd.ccs$) in two steps as listed in Algorithm 2. First, for each node nd the adjacent nodes ($\{link.dst \mid link \in nd.links\}$) having the maximal non-negative value $nd.gmax$ of the optimization function `optfn` are stored in the $nd.ccs$ sequence, see lines 3–11. Then, the preselected $nd.ccs$ are reduced to the mutual candidates by the `mcands` procedure, and the filtered out nodes are marked as propagated. The latter step is combined with a cluster formation operation in our implementation to avoid redundant passes over all nodes. The `mcands` procedure implements our *Maximal Mutual Gain (MMG)* consensus approach described in Section IV-A2, which is a meta-optimization technique that can be applied on top of any optimization function that satisfies a set of constraints described in the following paragraph.

Algorithm 2 Clustering Candidates Identification

```
1: function IDENTIFYCANDS(nodes)
2:     for all nd ∈ nodes do                      ▷ Evaluate clustering candidates
3:         nd.gmax ← −1                           ▷ Maximal gain
4:         for all ln ∈ nd.links do
5:             cgain ← optfn(nd, ln)              ▷ Clustering gain
6:             if cgain ≥ 0 and cgain ≥ nd.gmax then
7:                 if cgain > nd.gmax then
8:                     nd.ccs.clear()             ▷ Reset cands
9:                     nd.gmax ← cgain
10:                end if
11:                nd.ccs.append(ln.dst)          ▷ Extend cands
12:            end if
13:        end for
14:    end for

15:    for all nd ∈ nodes do   ▷ Reduce the candidates using the consensus approach, propagate remained nodes
16:        if nd.gmax < 0 or not mcands(nd) then
17:            nd.propagated ← true
18:        end if
19:    end for
20: end function
```

1) Optimization Function: In order to be used in our method, the optimization function `optfn` should be *a*) applicable to pairwise node comparison, i.e. $\exists\,$`optfn(#i, #j)` $\mid$ `#i` $\sim$ `#j` (adjusted pair of nodes); *b*) commutative, i.e.

`optfn(#i,#j) = optfn(#j,#i)`; and *c*) bounded on the non-negative range, where positive values indicate some quality improvement in the structure of the forming cluster. There exist various optimization functions satisfying these constraints besides modularity and inverse conductance (see the list in [35], for instance).

Our DAOC algorithm uses *modularity gain*, ΔQ (see (2)), as an optimization function. We chose modularity (gain) optimization because of the following advantages. First, modularity maximization (under certain conditions) is equivalent to the provably correct but computationally expensive methods of graph partitioning, spectral clustering and to the maximum likelihood method applied to the stochastic block model [36], [37]. Second, there are known and efficient algorithms for modularity maximization, including the Louvain algorithm [20], which are accurate and have a near-linear computational complexity.

2) MMG Consensus Approach: We propose a novel (micro) consensus approach, called *Mutual Maximal Gain (MMG)* that requires only a single pass over the input network, is more efficient and yields much more fine-grained results compared to state-of-the-art techniques.

Definition 1 (Mutual Maximal Gain (MMG)): MMG is a value of the optimization function (in our case modularity gain) for two adjacent nodes `#i` and `#j`, and is defined in cases where these nodes mutually reach the maximal value of the optimization function (i.e., reach consensus on the maximal value) when considering each other:

$$MMG_{i,j} = \{\Delta Q_{i,j} \mid \Delta Q_i = \Delta Q_j,\ \texttt{\#j} \sim \texttt{\#i}\} \qquad (3)$$

where $\sim$ denotes adjacency of `#i` and `#j`, and ΔQ_i is the maximal modularity gain for `#i`:

$$\Delta Q_i = \{\max \Delta Q_{i,j} \mid \texttt{\#j} \sim \texttt{\#i}\} \qquad (4)$$

where $\Delta Q_{i,j}$ is the modularity gain for `#i` and `#j` (see (2)).

MMG exists in any finite network, which can be easily proven by contradiction as follows. The nonexistence of MMG would create a cycle with increasing $\max \Delta Q$ and results in $\Delta Q_i < \Delta Q_i$ considering that $\forall$`#i` $\exists \Delta Q_i$: $(\Delta Q_i = \Delta Q_{i,j}) < (\Delta Q_j = \Delta Q_{j,k}) < ... < (\Delta Q_t = \Delta Q_{t,j} = \Delta Q_j)$, i.e. $\Delta Q_j < \Delta Q_j$, which yields a contradiction. MMG evaluation is deterministic and the resulting nodes are *quasi-uniform* clustering candidates, in the sense that inside each connected component they share the same maximal value of modularity gain. MMG takes into account fine-grained clusters as it operates on pairs of nodes, unlike conventional consensus approaches, where micro-clusters either require lots of reexecutions of the consensus algorithm, or cannot be captured at all. MMG does not always guarantee optimal clustering results but reduces degeneracy due to the applied consensus approach. According to (3), all nodes having MMG to `#i` have the same value of ΔQ_i, i.e., form the overlap in `#i`. Overlaps processing is discussed in the following section.

