Hierarchical link clustering algorithm in networks

Jernej Bodlaj[*]
University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, 1000 Ljubljana, Slovenia

Vladimir Batagelj[†]
University of Ljubljana, Faculty of Mathematics and Physics, Jadranska ulica 19, 1000 Ljubljana, Slovenia

(Received 3 November 2014; published 24 June 2015)

Hierarchical network clustering is an approach to find tightly and internally connected clusters (groups or communities) of nodes in a network based on its structure. Instead of nodes, it is possible to cluster links of the network. The sets of nodes belonging to clusters of links can overlap. While overlapping clusters of nodes are not always expected, they are natural in many applications. Using appropriate dissimilarity measures, we can complement the clustering strategy to consider, for example, the semantic meaning of links or nodes based on their properties. We propose a new hierarchical link clustering algorithm which in comparison to existing algorithms considers node and/or link properties (descriptions, attributes) of the input network alongside its structure using monotonic dissimilarity measures. The algorithm determines communities that form connected subnetworks (relational constraint) containing locally similar nodes with respect to their description. It is only implicitly based on the corresponding line graph of the input network, thus reducing its space and time complexities. We investigate both complexities analytically and statistically. Using provided dissimilarity measures, our algorithm can, in addition to the general overlapping community structure of input networks, uncover also related subregions inside these communities in a form of hierarchy. We demonstrate this ability on real-world and artificial network examples.

DOI: 10.1103/PhysRevE.91.062814 PACS number(s): 89.75.Hc

I. INTRODUCTION

While no common definition has been agreed upon regarding communities (clusters, groups) [1] in a network, we perceive them as structures of related network elements in terms of their properties (descriptions, attributes). In a given network they span connected subnetworks. In continuation we mostly use the term cluster, yet because of the nature of described algorithms they almost universally represent a community. Communities are also supposed to have a stronger internal than external cohesion. However, highly overlapping node clusters can have more external than internal links. To detect these kinds of communities a new approach has been proposed by Lehmann *et al.* [2], using the clustering of links instead of nodes. The algorithm constructs a hierarchy of links on the input network. Another method based on similar idea is Lambiotte's [3] approach through a line graph. The Lambiotte's method can utilize any conventional node clustering algorithm on the constructed line graph of the input network. In line graph, links of the network's source graph are represented by the line graph nodes; a pair of line graph nodes is linked if and only if the corresponding links share a common end node in the source graph. A cluster of nodes in the line graph determines a cluster of links in the source graph. In both methods the disjoint link clusters determine overlapping node communities (end nodes of links) in the source network. In this paper we shall use the term *cluster* for an arbitrary nonempty subset of nodes or links in a given network and the term *community* for a cluster that, in the given network, induces a connected subnetwork.

We first present a basic algorithm which clusters nodes only and then we propose a new adapted algorithm, which clusters both nodes and links. They construct a hierarchy of nodes or links, respectively, on the input undirected network. Instead of considering only the structural information of the source network they use also data describing nodes or links. In this way we elucidate the notion of a community. There are many structural definitions of a community [1]; generally, they are thought of as being cohesive, compact, and internally well connected regions in a network while potentially also being well separated from the rest of the network if they do not overlap.

Both algorithms are based on the agglomerative clustering of nodes or links, respectively, and work under the connectedness constraint; they join only adjacent clusters of nodes or links and therefore produce connected clusters (subnetworks). The basic algorithm joins linked clusters of nodes and the adapted algorithm joins clusters of links which share a common node. Both algorithms can use either recursively or directly computed dissimilarity measures, which consider also properties of nodes and links. Recursively computed dissimilarity measures consider dissimilarities between initial clusters consisting of individual nodes or links only. Dissimilarities for joint clusters are recursively computed from dissimilarities of clusters before joining. Directly computed dissimilarity measures consider the properties of nodes and links belonging to joint clusters only and compute their value "from scratch." Additionally, they can consider the properties of clusters as a whole. Both algorithms return hierarchies of clusters. The basic algorithm returns a hierarchy of nodes and the adapted algorithm a hierarchy of links. For a selected level in the hierarchy of nodes we can extract the corresponding partition of nonoverlapping clusters of nodes. In case of the hierarchy of links, we can extract a partition of nonoverlapping clusters

[*]bodlaytm@gmail.com
[†]vladimir.batagelj@fmf.uni-lj.si

of links, which can be viewed as a covering consisting of overlapping clusters of their end nodes. Either overlapping or nonoverlapping clusters of nodes represent the communities, attributed with common values of their node and in case of adapted algorithm also link properties. Although both algorithms can be used with dissimilarities based on properties measured in any measurement scale we focus on descriptions based on sets of nominal values (keywords or tags).

An example of a network on which to use the algorithms is the collaboration network of authors. Collecting also the keywords used by each author in their works, we can more precisely identify the underlying collaboration network structure. Besides being able to identify groups of collaborating authors, we can identify the topics they are working on and we can further uncover subgroups of authors who work on the most similar topics. Another example is a network of products sold by an online store. Links in this network represent the "other customers bought or viewed" relation. Considering the descriptions of products, we can identify *the most* related and also compatible clusters of products. A customer might view related but incompatible products. He or she checked, for example, a camera with an SD memory card slot and another camera with a CF memory card slot. A customer also checked for both memory cards. When considering only interlinks between these four products, we would detect only one cluster of related products, but when additionally considering descriptions of those products, we can further identify two subclusters.

Similar networks were used in Yang *et al.* [4], but they do not contain descriptions. While we use one network from their data set for a performance evaluation of the adapted algorithm, we collected other test networks on our own. Both algorithms can also be used on networks where no additional attributes are known by assigning to each node its own unique descriptor (identifier, label). In this case, algorithms favor joining clusters of nodes or links in a way that the number of nodes in a joint cluster increases the least tightly connected clusters.

In the next section we give a short review of some related works. In Sec. III we overview the agglomerative approach to hierarchical clustering on nodes and extend it to links in Sec. IV. In Sec. V we discuss suitable dissimilarity measures for both algorithms and their monotonicity. The proofs of the monotonicity of the selected dissimilarity measures can be found in the Appendix A. We then present two aspects of the algorithm evaluation. We examine the time and space complexity of algorithms in Sec. VI. We apply algorithms on example networks and evaluate results in Sec. VII. In Appendix B we give some details on the implementation of algorithms.

II. RELATED WORK

In Fortunato's review article [5] a exhaustive overview of network clustering approaches is given. He discusses topics from the definition of the main elements of the clustering problem to the presentation of most methods developed. He puts the focus on techniques designed by physicists, from the discussion of crucial issues like the significance of clustering and how methods should be tested and compared against each other to the description of applications on real networks. Currently, the best techniques in overlapping community detection are described in the work of J. Xie *et al.* [6]. Quality measures and benchmarks are provided in a comparison of 14 different algorithms. They were grouped into different approach techniques such as clique percolation proposed by Palla *et al.* [7], local expansion and optimization [8,9], fuzzy detection [10], agent-based and dynamical algorithms [11], line graph and link partitioning algorithms which are the basis for our approach, and others. A recent article of Yang *et al.* [4] addresses the problem of community evaluation. They compare various community detection methods in terms of similarity of identified communities with the real underlying communities based on the ground-truth knowledge about input networks.

Community detection methods, which inspired the development of this work, include the Louvain method [12], the Ferligoj and Batagelj approach [13], and the method of Kim *et al.* [14]. The first is a simple, efficient, and easy-to-implement method for identifying communities in large networks. It is based on the idea of greedy optimization of *modularity* of a partition of the network [15]. The approach of Ferligoj and Batagelj is an adaptation of the standard agglomerative clustering method for a clustering with a relational constraint. In their second paper [16] it is extended to nonsymmetric relations. To obtain a fast algorithm also for large networks, the computation of dissimilarities is bounded to the pairs of end nodes of existing links [17]. The method of Kim *et al.* attempts to detect communities of related Web sites using a semantic information while performing a link-based clustering. In their paper they describe a procedure to intelligently determine and look over a limited and semantically related set of Web pages. Another key work is the book written by F. Murtagh [18]. In his book he explains many principles and algorithms of multidimensional hierarchical clustering, and one of those algorithms represents the basis for our adapted algorithm. We will refer to it as Murtagh's algorithm although it is essentially based on ideas presented by C. de Rham in [19].

III. A BASIC AGGLOMERATIVE APPROACH TO NETWORK CLUSTERING

At its core, the algorithm is identical to a classical agglomerative clustering algorithm. It, however, considers, in addition to the dissimilarity, a relational constraint: Nodes inside clusters must be connected [13]. The resulting hierarchy of nodes can be, for example, interpreted as a multilevel abstraction of similar connected parts in the network. It can be used to find shortcuts between similar connected parts and to optimize searching algorithms in the network [20].

A. The algorithm

We denote the input network for the algorithm with N. We assume that it is connected, undirected, and without loops (links with both end nodes the same). It is composed from three sets: $N = (V,E,P)$, the set of nodes V, the set of undirected links or edges E, and the set of node properties P. The cardinality of the set of nodes will be denoted by $n = |V|$ and the cardinality of the set of edges by $m = |E|$. A nonempty subset C of the set $V, \emptyset \subset C \subseteq V$, is called a cluster of nodes. A clustering, i.e., the set of clusters, will be denoted by $\mathcal{C}$; $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. A clustering $\mathcal{C}$ is called a partition if and only if for all $C_s, C_t \in \mathcal{C}, s \neq t \Rightarrow C_s \cap C_t = \emptyset$, and $\bigcup_{C \in \mathcal{C}} C = V$. With $D(C_s, C_t)$ we will denote a dissimilarity measure between two clusters C_s and C_t and with $\mathbf{D}$ a set (list, dictionary, or sparse matrix) of dissimilarities between available clusters in the process of the algorithm. The dissimilarity D is usually expressed in terms of a selected dissimilarity d between nodes.

The inputs for the algorithm are a network N and a dissimilarity measure D. We denote a relational constraint with $\psi(C_s, C_t) \equiv \exists u \in C_s, v \in C_t : (u, v) \in E$. It is satisfied (true) if and only if there exists an edge linking the clusters C_s and C_t in the network N:

```
$k \leftarrow 1; \mathcal{C}_1 \leftarrow \{\{v\} : v \in V\}$;
**for all** $C \in \mathcal{C}_1$ **do** $h(C) \leftarrow 0$;
**for all** $(u : v) \in E$ **do** $\mathbf{D}[\{u\}, \{v\}] \leftarrow d(u, v)$;
**while** $\exists C_i, C_j \in \mathcal{C}_k : (i \neq j)$ **do**
    $(C_p, C_q) \leftarrow \text{argmin}\{\mathbf{D}[C_i, C_j] : i \neq j \wedge \psi(C_i, C_j)\}$;
    $C \leftarrow C_p \cup C_q; k \leftarrow k + 1$;
    $h(C) \leftarrow \mathbf{D}[C_p, C_q]$;
    $\mathcal{C}_k \leftarrow \mathcal{C}_{k-1} \setminus \{C_p, C_q\} \cup \{C\}$;
    **for all** $C_s \in \mathcal{C}_k \setminus \{C\} : \psi(C, C_s)$ **do**
        $\mathbf{D}[C, C_s] \leftarrow D(C, C_s)$
    remove entries containing $C_p$ or $C_q$ from $\mathbf{D}$;
    shrink $N$ by combining neighborhoods $n[C_p]$ and
        $n[C_q]$ into $n[C] \leftarrow n[C_p] \cup n[C_q]$;
**end while**
```

The algorithm determines a series of partitions $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$. Their union $\mathcal{H} = \cup_i \mathcal{C}_i$ is a tree hierarchy. It has the property $\forall C_s, C_t \in \mathcal{H} : C_s \cap C_t \in \{\emptyset, C_s, C_t\}$. The returned function $h(C)$ is a level function on the hierarchy $\mathcal{H}$ and together they form a clustering tree or a dendrogram $(\mathcal{H}, h)$.