B. Clusters Formation with Overlap Decomposition

Clusters are formed from candidate nodes selected by MMG as listed in Algorithm 3: *a*) nodes having a single mutual clustering candidate (*cc*) form the respective cluster directly as shown on line 8, *b*) otherwise the overlap is processed. There are three possible ways of clustering an overlapping node in a deterministic way: *a*) split the node into *fragments* to have one fragment per each *cc* of the node and group each resulting fragment with the respective *cc* into the dedicated cluster (see lines 10–11), or *b*) group the node together with all its *nd.ccs* items into a single cluster (i.e. coarsening on line 18), or *c*) propagate the node to be processed on the following clustering iteration if its clustering would yield a negative value of the optimization function. Each node fragment created by the overlap decomposition is treated as a virtual node representing the belonging degree (i.e., the fuzzy relation) of the original node to multiple clusters. Virtual nodes are used to avoid the introduction of the fuzzy relation for all network nodes (i.e., to avoid an additional complex node attribute) reducing memory consumption and execution time, and not affecting the input network itself. In order to get the most effective clustering result, we evaluate the first two aforementioned options and select the one maximizing the optimization function, ΔQ. Then, we form the cluster(s) by the `merge` or `mergeOvp` procedures as follows. The `mergeOvp` procedure groups each node fragment (i.e., the virtual node created by the overlap decomposition) together with its respective mutual clustering candidate. This results in either *a*) an extension of the existing cluster to which the candidate already belongs to, or *b*) the creation of a new cluster and its addition to the *cls* list. The `merge` procedure groups the node with all its clustering candidates either *a*) merging together the respective clusters of the candidates if they exists, or *b*) creating a new cluster and adding it to the *cls* list.

Algorithm 3 Clusters Formation

```
1: function FORMCLUSTERS(nodes)
2:     cls ← []                                   ▷ List of the formed clusters
3:     for all nd ∈ nodes do
4:         if nd.propagated then                  ▷ Prefileter nodes
5:             continue
6:         end if
7:         if nd.ccs.size = 1 then                ▷ Form a cluster
8:             merge(cls, nd, nd.ccs)
9:         else if odAccept(nd) and gainEach(nd) > gainAll(nd) then   ▷ Form overlapping clusters
10:            for all cand ∈ nd.ccs do
11:                mergeOvp(cls, nd, cand)
12:            end for
13:        else                                   ▷ DBSCAN inspired aggregation
14:            rccs ← maxIntersectOrig(nd)
15:            if rccs.size ≥ 1 then              ▷ Form a single cluster
16:                merge(cls, nd, rccs)
17:            else if gainAll(nd) ≥ 0 then       ▷ Form a cluster
18:                merge(cls, nd, nd.ccs)
19:            else
20:                nd.propagated ← true
21:            end if
22:        end if
23:    end for
24:    return cls                                 ▷ Resulting clusters
25: end function
```

Node splitting is the most challenging process, which is performed only if the accumulated gain from the decomposed node fragments to each of the respective mutual clustering candidates, *nd.ccs*, (`gainEach` procedure) exceeds the gain of grouping the whole node with all *nd.ccs* (`gainAll` procedure). The node splitting involves: *a*) the estimation of the node fragmentation impact on the clustering convergence (`odAccept` procedure given in Section IV-B2) and *b*) the evaluation of the weights for both the forming fragment and for the links between the fragments of the splitting node as described in Section IV-B1.

1) Overlap Decomposition (OD): An overlap occurs when a node has multiple mutual clustering candidates (*ccs*). To evaluate ΔQ when clustering the node with each of its K *ccs*, the node is split into K identical and fully interconnected fragments sharing the node weight and original node links. However, since the objective of the clustering is the maximization of ΔQ: *a*) the node splitting itself is acceptable only in case the resulting $\Delta Q \geq 0$, and *b*) the decomposed node can be composed back from the fragments only in case $\Delta Q \leq 0$. Hence, to have a reversible decomposition without affecting the value of the optimization function for the decomposing node, we end up with $\Delta Q = 0$.

The outlined constraints for an isolated node, which does not have any link to other nodes, can formally be expressed as:

$$\begin{cases} w = \sum_{k=1}^{K} w_k + \sum_{k=1}^{K} \sum_{t=1}^{K-1} \frac{w_{k,t}}{2} \\ \sum_{k=1}^{K} w_k - \sum_{k=1}^{K} \frac{(w_k + \sum_{t=1}^{K-1} \frac{w_{k,t}}{2})^2}{w} = 0 \end{cases}, \quad (5)$$

where w is the weight of the original node being decomposed into K fragments, w_k is the weight of each node fragment $k \in K$ and $w_{k,t}$ is the weight of each link between the fragments. $\Delta Q = Q_{split} - Q_{node} = Q_{split}$ since the modularity of the isolated node is zero (see (1)). The solution of (5) is:

$$w_k = \frac{w}{K^2},\ w_{k,t} = 2\frac{w}{K^2}. \quad (6)$$

Nodes in the network typically have links, which get split equally between the fragments of the node:

$$\forall\, k,t \in \{1\,..\,K\} \mid \#j \sim \#i : \begin{cases} w_{ik} = \frac{w_i}{K^2} \\ w_{ik,it} = 2\frac{w_i}{K^2} \\ w_{ik,j} = \frac{w_{i,j}}{K} \end{cases}, \quad (7)$$

where w_{ik} is the weight of each fragment `#ik` of the node `#i`, $w_{i,j}$ is the weight of the link $\{$`#i`, `#j`$\}$.

Example 1 (Overlap Decomposition): The input network on the left-hand side of Fig. 1 has node C with neighbor nodes $\{A, B, D\}$ being *ccs* of C. These neighbor nodes form the respective clusters $\{C1, C2, C3\}$ overlapping in C. C is decomposed into fragments $\{C_A, C_B, C_D\}$ to evaluate the overlap. Node C has an internal weight equal to 9 (which can be represented via an additional edge to itself) and three edges of weight 12 each. The overlapping clusters are evaluated using (7) as equivalent and virtual non-overlapping clusters formed using the new fragments of the overlapping node:

$$\forall\, k,t \in \{1\,..\,K\} \mid \#j \sim \#i : \begin{cases} w_{ik} = \frac{9}{3^2} = 1 \\ w_{ik,it} = 2\frac{9}{3^2} = 2 \\ w_{ik,j} = 2 + \frac{12}{3} = 6 \quad (\forall j \neq k) \\ w_{ik,k} = \frac{12}{3} = 4 \end{cases}$$

$$w_{C_A} = w_{C_B} = w_{C_D} = w_{ik} = 1,$$
$$w_{C1} = w_{C2} = w_{C3} = w_k + w_{ik} + w_{ik,k} = 10 + 1 + 4 = 15,$$
$$w_{C_k,C_t} = w_{ik,t} + w_{it,k} - w_{ik,it} = 6 + 6 - 2 = 10.$$

2) Constraining Overlap Decomposition: Overlap decomposition (OD) does not affect the value of the optimization function for the node being decomposed ($\Delta Q = 0$), hence it does not affect the convergence of the optimization function during the clustering. However, OD increases the complexity of the clustering when the number of produced clusters exceeds the number of clustered nodes decomposed into multiple fragments. This complexity increase should be identified and prevented to avoid indefinite increases in terms of clustering time.

In what follows, we infer a formal condition that guarantees the non-increasing complexity of OD. We decompose a node of degree d into k fragments, $2 \leq k \leq d$. Each forming cluster that has an overlap in this node owns one fragment (see Fig. 1) and shares at most $d-k$ links to the non-*ccs* neighbors of the node. The number of links between the k fragments resulting in the node split is $k \times \frac{k-1}{2}$. The aggregated number of resulting links should not exceed the degree of the node being decomposed to retain the same network complexity, therefore:

$$\begin{cases} k(d-k) + k\frac{k-1}{2} \leq d \\ 2 \leq k \leq d \end{cases} \quad (8)$$

The solution of (8) is $2 \leq k \leq d \leq 3$, namely: $k = 2$, $d = \{2, 3\}$; $k = 3$, $d = 3$.

If a node being decomposed has a degree $d >= 3$ or a node has more than k *ccs* then, before falling back to the coarse cluster formation, we apply the following heuristic inspired by the DBSCAN algorithm [21]. We evaluate the

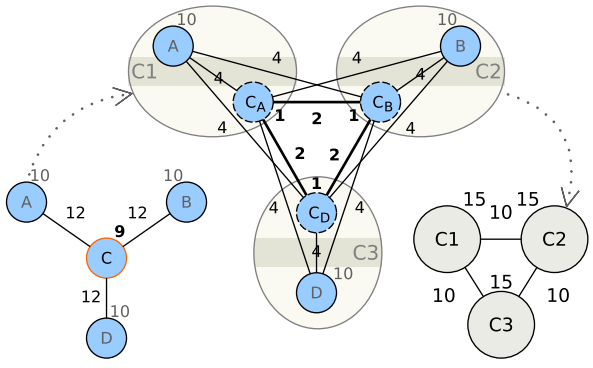


Fig. 1. Decomposition of the clusters $C1(A, C)$, $C2(B, C)$, $C3(D, C)$ overlapping in node C of the input network $\{A, B, C, D\}$ with weights on both nodes and links.

intersection of *nd.ccs* with each $\{c.ccs \mid c \in nd.ccs\}$ (`maxIntersectOrig` procedure on line 14 of Algorithm 3) and group the node with its clustering candidate(s) yielding the maximal intersection if the latter contains at least half of the *nd.ccs*. In such a way, we try prevent an early coarsening and obtain more fine-grained and accurate results.