IV. AN AGGLOMERATIVE APPROACH FOR CLUSTERING LINKS

The adapted algorithm for clustering links works on a similar principle as the basic algorithm from Sec. III but instead of nodes it clusters links of the source network into the hierarchy. The dissimilarity measure for the adapted algorithm is based on properties of links and/or nodes and the network structure. To preserve the connectivity of clusters, the algorithm joins only clusters of links which share a common node.

A. The adapted algorithm

Instead of node clusters and node clusterings we operate with link clusters and link clusterings in the adapted algorithm. A link cluster denoted by L is a nonempty subset of the set E; $\emptyset \subset L \subseteq E$. A link clustering is a set of link clusters and a link partition is a link clustering containing disjoint link clusters which cover the entire set E. We denote it with $\mathcal{L}$; $\mathcal{L} = \{L_1, L_2, \ldots, L_p\}$, $s \neq t \Rightarrow L_s \cap L_t = \emptyset$, and $\bigcup_{L \in \mathcal{L}} L = E$. The function $\text{ext}(e) = \{u, v\}$ is used to get the set of end nodes of the link $e = (u : v) \in E$. Additional two sets we use are a set of link cluster nodes $C_t = \text{ext}(L_t) = \bigcup_{e \in L_t} \text{ext}(e)$, denoted by C_t as it represents the cluster of nodes as in the basic algorithm in Sec. III A, and a set of active nodes A_t, $A_t \subseteq C_t$, which is used to identify adjacent link clusters; $A_t = \{v \in C_t; \exists s \neq t : v \in C_s\}$.

As the basic algorithm, the adapted algorithm is given the same two inputs: a network N and a dissimilarity measure D. However, D must be monotonic. Monotonic dissimilarity measures are discussed in Sec. V. Instead of the hierarchy of nodes, the algorithm returns the hierarchy of links based on the network N. The relational constraint of the adapted algorithm is given by $\psi_a(L_s, L_t) \equiv A_s \cap A_t \neq \emptyset$ and is equivalent to $C_s \cap C_t \neq \emptyset$. $\psi_a(L_s, L_t)$ is satisfied if and only if node clusters C_s and C_t belonging to their link clusters L_s and L_t share a common (active) node in the network N. The content of the main loop of the adapted algorithm is similar to the content of the main loop of the basic algorithm. The principle of operation, however, differs. The adapted algorithm is based on Murtagh's algorithm [18], which is used to cluster nodes in a network. We use it on the implicitly constructed line graph, respectively, the line subgraph of the input network. Using this approach we can take advantage of its time efficiency and additionally lower the space complexity, which would greatly increase if we used the naive approach using the whole line graph. Complexities of both algorithms are discussed in Sec. VI. An implementation of the adapted Murtagh's algorithm for clustering nodes is available in Pajek [17].

The idea of Murtagh's algorithm is in local clustering of the input network. The local clustering takes place at the end of a chain (path of consecutive adjacent node clusters in the input network), built iteratively by the algorithm and starting from a randomly selected cluster of nodes. In our algorithm the chain is denoted by *chain*. We define the following operations on it. *Push* is used to add an element to the end of the chain. *Pop* is used to retrieve and remove the last element from the chain. The operation *Last* retrieves only the last element of the chain. To implement the chain a stack is used. Murtagh's algorithm builds a chain by following the direction of the most decreasing value of dissimilarity between consecutive adjacent clusters of nodes, i.e., clusters of nodes which by pairs respect the relational constraint. In case it is not possible to elongate the chain in any direction from the cluster of nodes at its end, i.e., a cluster of nodes with a strictly smaller dissimilarity to the last cluster in the chain does not exists, the algorithm reached two consecutive cluster of nodes, which are reciprocally the least dissimilar. We call such pair the reciprocal nearest neighbors (RNNs). Due to the monotonicity of a chosen dissimilarity measure, which is a must for the correct working of the Murtagh's algorithm, it is possible to show [18] that when a joining of RNN clusters takes place, the dissimilarities between a new joint cluster of nodes and other clusters of nodes, i.e., neighbors of the previous pair of RNN clusters, at most increase. The remaining part of the already-built chain remains valid. With reference to this we discuss the Bruynooghe's reducibility property in Sec. V. The algorithm then removes both RNN clusters from the chain and joins them into a new cluster. If the chain still contains any clusters, the algorithm continues building it. In the opposite case it randomly selects a new unvisited cluster of nodes and starts building a new chain with it, and so on. The algorithm stops when the last two clusters of nodes are joined. The method is a more efficient version of the basic algorithm from Sec. III A. It, however, works correctly if a monotonic dissimilarity measure is used only. In theory, we can design an input network which will lead the Murtagh's algorithm to build only one long chain across all the clusters of nodes, forcing it to run by its worst complexity, which is the same as the complexity of the basic algorithm in this case, but this happens extremely rarely in practice.

In our link clustering algorithm we use the same principles as in the Murtagh's algorithm, i.e., the local clustering at the end of a chain of clusters of links from the input network. During the process of our algorithm, the line graph is being incrementally built in the neighborhood of the chain of clusters of links. We will therefore refer to it as an incremental line subgraph. The nodes of the line graph represent the clusters of links from the input network, the links in the line graph represent the adjacency in terms of the relational constraint ψ_a, and their weight represents the dissimilarity between the clusters of links in the input network. In case the algorithm empties a given chain in full, it frees the memory occupied by the related incremental line subgraph along with the calculated dissimilarities. We could design the algorithm to do so also if, for example, the current incremental line subgraph consumes too much memory. However, in this case the algorithm would have to reconstruct the previously constructed part of the line subgraph. Some parts of the line graph have to be reconstructed also in the first case. Using the appropriate optimization, for example, by keeping as much of the line subgraph in the memory as possible, by beginning to build new chains in the region of already available line subgraphs, and by using some form of active avoidance of the clusters of links containing links with high-degree end nodes in the input network, which form complete subgraphs in the line graph, it would be possible to minimize the size of the incremental line subgraphs and, consequentially, reduce the memory consumption of the algorithm. In our case the incremental line subgraph will be denoted by ils. We define some operations on it. If the link or the dissimilarity between clusters of links of the input network L_s and L_t, respectively, is present in the ils, the operation $GetCalculateD(L_s,L_t)$ will return it from the memory. Otherwise, it is calculated using $D(L_s,L_t)$ and stored into ils and its value is returned. The operation $SetD(L_s,L_t)$ calculates the dissimilarity between clusters of links $D(L_s,L_t)$ and stores it or updates it in the memory. Operation $ClearD(L_s)$ removes all links or respective dissimilarities which have the cluster L_s in a pair from the memory. Operation $Clear$ frees the entire incremental line subgraph from the memory. For the efficient implementation of the incremental line subgraph we use a pair of structures: the dictionary of dissimilarities between pairs of clusters of links from the input network, i.e., the dictionary of links from the line subgraph, which represent pairs of adjacent clusters of links and dissimilarities between them in the input network. The second structure used is the dictionary of sets of adjacent clusters of links in the input network. It serves for the efficient access of clusters of links adjacent to a given cluster of links in the line subgraph. To aid the algorithm we keep also for each node of the input network a set of clusters of links which in a given node intersect, i.e., ψ_a is true for any chosen pair of clusters of links from a given node's set. If the node's set has at least two clusters, then the node is active. By U we denote a set of unvisited clusters of links and by $PickFrom(U)$ the procedure which removes a random element from U and returns it:

```
$k \leftarrow 1; \mathcal{L}_1 \leftarrow \{\{e\}; e \in E\}$;
**for all** $L \in \mathcal{L}_1$ **do** $g(L) \leftarrow 0$;
$U \leftarrow \mathcal{L}_1$;
$chain \leftarrow [\,]$;
**while** $k < m$ **do**
    **if** $chain = [\,]$ **then**
        $ils.Clear$;
        $L_x \leftarrow PickFrom(U)$;
    **else**
        $L_x \leftarrow chain.Pop$;
    **end if**
    $chain.Push(L_x)$;
    **while** $chain$ does not end with RNNs **do**
        use the cluster $L'$ in accordance with the relational constraint $\psi_a(chain.Last,L')$ to elongate the $chain$ in the direction of the most strictly decreasing dissimilarity $ils.GetCalculateD(chain.Last,L')$;
    **end while**
    $L_p \leftarrow chain.Pop$; $L_q \leftarrow chain.Pop$;
    $L \leftarrow L_p \cup L_q$; $k \leftarrow k + 1$;
    $g(L) \leftarrow ils.GetCalculateD(L_p,L_q)$;
    $\mathcal{L}_k \leftarrow \mathcal{L}_{k-1} \setminus \{L_p,L_q\} \cup \{L\}$;
    $U \leftarrow U \setminus \{L_p,L_q\} \cup \{L\}$;
    **for all** $L' \in \mathcal{L}_k \setminus \{L\} : \psi_a(L',L)$ **do** $ils.SetD(L',L)$;
    $ils.ClearD(L_p)$; $ils.ClearD(L_q)$;
    shrink $N$ by combining neighborhoods $n[L_p]$ and
      $n[L_q]$ into $n[L] \leftarrow n[L_p] \cup n[L_q]$;
**end while**
```

The adapted algorithm determines a series of link partitions $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_m$. Their union $\mathcal{G} = \cup_i \mathcal{L}_i$ is called a link hierarchy. The returned function $g(L)$ is a level function on the hierarchy $\mathcal{G}$ and together they form a link dendrogram $(\mathcal{G},g)$. In the implementation of the adapted algorithm the hierarchy of clusters can be represented by the list of pointers to parents and dissimilarities between clusters using a list of their values. Details on our implementation of the algorithm can be found in the Appendix B.

Please note that using the basic algorithm it is possible to get exactly the same results as with the adapted algorithm, as long a monotonic dissimilarity measure was used. This should be the case anyway. The basic algorithm should be executed on the whole line graph of the original network and use the dissimilarity measure between clusters of nodes in the line graph, identical to the dissimilarity measure between the corresponding links (clusters of links) in the original network. In this case the basic algorithm determines the hierarchy of node clusters in the line graph, which is equivalent to the hierarchy of link clusters in the original network, otherwise obtained using the adapted algorithm. However, conversion of the whole original network into the line graph may require much more space. Every node with a degree k in the original network is converted into an embedded complete subgraph of size k in the line graph. The adapted algorithm in most cases avoids a complete conversion of the input network, especially on real-world networks and, consequentially, uses much less space. In Sec. VI we show this on some examples. On specifically designed networks, however, it can be forced to construct a whole corresponding line graph.