C. Complexity Analysis

The computational complexity of DAOC on sparse networks is $O(m \cdot \log m)$, where *m* is the number of links in the network. All links of each node ($d \cdot n = m$) are processed for $\log m$ iterations. In the worst case, the number of iterations is equal to the number of nodes *n* (instead of $\log m$) and the number of mutual candidates is equal to the node degree *d* instead of *1*. Thus, the theoretical worst-case complexity is $O(m \cdot d \cdot n) = O(m^2)$ and occurs only in a hypothetical dense symmetric network having equal MMG for all links (and, hence, requiring overlap decomposition) on each clustering iteration and in case the number of clusters is decreased at each iteration by one only. The memory complexity is $O(m)$.

V. Experimental Evaluation

A. Evaluation Environment

Our evaluation was performed using an open-source parallel isolation benchmarking framework, Clubmark[2] [38], on a Linux Ubuntu 16.04.3 LTS server with the Intel Xeon CPU E5-2620 v4 @ 2.10GHz CPU (16 physical cores) and 132 GB RAM. The execution termination constraints for each algorithm are as follows: 64 GB of RAM and 72 hours max per network clustering. Each algorithm is executed on a single dedicated physical CPU core with up to 64 GB of guaranteed available physical memory.

We compare DAOC against almost a dozen state-of-the-art clustering algorithms listed in Table II (the original implementations of all algorithms except Louvain are included into Clubmark and are executed as precompiled or JIT-compiled applications or libraries) and described in the following papers: SCP [18], Lvn(Louvain[3] [20]), Fcl (Fast Consensus on Louvain: FCoLouv [12]), Osl2(OSLOM2 [10]), Gnx(GANXiS also known as SLPA [39]), Psc(pSCAN [19]), Cgr[i](CGGC[i]_RG [11]), SCD [40] and Rnd(Randcommuns [38]). We have not evaluated a well known CPM-based overlapping clustering algorithm, CFinder [41], because *a*) GANXiS outperforms CFinder in all aspects by several accuracy metrics [39], [42] and *b*) we do evaluate SCP, a fast CPM-based algorithm. For a fair accuracy evaluation, we uniformly sample up to 10 levels from the clustering results (levels of the hierarchical / multilevel output or clusterings produced uniformly varying algorithm parameters in the operational range) and take the best value.

[2]https://github.com/eXascaleInfolab/clubmark

[3]http://igraph.org/c/doc/igraph-Community.html

B. Stability Evaluation

We evaluate stability in terms of both robustness and determinism for the consensus (ensemble) and deterministic clustering algorithms listed in Table II. Determinism (non-stochasticity and input order independence) evaluation is performed on synthetic and real-world networks below, where we quantify the standard deviation of the clustering accuracy. To evaluate stability in terms of robustness, we quantify the deviation of the clustering accuracy in response to small perturbations of the input network. The clustering accuracy on each iteration is measured relative to the clustering yielded by the same algorithm at the previous perturbation iteration. For each clustering algorithm, the accuracy is evaluated only for the middle level (scale or hierarchical level), since it is crucial to take the same clustering scale to quantify structural changes in the forming clusters of evolving networks. Robust clustering algorithms are expected to have their accuracy gradually evolving (without any *surges*) relative to the previous perturbation iteration. In addition, the clustering algorithms sensitive enough to capture the structural changes are expected to have their accuracy *monotonically decreasing* since the relative network reduction (perturbation) is increased at each iteration: X deleted links on iteration i represent a fraction of X/N_i, but on the following iteration this fraction is increased to $X/(N_i - X)$.

We evaluate robustness and sensitivity (i.e., the ability to capture small structural changes) on synthetic networks with

TABLE II
Evaluating clustering algorithms.

Features \ Algs	***Daoc***	**Scp**	**Lvn**	**Fcl**	**Osl2**	**Gnx**	**Psc**	**Cgr**	**Cgri**	**Scd**	**Rnd**
Hierarchical	+		+		+						
Multi-scale	+	+	+		+	+					
Deterministic	+	+					+				
Overlapping clusters	+	+			+	+	+				
Weighted links	+	+	+	○	+	+					+
Parameter-free	+!		+	*	*	*		*	*	*	+
Consensus/Ensemble	+			+	+			+	+		

Deterministic includes input-order invariance;
+! the feature is available, still the ability to force custom parameters is provided;
* the feature is partially available, parameters tuning might be required for specific cases;
○ the feature is available in theory but is not supported by the original implementation of the algorithm.

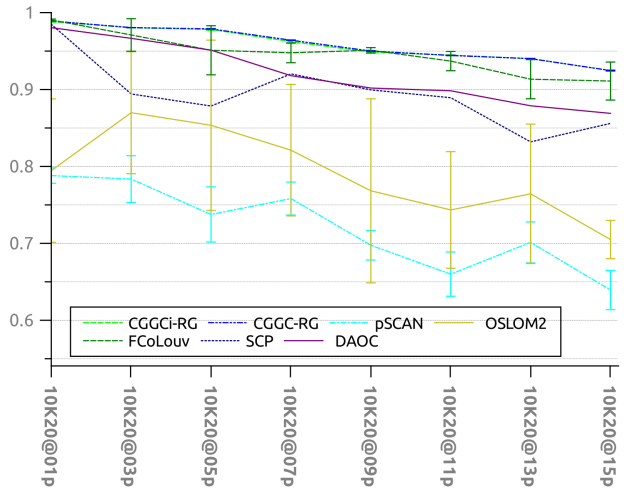
nodes forming overlapping clusters generated by the LFR framework [43]. We generate a synthetic network with ten thousand nodes having an average degree of 20 and using the mixing parameter $\mu = 0.275$. This network is shuffled (the links and nodes are reordered) 4 times to evaluate the input order dependence of the algorithms. Small perturbations of the input data are performed gradually reducing the number of links in the network by 2% of the original network size (i.e., $10 \times 1000 \times 20 \times 0.02 = 4000$ links) starting from 1% and ending at 15%. The links removal is performed *a*) randomly to retain the original distributions of the network links and their weights but *b*) respecting the constraint that each node retains at least a single link. This constraint prevents the formation of disconnected regions. Our perturbation does not include any random modification of the link weights or the creation of new links since it would affect the original distributions of the network links and their weights, causing surges in the clustering results.

The evaluations of stability in terms of robustness (absence of surges in response to small perturbation of the input network) and sensitivity (ability to capture small structural changes) are shown in Fig. 2. Absolute accuracy values relative to the previous link reduction iteration are shown in Fig. 2(a). The results demonstrate that, as expected, all deterministic clustering algorithms except DAOC (i.e. pSCAN and SCP) result in surges and hence are not robust. We also obtain some unexpected results. First, pSCAN, which is nominally "exact" (i.e., non-stochastic and input-order independent), actually shows significant deviations in accuracy. Second, the clusterings produced by OSLOM2 using default parameters and by FCoLouv using a number of consensus partitions $n_p = 5$ are prone to surges. Hence, OSLOM2 and FCoLouv cannot be classified as robust algorithms according to the obtained results in spite of being a consensus clustering algorithms. Fig. 2(b) illustrates the sensitivity of the algorithms, where the relative accuracy values compared to the previous perturbation iteration are shown. Sensitive algorithms have monotonically decreasing results for the subsequent link reduction, which corresponds to positive values on this plot. The stable algorithms (CGGC-RG, CGGCi-RG and DAOC) are highlighted with a bolder line width on the figure. These results demonstrate that being robust, CGGC-RG and CGGCi-RG are not always able to capture structural changes in the network, i.e., they are less sensitive than DAOC. Overall, the results show that only our algorithm, DAOC, is stable (it is deterministic, including input-order independence, and robust) and at the same time is able to capture even small structural changes in the input network.

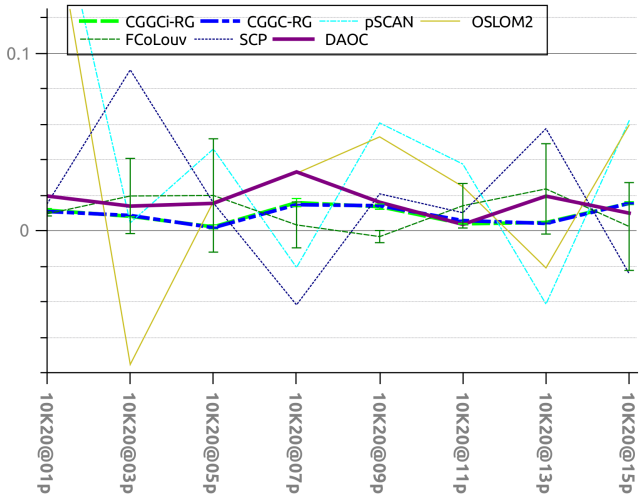
C. Effectiveness and Efficiency Evaluation

Our performance evaluation was performed both *a*) on weighted undirected synthetic networks with overlapping ground-truth clusters produced by the LFR frameworkintegrated into Clubmark [38] and *b*) on large real-world networks having overlapping and nested ground-truth communities[4] [44]. The synthetic networks were generated with 1, 5, 20 and 50 thousands nodes, each having an average node degrees of 5, 25 and 75. The maximal node degree is uniformly scaled from 70 on the smallest networks up to 800 on the largest ones. Synthetic networks generation parameters are taken by default as provided by Clubmark. The real-world networks contain from 334,863 nodes with 925,872 links (amazon) up to 3,997,962 nodes with 34,681,189 links (livejournal). The ground-truth communities of real-world networks were preprocessed to exclude duplicated clusters (communities having exactly the same nodes). Each network is shuffled (reordered) 4 times and the average accuracy value along with its standard deviation are reported.

A number of accuracy measures exist for overlapping clusters evaluation. We are aware of only two families of accuracy measures applicable to large overlapping clusterings, i.e. hav-



(a) F1h (average value and deviation) for subsequent perturbations (link removals). Stable algorithms are expected to have a gracefully decreasing F1h without any surges.