V. DISSIMILARITY MEASURES

The execution flow and the results of clustering algorithms strongly depend on the choice of dissimilarity measures. In every iteration of the algorithm, two closest clusters of either nodes or links are joined, based on the selected dissimilarity measure. We divide dissimilarity measures into two groups:

(i) dissimilarity measures which are directly computed from the cluster descriptions (Sec. V A) and

(ii) dissimilarity measures recursively computed from data about the clusters and dissimilarities between them (Sec. V B).

A common problem with dissimilarity measures is related to monotonicity. Using an arbitrary dissimilarity measure, some clusters joined closer to the root in the hierarchy later in the course of the agglomerative clustering algorithm may be more similar than at least one pair of subclusters joined earlier, or deeper, in the hierarchy. If this is the case, then we get an inversion in the resulting hierarchy. This property is unwanted as it introduces ambiguity; at the same level of hierarchy multiple distinct splits into clusters are possible (See Fig. 1). In addition to these issues, some algorithms, like our adapted algorithm, should not be used with nonmonotonic dissimilarity measures.

If for all iterations of the algorithm, for every joining of clusters C_p and C_q into a cluster C, the dissimilarity measure guarantees a hierarchy with a level function for which it holds $\max[h(C_p),h(C_q)] \leqslant h(C)$, the dissimilarity is monotonic. From the clustering theory we know the Bruynooghe's *reducibility property* [21], which is a sufficient condition for a dissimilarity measure D to be monotonic. It is given by the condition:

$$\begin{aligned}&D(C_p,C_q) \leqslant D(C_p,C) \;\wedge\; D(C_p,C_q) \leqslant D(C_q,C)\\ &\Rightarrow D(C_p,C_q) \leqslant D(C_p \cup C_q,C),\end{aligned} \tag{1}$$

where C_p, C_q, and C are clusters of nodes or links. Equation (1) expresses the fact that if two clusters C_p and C_q are the closest among clusters, then clusters C_p and C_q are at least as similar as a joint cluster $C_p \cup C_q$ is similar to any other cluster C, simply meaning, that as soon as two most similar clusters are joined, it is no longer possible to find any other cluster which would be more similar to the joint cluster than it is to individual clusters before the joining.

A. Directly computed dissimilarity measures

Directly computed dissimilarity measures are based on properties of network elements collected in the clusters and on the structure of the clusters.

1. Simple dissimilarity measures between clusters

A simple dissimilarity measure considering only the structure of the network is based on Jaccard correlation measure [22]. It is defined by:

$$D_J(L_p,L_q) = D_J(C_p,C_q) = 1 - \frac{|C_p \cap C_q|}{|C_p \cup C_q|} = \frac{|C_p \oplus C_q|}{|C_p \cup C_q|}, \tag{2}$$

where C_p and C_q are clusters of end nodes belonging to clusters of links L_p and L_q, respectively. $\oplus$ is the symmetric difference of sets. The problem with this measure is that it is not always monotonic.

Its improved version for links is the weighted Jaccard dissimilarity, given by:

$$D_w(L_p,L_q) = \frac{|C_p \oplus C_q|}{|C_p \cup C_q|}|L_p \cup L_q|. \tag{3}$$

To limit D_w to the interval $[0,1]$, we can additionally divide the right side of (3) by the constant $m = |E|$. In most cases the resulting hierarchies of links contain fewer inversions, but monotonicity is still not guaranteed.

D_J and D_w are both directly computed dissimilarity measures. They were the first attempts to develop the dissimilarity measures we use in our algorithms.

2. Dissimilarity measures based on keywords

While the idea of these measures originates from the bibliographic network analysis, we will explain it through the example of a collaboration network $N = (V,E,\kappa,\emptyset)$ of authors V, $\kappa : V \to [K]$, who use keywords from the set K. For every author a list of keywords and their usage frequency (or some other non-negative measure of importance) in their works is known. The collaboration network is constructed only for a specific scientific field and a time period. Based on keywords, we further investigate clusters of connected authors. In other words, we want to find subclusters of collaborating authors that use the most similar keywords (are working on similar topics). Based on lists of keywords of each subcluster we can also determine subtopics. In Eq. (4) we use a list F_s with pairs $(k : f_{s,k})$ as entries, where $f_{s,k}$ is the frequency of every keyword $k \in K$ used by the author $s \in V$. F_C and $f_{C,k}$ are analogously extended to a cluster of authors C,

$$\begin{aligned}F_s &= \{(k : f_{s,k}); k \in K\}, \quad s \in V\\ F_C &= \{(k : f_{C,k}); k \in K\}, \quad f_{C,k} = \sum_{s\in C} f_{s,k}.\end{aligned} \tag{4}$$

Note that if the author s does not use a keyword k, then $f_{s,k} = 0$. Analogously, $f_{C,k} = 0$ if in the cluster C a keyword k is not present. In an implementation the entries $(k : f_k)$ with $f_k = 0$ are omitted. A natural representation is a dictionary data structure.

In the Eq. (5) we extend the definition (4) to the list of frequencies for every given keyword k used by two collaborating authors s and t or by two clusters of authors C_p and C_q together,

$$F_s \sqcup F_t = \{(k : f_{s,k} + f_{t,k}); k \in K\}$$
$$F_{C_p} \sqcup F_{C_q} = \left\{\left(k : f_{C_p,k} + f_{C_q,k}\right); k \in K\right\}. \tag{5}$$

In Eq. (6) the absolute difference between frequencies of the individual keyword used by two authors, respectively, by two clusters of authors, is given:

$$F_s \Box F_t = \{(k : |f_{s,k} - f_{t,k}|); k \in K\},$$
$$F_{C_p} \Box F_{C_q} = \left\{\left(k : \left|f_{C_p,k} - f_{C_q,k}\right|\right); k \in K\right\}. \tag{6}$$

Finally, in Eq. (7) the new directly computed dissimilarity measure is defined. It is a normalized version of the absolute difference between keyword frequencies of two authors or two clusters of authors:

$$D_k(C_p,C_q) = \frac{\left|F_{C_p} \Box F_{C_q}\right|}{\left|F_{C_p} \sqcup F_{C_q}\right|},$$
$$d_k(s,t) = D_k(\{s\},\{t\}) = \frac{|F_s \Box F_t|}{|F_s \sqcup F_t|}, \tag{7}$$

where $|F| = \sum_{f \in F} f$. The dissimilarity d_k can be used as a part of one of the recursively computed dissimilarity measures described later in Sec. V B and D_k can be used directly. However, D_k is nonmonotonic.

Both d_k and D_k are dissimilarity measures between node clusters. Using recursive principles (Sec. V B) it is possible to use d_k in the adapted algorithm, but we have to adapt it appropriately. Instead of lists of keywords F_s and F_t of respective authors s and t, we have to use the lists of keywords F_u and F_v assigned to respective links u and v.

A simple way to obtain a list of keywords F_e for a link (edge) e is to consider a union of its end nodes's keyword lists. Therefore, for every link $e = (s : t) \in E$, we obtain its keyword list as $F_e = F_s \sqcup F_t$. To generate sets of keywords for the links in our benchmark networks to test our adapted algorithm we used this approach. In general, instead of combining properties of link's end nodes, we could directly use the properties of links as well.

3. *Some directly computed monotonic dissimilarity measures*

In Eq. (12) we now give three examples of monotonic dissimilarity measures for units described by keywords as in Sec. V A 2. Let us start with a definition of an auxiliary function gt:

$$\mathrm{gt}(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

For clusters C_p and C_q we then define:

$$S_{\mathrm{SF}}(C_p,C_q) = \sum_{k \in K} \max\left(f_{C_p,k}, f_{C_q,k}\right), \tag{9}$$

$$S_{\mathrm{SC}}(C_p,C_q) = \sum_{k \in K} \max\left[\mathrm{gt}\left(f_{C_p,k}\right), \mathrm{gt}\left(f_{C_q,k}\right)\right], \quad \text{and} \tag{10}$$

$$S_{\mathrm{MF}}(C_p,C_q) = \max_{k \in K}\left(f_{C_p,k}, f_{C_q,k}\right). \tag{11}$$

Finally, the dissimilarity $D_x(C_p,C_q)$, where x is SF, SC, or MF, is defined as follows:

$$D_x(C_p,C_q) = \begin{cases} S_x(C_p,C_q) & p \neq q \\ 0 & p = q \end{cases}. \tag{12}$$

The first measure, D_{SF}, favors clusters with similar sets of keywords and with similar frequencies of individual keywords. The second measure, D_{SC}, is the simplification of the first measure. It considers only a presence of an individual keyword instead of considering their frequencies. The third measure, D_{MF}, is the opposite of the second one. It considers the frequencies of keywords only.

The reasoning behind these dissimilarities is the following. Note that $F_C = F_{C_p} \sqcup F_{C_q}$ when $C = C_p \cup C_q$. When a set of keywords of a cluster C_p is a superset of keywords of a neighbor cluster C_q in the course of the algorithm and the dissimilarity between C_p and C_q is the lowest, the cluster C_p gets its frequencies of keywords incremented by C_q's frequencies of keywords following the measure D_{SF} or is left unchanged, keeping its original frequencies of keywords directed by measures D_x, x being either SC or MF: $C_q \subseteq C_p \Rightarrow D_x(C_p,C_q) = S_x(C_p,C_p)$. This effect can be viewed as the act of engulfing clusters with smaller sets of the same keywords, favored by all three dissimilarities. As soon as the engulfing, due to less compatible clusters, cannot take place, dissimilarities favor joining the least different neighbor clusters.

Dissimilarity measures D_{SF}, D_{SC}, and D_{MF} can be used in the adapted algorithm as well, but we have to modify them first. Instead of keyword lists F_{C_p} and F_{C_q} assigned to clusters of authors C_p and C_q, we have to use the lists of keywords F_{L_p} and F_{L_q} assigned to clusters of links L_p in L_q. Lists of keywords assigned to clusters of links can be obtained using the same approach as for the clusters of authors or nodes, respectively, for the dissimilarity measure d_k.

B. Recursively computed dissimilarity measures

For pairs of trivial clusters a dissimilarity measure from this group can be directly obtained using the dissimilarity d between individual nodes of a network. The dissimilarity between nontrivial disjoint clusters C_p and C_q, $D(C_p,C_q)$, is obtained recursively from dissimilarities of their merged subclusters.

For example, in the traditional single-linkage (also known as minimal-linking) method the dissimilarity between two clusters is determined as the smallest dissimilarity between any two elements, each being from a different cluster:

$$D_m(C_p,C_q) = \min_{u \in C_p, v \in C_q} d(u,v). \tag{13}$$

In this case we get the recursive relation:

$$\begin{aligned} D_m(C, C_p \cup C_q) &= \min_{u \in C, v \in C_p \cup C_q} d(u,v) \\ &= \min\left(\min_{u \in C, v \in C_p} d(u,v), \min_{u \in C, v \in C_q} d(u,v)\right) \\ &= \min\left(D_m(C,C_p), D_m(C,C_q)\right). \end{aligned} \tag{14}$$

The definition of the dissimilarity D_m requires a whole matrix of dissimilarities d on $V \times V$—at least $O(n^2)$ space. This is too much for large sparse networks. To get an efficient algorithm, we limit the definition of the dissimilarity D to dissimilarities between the end nodes of existing links [17] only. Let $E(C_p,C_q) = \{e \in E : \text{ext}(e) \cap C_p \neq \emptyset \wedge \text{ext}(e) \cap C_q \neq \emptyset\}$. For nodes u and v we then set

$$D(\{u\},\{v\}) = \begin{cases} d(u,v) & (u:v) \in E \\ \infty & \text{otherwise} \end{cases} \tag{15}$$

and for an analog to the single-linkage (13) dissimilarity, we define the constrained single-linkage dissimilarity:

$$D_{\text{mc}}(C_p,C_q) = \min_{(u:v)\in E(C_p,C_q)} d(u,v). \tag{16}$$

It is easy to verify that

$$D_{\text{mc}}(C,C_p \cup C_q) = \min\left(D_{\text{mc}}(C,C_p),D_{\text{mc}}(C,C_q)\right). \tag{17}$$

Similarly, we introduce the constrained maximal-linkage dissimilarity:

$$\begin{aligned} D_{\text{Mc}}(C_p,C_q) &= \max_{(u:v)\in E(C_p,C_q)} d(u,v) \\ D_{\text{Mc}}(C,C_p \cup C_q) &= \max\left(D_{\text{Mc}}(C,C_p),D_{\text{Mc}}(C,C_q)\right) \end{aligned} \tag{18}$$

and the constrained average-linkage dissimilarity:

$$\begin{aligned} D_{\text{ac}}(C_p,C_q) &= \frac{1}{w[E(C_p,C_q)]} \sum_{(u:v)\in E(C_p,C_q)} d(u,v) \\ D_{\text{ac}}(C,C_p \cup C_q) &= \frac{w[E(C,C_p)]}{w[E(C,C_p \cup C_q)]} D_{\text{ac}}(C,C_p) \\ &\quad + \frac{w[E(C,C_q)]}{w[E(C,C_p \cup C_q)]} D_{\text{ac}}(C,C_q), \end{aligned} \tag{19}$$

where $w : E \to \mathbb{R}^+$ is a weight of individual links, $w(L) = \sum_{l\in L} w(l)$, $L \subseteq E$, and $w[E(C,C_p \cup C_q)] = w[E(C,C_p)] + w[E(C,C_q)]$ for $E(C,C_p) \cap E(C,C_q) = \emptyset$. For $E(C_p,C_q) = \emptyset$ we define $D_x(C_p,C_q) = \infty$, where x is mc, Mc, or ac.

On a complete network (graph) of size n, $N \equiv K_n$, the constraint dissimilarities reduce to the corresponding traditional dissimilarities. Other dissimilarity measures exist and most of them are described in Fortunato's review article [5]. We, however, do not know if their constrained versions can be transformed into recursion as we have shown in the upper examples.

To use D_{mc}, D_{Mc}, and D_{ac} in the adapted algorithm, where link clusters L_p and L_q are used, we have to consider a corresponding function of active nodes $A(L_p,L_q)$ instead of $E(C_p,C_q)$; $A(L_p,L_q) = \{v \in V : v \in C_p; \exists u \neq v : u \in C_q\}$, where C_p and C_q are node clusters belonging to their respective link clusters as described in Sec. IV. For links u and v we then set

$$D(\{u\},\{v\}) = \begin{cases} d(u,v) & \text{ext}(u) \cap \text{ext}(v) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}. \tag{20}$$

For example, the constrained single-linkage dissimilarity adapted to link clusters is

$$\begin{aligned} &D_{mcl}(L_p,L_q) \\ &= \min_{u\in L_p, v\in L_q : \text{ext}(u)\cap A(L_p,L_q)\neq\emptyset \wedge \text{ext}(v)\cap A(L_p,L_q)\neq\emptyset} d(u,v). \end{aligned} \tag{21}$$

1. Lehmann's dissimilarity

An example of recursively computed dissmilarity measure, based on Jaccard dissimilarity, is used in Lehmann's work [2] to determine the dissimilarity between link clusters. The dissimilarity between individual adjacent links u and v is given by:

$$d_L(u,v) = \frac{|\text{n}(u) \oplus \text{n}(v)|}{|\text{n}(u) \cup \text{n}(v)|}. \tag{22}$$

The term $\text{n}(u)$ represents the set of neighbor nodes of a link u in a network, i.e., the set of nodes accessible from link u through a single link. If links u and v do not share a node (are not adjacent), then $d_L(u,v) = 1$. A major difference between the Jaccard dissimilarity from Sec. V A 1 and Lehmann's one is in how to determine the dissimilarity between link clusters. They use the single-linkage principle:

$$D_L(L_p,L_q) = \min_{u\in L_p, v\in L_q} d_L(u,v) \tag{23}$$

and therefore the Lehmann's dissimilarity generates monotonic hierarchies. A related proof is given in Appendix A 2.

VI. ALGORITHM COMPLEXITIES

A. Basic algorithm

The basic algorithm, described in Sec. III is quite efficient. The time complexity is limited by the number of source network's nodes n and its maximal degree Δ, $O(n\Delta)$. In the worst case, if $\Delta = n$, the complexity can become quadratic, but networks with such high maximal degree are rare in practice. This applies also to scale-free networks which we focus on in this paper. We ran the algorithm on generated scale-free networks [23,24] of sizes between 10 and 10 000 nodes. The maximal degree in any of them was 6780. To generate networks we used the network analysis package NETWORKX [25]. We assigned a distinct keyword to each node, i.e., a string of node's consecutive number in a network. On Fig. 2 we provide the diagram of average running times in relation to the product of network size and its maximal node degree. Depending on the shape of source network and its weights, the worst-case scenario is a star network ($\Delta = n - 1$) where in each iteration of the algorithm all remaining dissimilarities should be recalculated. In this rare case the time complexity becomes quadratic $O(n^2)$. The space complexity is linear in terms of the number of links in the source network m, $O(m)$.

The algorithm starts with n clusters of nodes C_i, $i = 1,\ldots,n$, where each node belongs to its own cluster. A priority queue Q is then filled with dissimilarities between individual clusters of nodes, but only where nodes in its clusters are linked in the network. We set m dissimilarities in the initial Q. For each link $(u : v) \in E$ we have a dissimilarity $d(C_u,C_v)$. In every iteration of the algorithm two clusters C_p and C_q are joined into a new cluster of nodes C and links (at least one) between nodes from just joined clusters are absorbed into the cluster. Dissimilarities between former clusters C_p or C_q and all other clusters are removed from the priority queue. The time complexity is minimal in path networks. In this case when any link representing the lowest dissimilarity between two clusters is chosen, at worst two dissimilarities should be

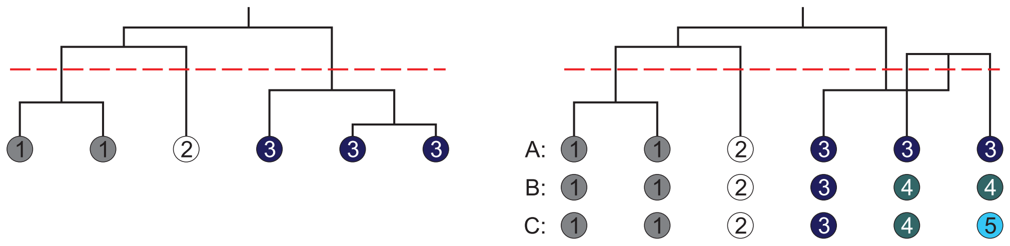

FIG. 1. (Color online) Two dendrograms of hierarchies obtained by a monotonic dissimilarity (left) and a nonmonotonic dissimilarity (right). While the clustering partition on the left, obtained by cutting the hierarchy into clusters 1, 2, and 3, is uniquely defined, the clustering partition in the right case is not. We have to decide which of the three clustering partitions, A, B, or C, to choose.

recalculated, giving us $O(n)$. In the case of a star, however, whichever dissimilarity is chosen, all remaining dissimilarities should be recalculated. Please note that each cluster carries also the underlying union of keywords that each of their nodes has. If the number of keywords is large and the dissimilarity measure takes all of them into account, the time complexity may increase considerably.

B. Adapted algorithm for links

It is harder to estimate the complexity of the adapted algorithm as it is more complex and utilizes additional data structures. As seen from the statistical analysis in the continuation, the complexity is heavily dependent on the input network structure and its element properties. The choice of a dissimilarity measure has a significant impact on the complexity as well. The algorithm needs the most resources on fully linked regions in networks (complete subgraphs). It is useful to know their presence in the source network. Using the k-cores algorithm [26] on the source network N, we are able to identify regions with possibly large complete subgraphs in N. The k-cores algorithm runs in $O(m)$, m being the number of links in the network. The main core, i.e., the core of the largest order k, can contain complete subgraphs of size $k+1$ but not of size $k+2$. The complexity of the adapted algorithm is dependent readily on the presence of individual nodes with high degrees in the network, as inside the corresponding line graph they evolve into complete subgraphs. It is quite obvious that it is good to first cluster more isolated links in the network. After joining any number of links in the close neighborhood of nodes with high degrees, the degrees of these get decreased and they therefore generate smaller subgraphs later. Our adapted algorithm does not have any built-in system to avoid high-degree nodes.

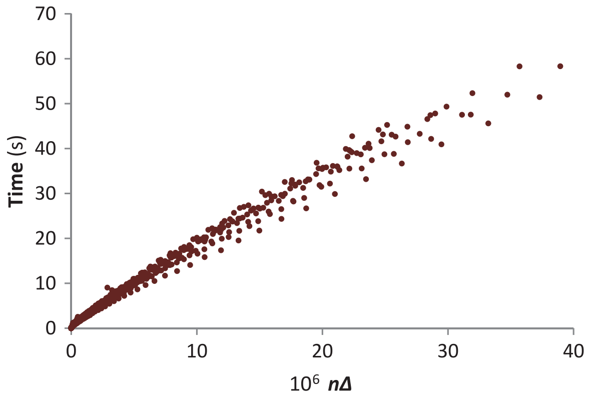

FIG. 2. (Color online) A diagram of running times of the basic algorithm in relation to the product of the number of nodes and the maximal degree in random generated scale-free networks. Each node had its own single unique keyword.

For path networks and for complete networks we determined complexities of the adapted algorithm analytically. These forms of the input network represent both possible extremes. By measuring the time required for the algorithm to cluster individual networks from a set of generated scale-free networks, which are more common in practice, we determined its complexity statistically.