(b) ΔF1h relative to the previous perturbation iteration. Stable and sensitive algorithms are highlighted with bolder line width and have positive ΔF1h evolving without surges. Standard deviation is shown only for the consensus algorithms but visible only for FCoLouv and CGGCi-RG.

Fig. 2. Stability and sensitivity evaluation.

[4]https://snap.stanford.edu/data/\#communities

TABLE III
PEAK MEMORY CONSUMPTION (RSS) ON THE REAL-WORLD NETWORKS, MB.

Nets\Algs	*Daoc*	Scp*	Lvn	Fcl	Osl2	Gnx	Psc	Cgr	Cgri	Scd	Rnd
amazon	238	3,237	339	3,177	681	3,005	155	247	1,055	*37*	337
dblp	225	3,909	373	3,435	717	2,879	167	247	1,394	*36*	373
youtube	737	4,815	1,052	–	–	8,350	508	830	3,865	*131*	1,050
livejournal	5,038	–	10,939	–	–	–	4,496	4,899	11,037	*761*	–

– denotes that the algorithm was terminated for violating the execution constraints;
* the memory consumption and execution time for SCP are reported for a clique size $k = 3$ since they grow exponentially with k on dense networks, though accuracy was evaluated varying $k \in 3..7$.

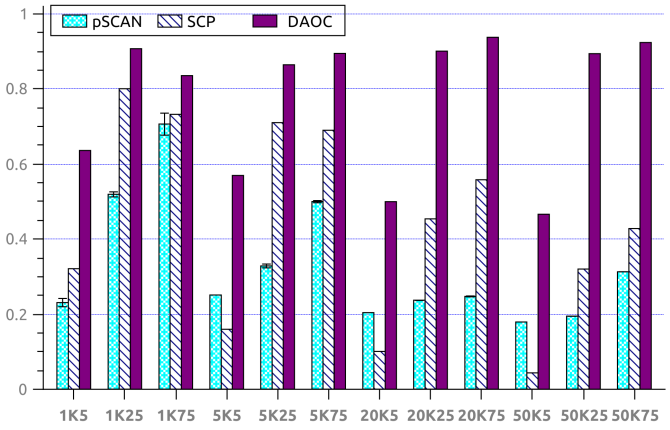
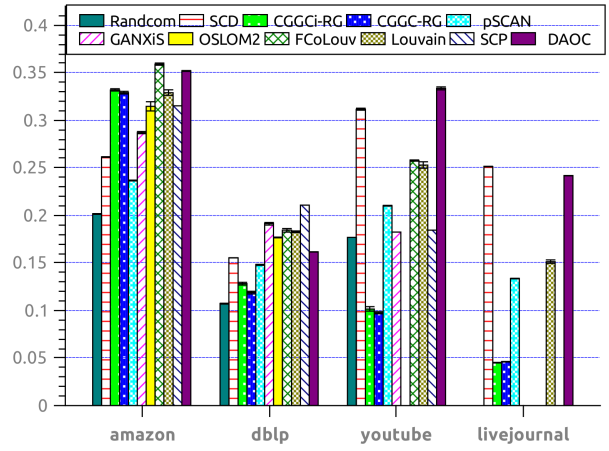
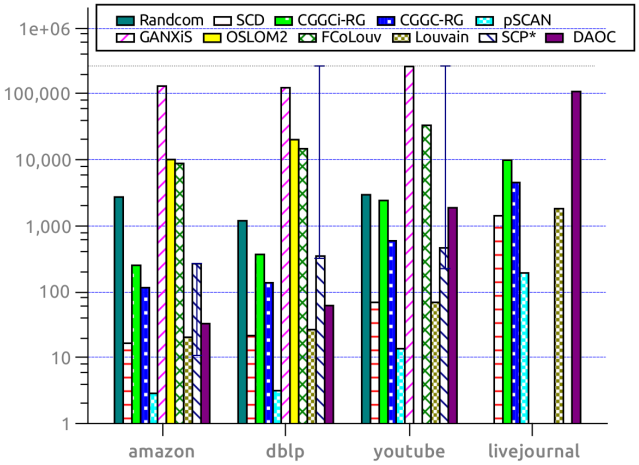


Fig. 3. F1h of the deterministic algorithms on the synthetic networks.



(a) F1h (average value and deviation).



(b) Execution time for a single algorithm run on a single and dedicated CPU core, sec. The range in SCP shows the execution time for $k > 3$.

Fig. 4. Performance on the real-world networks.

ing a near-linear computational complexity with the number of nodes: the F1 family [45] and generalized NMI (GNMI) [45], [46]. However, mutual information-based measures are biased to a large numbers of clusters while GNMI does not have any bounded computational complexity in general. Therefore, we evaluate clustering accuracy with *F1h* [45], a modification of the popular *average F1-score (F1a)* [40], [47] providing indicative values in the range $[0, 0.5]$, since the artificial clusters formed from all combinations of the input nodes yield $F1a \to 0.5$ and $F1h \to 0$.

First, we evaluate accuracy for all the deterministic algorithms listed in Table II on synthetic networks, and then evaluate both accuracy and efficiency for all clustering algorithms on real-world networks. Our algorithm, DAOC, shows the best accuracy among the deterministic clustering algorithms on synthetic networks, outperforming others on each network and being more accurate by 25% on average according to Fig. 3. Moreover, DAOC also has the best accuracy on average among all evaluated algorithms on large real-world networks as shown in Fig. 4(a). Being parameter-free, our algorithm yields good accuracy on *both* synthetic networks and real-world networks, unlike some other algorithms having good performance on some datasets but low performance on others.

Besides being accurate, DAOC consumes the least amount of memory among the evaluated hierarchical algorithms (Louvain, OSLOM2) as shown in Table III. In particular, DAOC consumes 2x less memory than Louvain on the largest real-world evaluated network (livejournal) and 3x less memory than OSLOM2 on dblp, while producing much more fine-grained hierarchies of clusters with almost an order of magnitude more levels than other algorithms. Moreover, among the evaluated overlapping clustering algorithms, only pSCAN and DAOC are able to cluster the livejournal network within the specified execution constraints, the missing bars in Fig. 4(b) corresponding to the algorithms that we had to terminate.

VI. Conclusions

In this paper, we presented a new clustering algorithm, DAOC, which is at the same time stable and provides a unique combination of features yielding a fine-grained hierarchy of overlapping clusters in a fully automatic manner. We experimentally compared our approach on a number of different datasets and showed that while being parameter-free and efficient, it yields accurate and stable results on *any* input networks. DAOC builds on a new (micro) consensus technique, MMG, and a novel overlap decomposition approach, OD, which are both applicable on top of non-overlapping clustering algorithms and allow to produce overlapping and robust clusters. DAOC is released as an open-source clustering library implemented in C++ that includes various cluster analysis features not mentioned in this paper and that is integrated with several data mining applications (StaTIX [48], or DAOR [49] embeddings). In future work, we plan to design an approximate version of MMG to obtain near-linear execution times on dense networks, and to parallelize DAOC taking advantage of modern hardware architectures to further expand the applicability of our method.

References

[1] A. laszlo Barabasi, *Linked: The New Science of Networks*, apr 2002.

[2] B. Karrer, E. Levina, and M. E. J. Newman, "Robustness of community structure in networks," *Phys. Rev. E*, vol. 77, p. 046119, Apr 2008.

[3] R. N. Davé and R. Krishnapuram, "Robust clustering methods: a unified view," *IEEE Transactions on Fuzzy systems*, vol. 5, no. 2, 1997.

[4] M. A. Borkin, "Perception, Cognition, and Effectiveness of Visualizations with Applications in Science and Engineering," Ph.D. dissertation, Harvard University, 2014.

[5] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *The Psychol. Rev.*, vol. 63, no. 2, 1956.

[6] A. L. N. Fred and A. K. Jain, "Robust data clustering," in *CVPR*, 2003, pp. 128–136.

[7] S. Vega-Pons and J. Ruiz-Shulcloper, "A survey of clustering ensemble algorithms," *Int. J. Pattern Recogn.*, vol. 25, no. 03, pp. 337–372, 2011.

[8] A. Lancichinetti and S. Fortunato, "Consensus clustering in complex networks," *Sci. Rep.*, vol. 2, 2012.

[9] D. Mandaglio, A. Amelio, and A. Tagarelli, "Consensus community detection in multilayer networks using parameter-free graph pruning," in *PAKDD*, 2018.

[10] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, "Finding Statistically Significant Communities in Networks," *PLoS ONE*, vol. 6, 2011.

[11] M. Ovelgönne and A. Geyer-Schulz, "An ensemble learning strategy for graph clustering," ser. Contemp. Math., vol. 588, 2013, pp. 187–206.

[12] A. Tandon, A. Albeshri, V. Thayananthan, W. Alhalabi, and S. Fortunato, "Fast consensus clustering in complex networks," *Phys. Rev. E*, vol. 99, p. 042301, Apr 2019.

[13] T. Su and J. G. Dy, "In search of deterministic methods for initializing k-means and gaussian mixture clustering," *Intelligent Data Analysis*, vol. 11, no. 4, pp. 319–338, 2007.

[14] M. E. Celebi and H. A. Kingravi, "Linear, deterministic, and order-invariant initialization methods for the k-means clustering algorithm," in *Partitional Clustering Algorithms*, 2015, pp. 79–98.

[15] M. Tepper, P. Musé, A. Almansa, and M. Mejail, "Automatically finding clusters in normalized cuts," *Pattern Recognition*, vol. 44, no. 7, 2011.

[16] J. Hou and W.-X. Liu, "Clustering based on dominant set and cluster expansion," in *PAKDD*, 2017.

[17] I. Derényi, G. Palla, and T. Vicsek, "Clique percolation in random networks," *Phys. Rev. Lett.*, vol. 94, p. 160202, Apr 2005.