Using the dissimilarity measure D_{SF} we analyzed the time complexity of the adapted algorithm for the simplest connected networks—paths of size $m = n - 1$ links. We assume the complexity of the dissimilarity D_{SF} calculation takes a constant time (set K is small). As the algorithm joins two adjacent clusters of links in each iteration and calculates at most two dissimilarities between the newly formed cluster and other clusters, the resulting time complexity is $O(m)$ in this case. On the other hand, we analyzed the time complexity in the case of complete networks with n nodes. The complexity equals $O(n\Delta^2) = O(n^3)$ in this case. We get such complexity as the algorithm starts to build a chain in one of nodes in the incremental line subgraph, i.e., at one of the link clusters in the input network and in the worst-case scenario continues throughout all of its adjacent nodes or link clusters around a common active vertex in the input network, respectively. In each step the algorithm has to determine the link cluster which is the least dissimilar to the last link cluster in the chain. The complexity of this step is $O(\Delta) = O(n)$ as each link cluster in the complete graph is adjacent to $2(\Delta - 1) = 2n - 2$ other link clusters in the beginning, giving the step $O(\Delta) = O(n)$. In the course of the algorithm, the number of adjacent link clusters to other link clusters in average linearly decreases to 0 due to their joining. In average the number of adjacent link clusters is therefore proportional to n keeping the $O(n)$ complexity of the step. The algorithm therefore needs $O(n^2)$ time to cluster each node and while there is n nodes in a complete network, the overall time complexity is equal to $O(n\Delta^2) = O(n^3)$. While the calculation of the dissimilarity between a pair of link clusters can take more than $O(1)$ time, the overall complexity of the algorithm may increase.

In other cases we measured the time complexity statistically. We used the dissimilarity measure D_{SF}. Using the keyword dissimilarity measure, described in Sec. V A 2, the complexity would increase proportionally to the number of

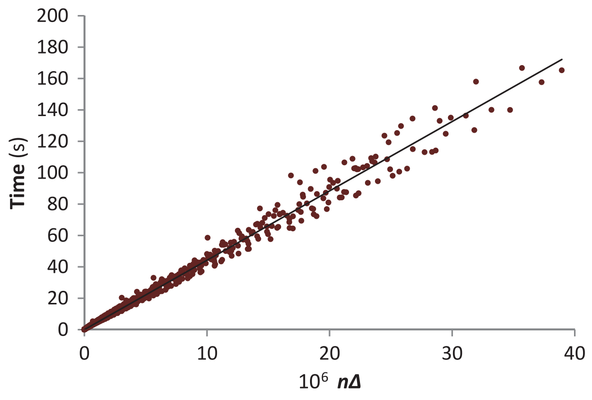

FIG. 3. (Color online) A diagram of running times of the adapted algorithm in relation to the product of the number of nodes and the maximal degree (Δ) in random generated scale-free networks. Each node had its own single unique keyword.

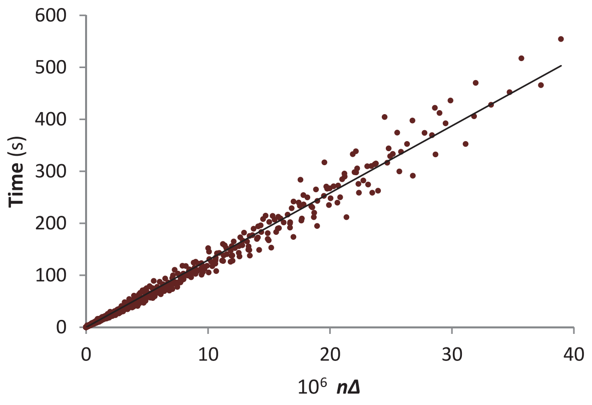

FIG. 4. (Color online) A diagram of running times of the adapted algorithm in relation to the product of the number of nodes and the maximal degree (Δ) in random generated scale-free networks. Each node had 10 random keywords of a set from 200 keywords.

keywords. On the same generated scale-free networks, as for the basic algorithm, we measured the required times for the adapted algorithm to finish. We plotted the complexity measure in terms of the network size where m stands for the number of links and n for the number of nodes in the input network. Results are displayed in Figs. 3 and 4. The results are quite similar. The computation time mostly varies depending on the structure of input networks, which were the same in both cases. The larger number of keywords, however, significantly slows down the algorithm in the second case (by $\sim$3.3 times), but the slowdown is almost even. The estimated complexity of the algorithm on both sets of networks is given by $O(n\Delta)$. We are going to see that scale-free networks mostly contain a small amount of high-degree nodes, yet these mostly increase the time complexity. Based on observations and reasoning, the estimation is therefore sensible. The maximal length of any chain in case of any of the networks we ran the algorithm on was 15 clusters. The largest incremental line subgraph contained 13 625 link clusters and 4 930 881 dissimilarities between them.

Some tests were performed on real-life networks as well. We took the Amazon product copurchasing network, collected by Yang *et al.* [4]. We generated four subnetworks and measured the time for the adapted algorithm to complete each of them. The subnetworks are the largest components in the original network on the subset of nodes having degrees higher than a given threshold. The largest component on nodes with a degree of 2 or more is approximately 20% smaller than the original network. We give the average computation times in Table I. The measured times are relatively high, but we have to consider the fact that the largest incremental subgraph the algorithm constructed (the most critical part) was composed of less than 50 000 link clusters and 400 000 dissimilarities between them only. In our implementation of the algorithm this resulted in less than 100 MB of memory. Note that if the incremental line subgraph would not be freed each time the chain was totally consumed, the time required for the algorithm to finish would significantly decrease.

It is known that scale-free networks can have quite large maximal degrees [24], but the number of so-called hubs, i.e., nodes with large degrees, is usually small. The number of nodes p_k of varying degrees k follows the power-law distribution [24]: $p_k \sim k^{-\gamma}$, where γ is the exponent of the degree distribution whose value is typically in the range $2 < \gamma < 3$. Upper limits of complexities of our algorithm are based on the maximal degree in a network, but such high complexity takes place only if the majority of nodes would have high degrees. This is unusual in scale-free networks. To get a feeling of the structure of some scale-free networks, we present some in Table II: The above-mentioned Amazon network and Brightkite location-based online social networks were obtained from SNAP Datasets [27], the lexical network of

TABLE I. Running times of the adapted algorithm on the largest components $S_1, \ldots, S_4$ in the Amazon network on nodes with a limited minimal degree. The longest chain in any of the upper examples was 24 clusters long. The shortest in the group of the longest chains for each individual example contained 14 clusters. The highest number of stored dissimilarities (links) in the incremental line subgraph among all upper examples was 345 764. In that particular case there were also stored the most clusters (nodes), i.e., 47 525. The computer used to run the algorithm had a 2.5-GHz Intel Core2Duo processor and 4 GB of RAM.

Network	S_1	S_2	S_3	S_4
Minimal degree	10	5	3	2
Maximal degree	52	172	298	361
Number of nodes, n	21 765	145 592	268 380	309 154
Number of links, m	48 077	455 093	822 523	900 163
Average time (one unique keyword per node) (s)	17	1198	4417	5401
Avg. time (10 random keywords from a set of 200 per node) (s)	26	1205	4418	5486

TABLE II. Properties of some demonstration scale-free networks.

Network	n	m	Δ	γ
Amazon	334 863	925 872	549	3.0
Brightkite	58 228	428 156	2268	1.9
Wordnet	146 005	656 999	1008	2.2
Coauthorship network TI	120 953	209 522	382	2.6
CPNS	61 095	71 651	1167	2.7

words from the Wordnet data set was constructed by Fellbaum [28], the coauthorship network in the field of topological indices (TI) was constructed by Bodlaj and Batagelj [29], and a product copurchasing network (CPNS) from one Slovenian online store was partially collected for this work.

1. Space complexity of the adapted algorithm

The space complexity of the adapted algorithm is mostly dependent on the maximal length l of any chain and the incremental line subgraph constructed around it in the course of the algorithm. In the worst case it can happen that the chain is built along all the links in the input network and a whole line graph is built with it. Further, if the input network is a complete network, the chain itself will take $O(m) = O(n^2)$ space in this case and the corresponding line graph $O(n^3)$ space. This is the worst space complexity of the adapted algorithm. While large and dense networks are less common in practice and the chains built by the algorithm are mostly very short, the expected space complexity is significantly lower than $O(n^3)$. A common type of networks to be analyzed with our algorithm belongs to scale-free networks where a small amount of nodes only has a substantially high degree, close to the maximal degree Δ of the network. If one runs the algorithm on such a network, sooner or later the algorithm will encounter one of the links that has a high-degree end node. When this happens, the algorithm will construct a complete subgraph with $O(\Delta^2)$ dissimilarities (links) in the incremental line subgraph. While the chains are short in practice $l \ll m$ and the number of high-degree nodes is relatively low, the expected space complexity of the adapted algorithm in scale-free networks is in the range of $O(l\Delta^2)$, $l \ll m$. Whenever the chain is rebuilt, the current incremental line subgraph is disposed.

Additional space needed to run the algorithm efficiently is the space required to store lists of clusters which share each active node. A node with degree Δ can be shared among at most Δ clusters. In the worst case we have n such nodes in the network, meaning $O(n\Delta)$ space is needed additionally for the entire network at most. This does not contribute to the overall worst space complexity but can represent the major part of space complexity in specific cases.

VII. ALGORITHM APPLICATION ON NETWORKS AND EVALUATION

The main issue with evaluating the suitability of results of community detection algorithms is in lack of strict definition what a community actually is Ref. [4]. As long this is the case, it will remain impossible to uniformly compare results of clustering algorithms side by side.

In the optimization criterion, such as in modularity-based algorithms, the notion of community is operationalized by the criterion itself. In terms of some other criterion-based algorithm, the evaluation of the results of the first algorithm using the comparison of clusters obtained by both algorithms is therefore questionable. Algorithms for community detection were therefore mostly tested on small empirical networks and on synthetic benchmark networks [30]. Tests mostly targeted features of the algorithms, for which the algorithms were designed in the first place. Essentially, we get the answer to the question regarding which criterion function is better for a given task. A comparison of nonhierarchical algorithms with hierarchical ones represents another issue. It can be performed on the basis of the partition, obtained by a hierarchical clustering algorithm and having the highest *partition density* [15]. However, we have to take into account that clusterings of hierarchical algorithms might still fit better than the nonhierarchical ones. The same applies to overlapping communities. G. Tibély *et al.* [30] try to address these issues.

Our algorithm considers also network attributes. For these reasons it is rather hard to compare the results of existing algorithms with results of our new one. For the evaluation of the results of our algorithm we therefore use a straightforward comparison of clusterings [31] and a common-sense approach using the real-world networks which were characterized by humans in advance [4]. We try to compare the obtained clusters to the ground-truth clusters and empirically interpret the results. A quite extensive evaluation of the link clustering algorithm was given in Lehmann's work [2]. Using their dissimilarity measure D_L (23), our algorithm essentially returns the same results as their algorithm.