[18] J. M. Kumpula, M. Kivelä, K. Kaski, and J. Saramäki, "Sequential algorithm for fast clique percolation," *Phys. Rev. E*, vol. 78, no. 2, 2008.

[19] L. Chang, W. Li, X. Lin, L. Qin, and W. Zhang, "pscan: Fast and exact structural graph clustering," in *ICDE*, 2016, pp. 253–264.

[20] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J Stat Mech.*, vol. 2008, no. 10, p. P10008, oct 2008.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, 1996, pp. 226–231.

[22] B. Chen and K. M. Ting, "Neighbourhood contrast: A better means to detect clusters than density," in *PAKDD*, 2018.

[23] M. E. J. Newman, "The structure and function of complex networks," *SIAM Rev.*, vol. 45, no. 2, 2003.

[24] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113, 2004.

[25] S. Gregory, "Fuzzy overlapping communities in networks," *J Stat Mech.*, vol. 2011, no. 02, p. P02017, 2011.

[26] M. Chen and B. K. Szymanski, "Fuzzy overlapping community quality metrics," *SNAM*, vol. 5, no. 1, pp. 40:1–40:14, 2015.

[27] T. Nepusz, A. Petróczi, L. Négyessy, and F. Bazsó, "Fuzzy communities and the concept of bridgeness in complex networks," *Phys. Rev. E*, vol. 77, p. 016107, Jan 2008.

[28] H.-W. Shen, X.-Q. Cheng, and J.-F. Guo, "Quantifying and identifying the overlapping community structure in networks," *J Stat Mech.*, vol. 2009, no. 07, p. P07042, 2009.

[29] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri, "Extending the definition of modularity to directed graphs with overlapping communities," *J Stat Mech.*, vol. 3, p. 24, Mar. 2009.

[30] A. Lázár, D. Abel, and T. Vicsek, "Modularity measure of networks with overlapping communities," *EPL*, vol. 90, no. 1, p. 18001, 2010.

[31] J. Liu, "Fuzzy modularity and fuzzy community structure in networks," *Eur. Phys. J. B*, vol. 77, no. 4, pp. 547–557, 2010.

[32] S. Gregory, "A fast algorithm to find overlapping communities in networks," in *ECML PKDD*, 2008, pp. 408–423.

[33] B. H. Good, Y.-A. de Montjoye, and A. Clauset, "Performance of modularity maximization in practical contexts," *Phys. Rev. E*, vol. 81, no. 4, p. 046106, 2010.

[34] D. S. Bassett, M. A. Porter, N. F. Wymbs, S. T. Grafton, J. M. Carlson, and P. J. Mucha, "Robust detection of dynamic community structure in networks," *Chaos*, vol. 23, no. 1, p. 013142, 2013.

[35] P. C. Céspedes and J. F. Marcotorchino, "Comparing different modularization criteria using relational metric," in *GSI*, 2013, pp. 180–187.

[36] M. E. J. Newman, "Spectral methods for network community detection and graph partitioning," *Phys. Rev. E*, vol. 88, no. 4, p. 042822, 2013.

[37] M. E. J. Newman, "Equivalence between modularity optimization and maximum likelihood methods for community detection," *Phys. Rev. E*, vol. 94, 2016.

[38] A. Lutov, M. Khayati, and P. Cudré-Mauroux, "Clubmark: a parallel isolation framework for benchmarking and profiling clustering algorithms on numa architectures," in *ICDMW*, 2018, pp. 1481–1486.

[39] J. Xie, B. K. Szymanski, and X. Liu, "Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *ICDMW*, 2011, pp. 344–349.

[40] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey, "High quality, scalable and parallel community detection for large real graphs," ser. WWW '14, 2014, pp. 225–236.

[41] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, pp. 814–818, 2005.

[42] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: The state-of-the-art and comparative study," *ACM Comput. Surv.*, vol. 45, no. 4, pp. 1–35, aug 2013.

[43] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Phys. Rev. E*, vol. 80, Jul 2009.

[44] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowl. Inf. Syst.*, 2015.

[45] A. Lutov, M. Khayati, and P. Cudré-Mauroux, "Accuracy evaluation of overlapping and multi-resolution clustering algorithms on large datasets," in *BigComp*, 2019, pp. 1–8.

[46] A. V. Esquivel and M. Rosvall, "Comparing network covers using mutual information," *CoRR*, vol. abs/1202.0425, 2012.

[47] J. Yang and J. Leskovec, "Overlapping community detection at scale: A nonnegative matrix factorization approach," in *WSDM '13*, pp. 587–596.

[48] A. Lutov, S. Roshankish, M. Khayati, and P. Cudré-Mauroux, "Statix — statistical type inference on linked data," in *BigData*, 2018.

[49] A. Lutov, D. Yang, and P. Cudré-Mauroux, "Bridging the gap between community and node representations: Graph embedding via community detection," in *IEEE BigData*, 2019.