A. Artificial demonstration network

To demonstrate the algorithm's ability to cluster similar regions in the network, we constructed a small network, containing 18 nodes and 31 links arranged into densely connected regions around a central (18th) node, Fig 5. Nodes in the upper region were assigned keywords $\{B,U\}$ on the left part and keywords $\{B,V\}$ on the right part. The bottom region consists of three parts, whereby nodes in the left part have $\{A\}$, the middle part is without keywords, and the right part has a keyword $\{C\}$. The interconnecting node is without keywords as well. The network is displayed three times in Fig. 5. It has color coded (and surrounded by broken lines) link clusters, determined by the cut intervals in the hierarchy, produced by the algorithm from the network using the dissimilarity measure D_{SC} from Eq. (12). Nodes are labeled with a number, followed by a string of keywords. The flattened hierarchy is shown in Fig. 6. Flattening was performed from the root of the original binary hierarchy by recursively moving up upon both branches of a child branching to the parent's level when the similarity in the parent's branching was equal to the similarity in the child's branching. We expected the algorithm to detect the connected regions with similar sets of keywords. From the diagram of hierarchy this is clearly evident; however, straight cuts across the hierarchy result in joint clusters with less different keywords, following the used dissimilarity measure D_{SC}. At first, every link is in its own cluster (not shown on Fig. 5), and then, at dissimilarity level 1, neighboring links with nodes with

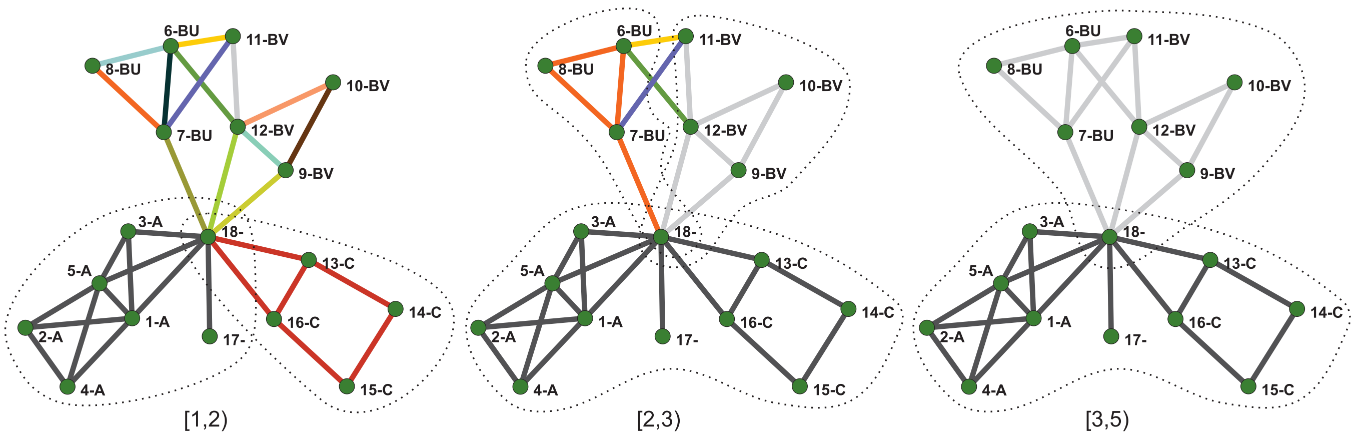

FIG. 5. (Color online) Three clusterings on the demonstration network. Nodes are labeled with their numbers followed by a string of their keywords. Clusters obtained by cutting the hierarchy (Fig. 6) produced by the algorithm on the intervals below each network sketch are color coded and surrounded by broken lines. In the top region of the leftmost clustering, each link is in its own cluster.

the same single keyword are joined into clusters. Note the link $(17- : 18-)$ which could be clustered into either one-keyword cluster (cluster with $\{A\}$ or cluster with $\{C\}$). Clusters of links with two distinct keywords are then joined at dissimilarity level 2, resulting in joining of links with $\{B,U\}$ into a common cluster and $\{B,V\}$ into the other one. Clusters of links with $\{A\}$ and $\{C\}$ are clustered into a joint cluster with $\{A,C\}$ as well. At dissimilarity level 3, clusters with three different keywords are joined. At level 5 all links are finally clustered into a single cluster.

The dissimilarity D_{SC} favors joining smaller clusters which inherently have smaller number of different keywords. Only the number of different keywords is important, regardless of the number of common keywords. For instance, the dissimilarity of $\{A\}$ and $\{B\}$ is 2 and dissimilarity of $\{B,U\}$ and $\{B,V\}$ is 3.

Now let us consider the same example network using dissimilarity D_{SF}. The visualization of the resulting hierarchy is displayed in Fig. 7. For each distinct clustering based on all possible cut values in the resulting hierarchy, we used the measure δ to determine the difference between calculated clusterings $\mathcal{C}_k = \{C_{k,1}, C_{k,2}, \ldots, C_{k,|\mathcal{C}_k|}\}$ and the ground-truth clustering $\mathcal{G} = \{G_1, G_2, \ldots, G_{|\mathcal{G}|}\}$. Differences are shown in Fig. 8. The measure of clustering difference $\delta(\mathcal{C}_k,\mathcal{G})$ is known as the *Best Match* [31]. Using μ to represent the number of moves necessary to convert cluster C into C':

$$\mu(C,C') = |C \oplus C'| \tag{24}$$

we can compute for each cluster $C_{k,i} \in \mathcal{C}_k$ its best representative $G^* \in \mathcal{G}$ such that $G^* = \mathrm{argmin}_{G\in\mathcal{G}}\,\mu(C_{k,i},G)$. Note that by normalizing $\mu(C,C')$ by $|C \cup C'|$ we obtain the Jaccard dissimilarity $D_J(C,C')$ given by Eq. (2). By summing numbers of moves to convert each member of $\mathcal{C}_k$ to its respective best representative in $\mathcal{G}$ we can compute how well $\mathcal{G}$ represents $\mathcal{C}_k$. To make the measure symmetric we also sum the numbers of moves from every member of $\mathcal{G}$ to its corresponding best representative in $\mathcal{C}_k$ and obtain:

$$\delta(\mathcal{C}_k,\mathcal{G}) = \sum_{C\in\mathcal{C}_k} \min_{G\in\mathcal{G}} \mu(C,G) + \sum_{G\in\mathcal{G}} \min_{C\in\mathcal{C}_k} \mu(G,C). \tag{25}$$

Best Match and other measures were developed to quantitatively determine the difference between two possibly overlapping clusterings. The ground-truth clustering of nodes in our example network was determined by five clusters: nodes with the keyword $\{A\}$, nodes with keywords $\{B,U\}$, nodes with $\{B,V\}$, and nodes with $\{C\}$, regardless of connectivity. Those without keywords were assigned to the fifth separate cluster.

Using the evaluation on the demonstration network we are looking to get reference values of the measure of difference

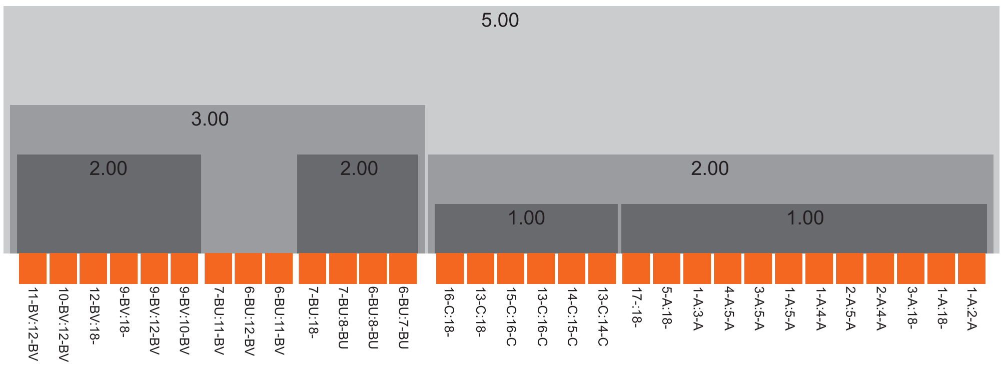

FIG. 6. (Color online) Flattened hierarchy of links obtained by the adapted algorithm using dissimilarity D_{SC} on the demonstration network. Dissimilarity values are displayed at the top of branchings. Links are displayed on the bottom by pairs of linked network nodes.

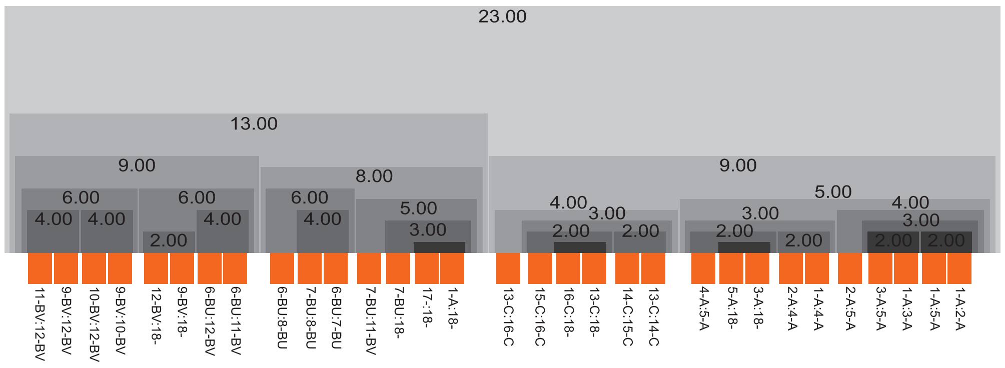

FIG. 7. (Color online) Flattened hierarchy of links obtained by the adapted algorithm using dissimilarity D_{SF} on the demonstration network. Dissimilarity values are displayed at the top of branchings. Links are displayed on the bottom by pairs of linked network nodes.

among the best clustering produced by the algorithm and the ground-truth clustering, so we can compare them with differences of clusterings obtained on larger and real networks. One such diagram is displayed in Fig. 9. Note the explanation of diagrams in Figs. 8 and 9.

B. Real-world network example: Network of online-store-based products

The network was obtained from an online store, selling various equipment. A similar approach was used in Yang's work [4] from Amazon where they also construct the network based on copurchased items. As ground-truth communities they use the hierarchy of categories into which items are classified. Each product is characterized by attributes like color, type, size, shape, and so on. In our case, products were instances of electronic measuring equipment, mainly oscilloscope probes of various types. Each product was categorized into one of four types. We constructed the network of these products. We connected them by links to other products following the relation, based on the behavior of customers viewing other products as well. If a sufficient number of customers were interested in a specific product, and they were interested in another product as well, this second product was listed in the first product's "others viewed" list. On the network we ran the adapted algorithm and, using observation, searched for the appropriate threshold to cut the obtained hierarchy in such a way that the resulting clusters covered the most suitable regions in the network. The comparison results are given by the diagram in Fig. 9.

Figures 8 and 9 support our expectations. If we cluster all links into only one cluster (on the left in both diagrams, the cut value is zero) the difference to the ground truth is high. Then, as we cut hierarchies at higher values, we progressively reach the optimal cut, where the obtained clusters are the most similar to the ground-truth clusters. Increasing the cut value towards its maximum, where each link is clustered into its own cluster, the difference to the ground-truth clusters again increases.

For demonstration purposes we considered only the uniform level cuts in hierarchies. If one would, however, consider variable level cuts (by cutting branches at different levels in the hierarchy), the difference of clusterings obtained at the optimal selection of levels to ground-truth clusterings might improve.

C. Real-world network example: Collaboration between authors in the field of topological indices

In Ref. [29] we investigated bibliographical networks on the field of topological indices constructed from the Web of Science data. Networks were constructed by considering works, authors of works, journals where works were published, and keywords authors used in their works. One of the networks

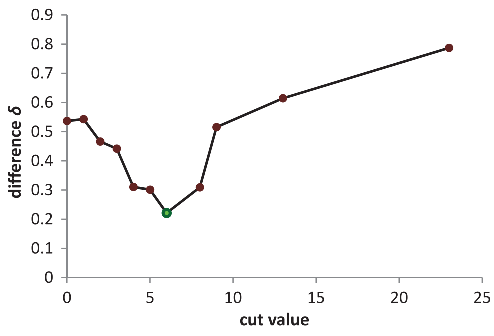

FIG. 8. (Color online) A diagram of differences (δ) between each clustering obtained by a given cut value in the hierarchy generated on the demonstration network by the adapted algorithm and the ground-truth clustering. The optimal cut takes place at value 6 and $\delta = 0.221$. It is marked with a bigger bullet in the diagram.

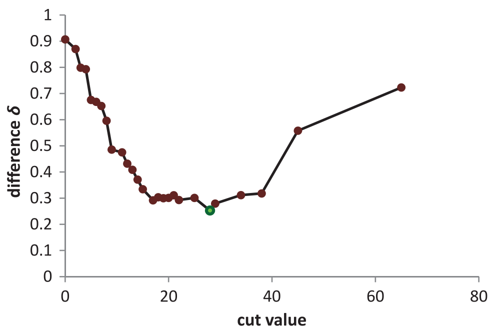

FIG. 9. (Color online) A diagram of differences (δ) between each clustering obtained by a given cut value in the hierarchy generated on the network of oscilloscope probes by the adapted algorithm, and the ground-truth clustering. The optimal cut takes place at value 28 and $\delta = 0.253$. See the emphasized dot in the diagram.

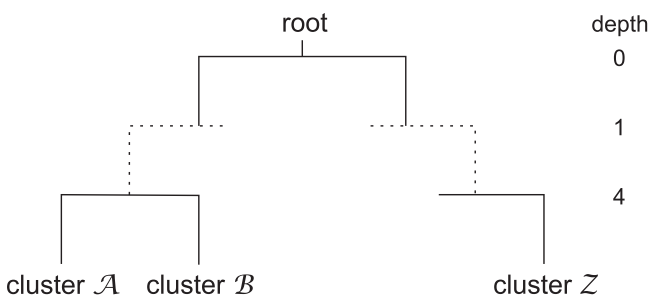

FIG. 10. Scheme of locations of clusters of authors in the hierarchy from the author collaboration network.

studied was a collaboration network of authors where authors are represented by nodes and their collaborations in terms of at least one common work are represented by links. Based on keywords of works, we equipped each node (author) with the set of weights (frequencies) for each keyword the author used in articles he or she (co-)authored. Weights were obtained by *tf-idf* ranking [32]. Based on sets of frequencies of keywords authors used, we then applied the adapted algorithm to construct the hierarchy of similar authors. For an illustration we show three sets of keywords belonging to arbitrarily chosen branches in the obtained hierarchy as displayed in Fig. 10; sets $\mathcal{A}$ and $\mathcal{B}$ from common branchings and the set $\mathcal{Z}$ from a remote branching ($|\mathcal{A}| = 491$, $|\mathcal{B}| = 533$, $|\mathcal{Z}| = 300$). As expected in comparison to set $\mathcal{Z}$, sets $\mathcal{A}$ and $\mathcal{B}$ have a quite similar structure of keywords which can be observed in Fig. 11: $|\mathcal{A} \cap \mathcal{B}| = 227$, $|\mathcal{A} \cap \mathcal{Z}| = 59$. Even if the set $\mathcal{Z}$ would be twice its size, its actual size is approximately a half of the size of sets $\mathcal{A}$ or $\mathcal{B}$, the number of keywords in its intersection with either of sets $\mathcal{A}$ or $\mathcal{B}$ would still contain a half of that of $\mathcal{A} \cap \mathcal{B}$ only. Now let us point out the following. While we can expect the strong resemblance of sets corresponding to branchings close together in the hierarchy, the opposite is not always the case. Even though sets corresponding to distant branchings in the hierarchy are mainly distinct, based on keywords, one could have two similar regions in the network, yet separated by a long path. Due to the relational constraint of the algorithm, the hierarchy generated in this case would have two apparent distinct branches, yet their accompanying sets of keywords would be similar.

Keywords in green (darker) color on the leftmost and on the middle set on Fig. 11 correspond to keywords from the intersection $\mathcal{A} \cap \mathcal{B}$ of clusters $\mathcal{A}$ and $\mathcal{B}$. Dark keywords on the rightmost set correspond to the cluster $\mathcal{A} \cap \mathcal{Z}$. Authors with their short descriptions [29] corresponding to individual clusters can be found in Table III.

VIII. DISCUSSION

While the adapted algorithm works primarily on undirected networks, it can easily be turned into the variation, considering directed networks. When joining the closest clusters in terms of dissimilarity we could consider additional relational constraints [33]. The default algorithm preserves the weak connectivity. Two clusters of links are joined in a step of the algorithm only if they are linked, i.e., share at least one common node. Therefore the weak connectivity is maintained. If, in addition to this, more rules are considered, for instance, connectivity to a single center or strong connectivity, the direction of links becomes important.

In Sec. V we discussed dissimilarity measures. Our focus was mostly on the keyword- or tag-based dissimilarity measures, for example, in the author collaboration network. Let us stress once more that dissimilarity measures can be chosen arbitrarily, based on various properties of networks, trying to express as well as possible the clustering goal. While recursively computed measures only can be based on dissimilarities between individual elements of a network, directly computed dissimilarity measures can consider also specific properties of groups of network elements, constructed in the course of the algorithm, their substructure, for example. We must, however, take a special care to preserve direct dissimilarity measures monotonic. Standard approaches for recursive dissimilarity measures already assure monotonicity.

Monotonicity is an important property of dissimilarity measures as hierarchies, obtained by their use, regardless of the

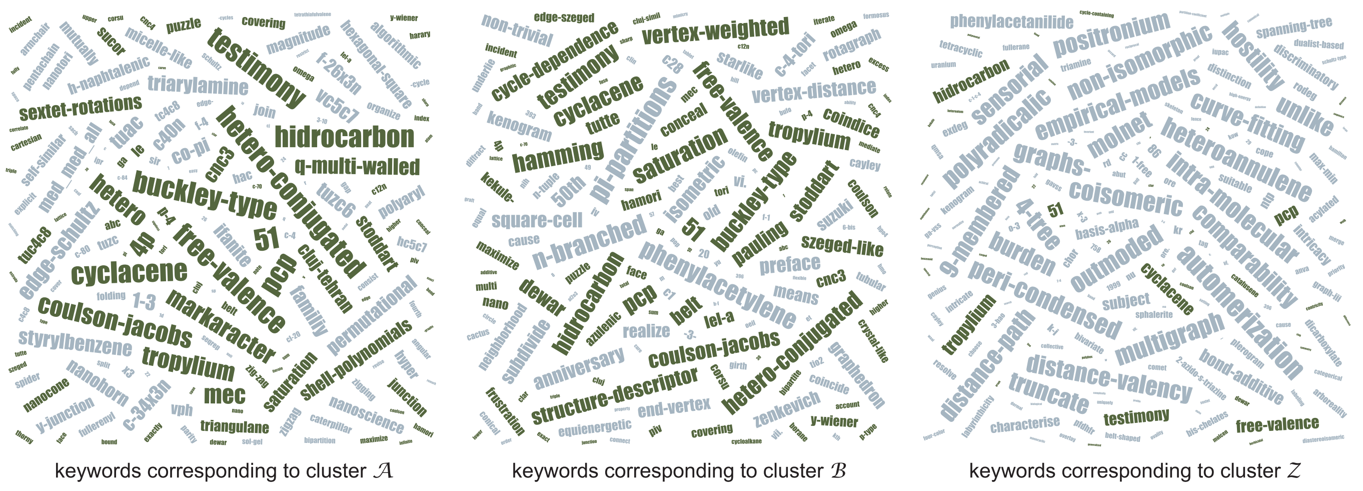
keywords corresponding to cluster $\mathcal{A}$
keywords corresponding to cluster $\mathcal{B}$
keywords corresponding to cluster $\mathcal{Z}$

FIG. 11. (Color online) Sets of keywords corresponding to selected clusters in the hierarchy from the author collaboration network. Keywords in green (darker) color on the leftmost and on the middle set correspond to keywords from the intersection of clusters $\mathcal{A} \cap \mathcal{B}$. Darker keywords on the rightmost set correspond to the cluster $\mathcal{A} \cap \mathcal{Z}$.

TABLE III. List of authors corresponding to individual clusters. Note the author I. Gutman (GUTMAN_I) in all three clusters.

$\mathcal{A}$	AHMADI_A, AIRES-DE_J, ALIPOUR_M, ALIZADEH_Y, AMINI_K, AREZOOMA_M, ASHRAFI_A, AZAD_A, AZARI_M, BADAKHSH_L, BAHRAMI_A, CIOSLOWS_J, CVETKOVI_, DARAFSHE_M, DAS_K, DOROSTI_N, DOSLIC_T, DU_Z, DURDEVIC_J, ELBASIL_S, ELIASI_M, ESTRADA_E, FAGHANI_M, FARHAMI_P, GHAZI_M, GHOLAMI-_F, GHOLAMI_N, GHOLIZAD_S, GHORBANI_M, GILANI_A, GODSIL_C, GOJAK_S, GUTMAN_I, HAMZEH_A, HEMMASI_M, HEYDARI_A, HOSSEIN-_S, HOSSEINZ_M, HOU_Y, ILIC_A, IRANMANE_A, JADDI_M, JALALI_M, KAFRANI_A, KATONA_G, KHAKI_A, KHAKSAR_A, KHALIFEH_M, KHORMALI_O, LAM_P, LI_X, LINERT_W, LOGHMAN_A, MAHMIANI_A, MAIMANI_H, MANDLOI_M, MANOOCHE_B, MATELJEV_M, MESGARAN_H, MILUN_M, MINAILIU_O, MIRZAIE_S, MIRZARGA_M, MOGHARRA_M, NADJAFI-_M, NAGY_C, NAGY_K, NIKZAD_P, PAKRAVES_Y, PARV_B, PESEK_I, POP_M, RADENKOV_S, REZAEI_F, ROUVRAY_D, SAATI_H, SABAGHIA_H, SAHELI_M, SALEHI_N, SEYEDALI_S, SHABANI_H, SHAKERAN_S, SHI_Y, SHIU_W, SOLEIMAN_B, STANKOVI_S, TAERI_B, TAHERKHA_B, TEHRANIA_A, TOMOVIC_Z, TRINAJST_, WAGNER_S, YAN_W, YANG_B, YAZDANI_J, YEH_Y, YOUSEFI-_H, YOUSEFI_S, ZAEEMBAS_A, ZERAATKA_M, ZHOU_B, ZIGERT_P, ZIVKOVIC_T
$\mathcal{B}$	ABELEDO_H, AOUCHICH_M, ARAUJO_O, ARSIC_B, ASHRAFI_A, CAPOROSS_G, CHEPOI_V, CIGHER_S, CLARK_L, CMILJANO_N, DAS_K, DELAPENA_J, DOBRYNIN_A, DOMOTOR_G, DOSLIC_T, DU_Z, DURDEVIC_J, FATH-TAB_G, FENG_L, FURTULA_B, GHOLAMI-_F, GOJAK_S, GORDEEVA_E, GRAOVAC_A, GUEVARA_N, GUTMAN_I, HAMZEH_A, HANSEN_P, HEMMASI_M, HOSSEIN-_S, HOSSEINZ_M, ILIC_A, ILIC_M, JUVAN_M, KARBASIO_A, KLAVZAR_S, KOVSE_M, LEPOVIC_M, LINERT_W, LIU_B, LO_S, LUO_Y, MARKOVIC_Z, MARSHALL_G, MEL'NIKO_L, MELOT_H, MILJKOVI_O, MILOSAVL_S, MOGHARRA_M, MOHAR_B, MORALES_D, MOTOC_I, NADJAFI-_M, NAGY_C, PAVLOVIC_L, PESEK_I, PLAVSIC_D, POP_M, POPOVIC_L, RADA_J, RADENKOV_S, RAJAPAKS_A, SAATI_H, SAHELI_M, SALEM_K, SEITZ_W, SHAO_J, SHRIVAST_A, SOSKIC_M, STANKOVI_S, STEVANOV_D, TOMOVIC_Z, URSU_O, VIDOVIC_D, VIZITIU_A, XU_K, YARAHMAD_Z, YU_G, ZEROVNIK_J, ZHANG_F, ZHAO_H, ZIGERT_P
$\mathcal{Z}$	AGRAWAL_V, BABIC_D, BALABAN_A, BALABAN_T, BASAK_S, BRISSEY_G, GRUNWALD_G, GUTMAN_I, HARARY_F, IVANCIUC_O, JAKLIC_G, KHADIKAR_P, KOPECKY_K, KRILOV_G, LEPOVIC_M, LERS_N, MEDELEAN_M, MILICEVI_A, MINAILIU_O, NATARAJA_R, PAVAN_M, PISANSKI_T, POMPE_M, POPOVIC_L, RANDIC_M, RUCKER_C, SABLJIC_A, STANKOVI_S, TRINAJST_N, VIDOVIC_D, VRACKO_M, VUKICEVI_D, YAMAGUCH_T

cut value, assure uniquely defined clusterings. Monotonicity is also important for our adapted algorithm as its correctness depends on it. It is important also for aspects of the clustering algorithm, discussed in the following sections, i.e., regarding the algorithm stability (Sec. VIII A) and parallelization of the algorithm (Sec. VIII B). While standard approaches for recursively computed dissimilarity measures assure monotonicity, this is not always the case with directly computed dissimilarity measures.

A. Stability of the adapted algorithm

If the adapted algorithm encounters multiple alternative choices of clusters with equal dissimilarities at the end of the current chain, the first cluster in regard to the order in which the links were loaded into the internal network structure will be greedily appended to the chain. The sequence of equidistant clusters appended to the chain is operationalized by the sequence of node and link definitions in the input network file and the implementation of data structures the algorithm is based on. In general clustering, however, when there are ties in the dissimilarities between clusters [34], hierarchies can no longer be uniquely determined.

In terms of stability, we observed how much the results differ if the algorithm is run multiple times on the same network or on the same network with a permuted sequence of nodes and links. The network used was the authors collaboration network on the field of topological indices, see Sec. VII C. In this particular case, the algorithm returned hierarchies that were equal up to the permutation of their branches. This means that the sole order of branches in hierarchies differed, but, structurally, all resulting hierarchies were equal, i.e., they could be rearranged into one another by reordering the branches in their branchings only. Dissimilarity values in their branchings were equal as well. All results formally give the same canonical form of the hierarchy. To some extent this indicates that even if the input network is relatively large and has enough distinct keywords assigned to its nodes or links, the algorithm can return consistent results. Using, for example, a less discriminating dissimilarity measure on perhaps a different network can, however, lead to less consistent, yet similar, results. A more detailed analysis of the stability was not performed.

A convenient addition to the algorithm would be a process to detect the state when multiple equally similar pairs of nodes or links are available for the algorithm to choose and to notify the user at the end that multiple solutions exist. An even better solution to the problem was discussed by Fernández and Gómez [35], who propose implementing in the course of clustering the implicit combining of neighboring branchings with equal dissimilarities in the hierarchy into combined branchings. Instead of obtaining a hierarchy with only the two-branch branchings, it is then possible to get a hierarchy with fewer branchings, but a number of branches in individual branchings can be larger than two. This kind of "flattened" (see Sec. VII A) hierarchy can be represented with a multidendrogram [35].

B. Parallelization

Due to the relational constraint in the algorithm, remote regions in a source network cannot be joined into a common cluster in fewer iterations of the algorithm than the length of the shortest path between those regions. We could therefore split the algorithm to work in multiple threads to cluster remote regions in parallel. The parallel algorithms, the basic and the adapted, should, however, use a monotonic dissimilarity measure to produce consistent results. For the adapted algorithm this is already required. The problems would arise otherwise when combining partial hierarchies, generated by individual threads. A similar approach is mentioned as a *node-centric* approach in Yang's work [4].

IX. CONCLUSION

In this work we presented a basic agglomerative algorithm for clustering nodes in a network and proposed its adapted version for clustering links in a network. We presented a few compatible dissimilarity measures as a necessary part of both algorithms to perform their task. We divided them into groups of recursively and directly computed dissimilarity measures and propose some new ones, designed to consider, in addition to network structure, the attributes of the network, enabling both algorithms to increase accuracy of deeper levels in the resulting hierarchies. In the synthetic example and real networks we then illustrated how algorithms work, analyzed their complexity, and showed that they can produce reasonable results. We also discussed their real-life applicability. In Appendix B we give some information on our implementation of the adapted algorithm.

The algorithm is available for download as a part of Abelium's network analysis library net.Plexor [36]. The source code of the algorithm can also be obtained from the authors.

ACKNOWLEDGMENTS

This work was in part financed by Slovenian Research Agency (ARRS), Grant No. J5-5537, as well as by a grant within the EUROCORES Programme EUROGIGA (Project GReGAS) of the European Science Foundation. The first author was financed in part by the European Union, European Social Fund, Slovenian Ministry of Education, Science and Sport and in part through operation Competence Center KC CLASS, financed by the European Union, European Regional Development Fund.

APPENDIX A: PROOFS FOR THE MONOTONICITY OF DISSIMILARITY MEASURES

1. Dissimilarity measures D_{SF}, D_{SC}, and D_{MF} are monotonic

Let us consider having three general clusters of nodes or links C_p, C_q, and C with respective descriptions with keywords F_{C_p}, F_{C_q}, and F. We now presume the algorithm just joined clusters C_p and C_q. The description of a joint cluster $C_p \cup C_q$ is then $F_{C_p} \sqcup F_{C_q}$.

a. *Monotonicity of D_{SF}*

To prove that D_{SF} has the reducibility property (1) we express the assumption $D_{\mathrm{SF}}(C_p,C_q) \leqslant D_{\mathrm{SF}}(C_p,C)$ for the dissimilarity D_{SF}:

$$\sum_{k\in K} \max\left(f_{C_p,k}, f_{C_q,k}\right) \leqslant \sum_{k\in K} \max\left(f_{C_p,k}, f_{C,k}\right), \tag{A1}$$

since $f_{C_q,k} \geqslant 0$, $f_{C_p,k} \leqslant f_{C_p,k} + f_{C_q,k}$ holds and therefore

$$\leqslant \sum_{k\in K} \max\left(f_{C_p,k} + f_{C_q,k}, f_{C,k}\right) = D_{\mathrm{SF}}(C_p \cup C_q, C), \tag{A2}$$

which gives $D_{\mathrm{SF}}(C_p,C_q) \leqslant D_{\mathrm{SF}}(C_p \cup C_q, C)$. The dissimilarity D_{SF} is monotonic. ■

b. *Monotonicity of D_{SC}*

To prove that D_{SC} has the reducibility property (1) we express the assumption $D_{\mathrm{SC}}(C_p,C_q) \leqslant D_{\mathrm{SC}}(C_p,C)$ for the dissimilarity D_{SC}:

$$\sum_{k\in K} \max\left(\mathrm{gt}(f_{C_p,k}), \mathrm{gt}(f_{C_q,k})\right) \leqslant \sum_{k\in K} \max\left(\mathrm{gt}(f_{C_p,k}), \mathrm{gt}(f_{C,k})\right), \tag{A3}$$

since $f_{C_q,k} \geqslant 0$, $f_{C_p,k} \leqslant f_{C_p,k} + f_{C_q,k}$ holds and therefore

$$\leqslant \sum_{k\in K} \max\left(\mathrm{gt}(f_{C_p,k} + f_{C_q,k}), \mathrm{gt}(f_{C,k})\right) = D_{\mathrm{SC}}(C_p \cup C_q, C), \tag{A4}$$

which gives $D_{\mathrm{SC}}(C_p,C_q) \leqslant D_{\mathrm{SC}}(C_p \cup C_q, C)$. The dissimilarity D_{SC} is monotonic. ■

c. *Monotonicity of D_{MF}*

To prove that D_{MF} has the reducibility property (1) we express the assumption $D_{\mathrm{MF}}(C_p,C_q) \leqslant D_{\mathrm{MF}}(C_p,C)$ for the dissimilarity D_{MF}:

$$\max_{k\in K}\left(f_{C_p,k}, f_{C_q,k}\right) \leqslant \max_{k\in K}\left(f_{C_p,k}, f_{C,k}\right), \tag{A5}$$

since $f_{C_q,k} \geqslant 0$, $f_{C_p,k} \leqslant f_{C_p,k} + f_{C_q,k}$ holds and therefore:

$$\leqslant \max_{k\in K}\left(f_{C_p,k} + f_{C_q,k}, f_{C,k}\right) = D_{\mathrm{MF}}(C_p \cup C_q, C), \tag{A6}$$

which gives $D_{\mathrm{MF}}(C_p,C_q) \leqslant D_{\mathrm{MF}}(C_p \cup C_q, C)$. The dissimilarity D_{MF} is monotonic. ■

2. Constrained single-linkage is monotonic

Assuming that for clusters C_p, C_q, and C $D_{\mathrm{mc}}(C_p,C_q) \leqslant D_{\mathrm{mc}}(C_p,C)$ and $D_{\mathrm{mc}}(C_p,C_q) \leqslant D_{\mathrm{mc}}(C_q,C)$, the following holds:

$$\begin{aligned} \min_{(u:v)\in E(C_p,C_q)} d(u,v) &\leqslant \min_{(u:v)\in E(C_p,C)} d(u,v) \quad \text{and} \\ \min_{(u:v)\in E(C_p,C_q)} d(u,v) &\leqslant \min_{(u:v)\in E(C_q,C)} d(u,v). \end{aligned} \tag{A7}$$

Inequalities in (A7) imply:

$$\begin{aligned} &\min_{(u:v)\in E(C_p,C_q)} d(u,v) \\ &\quad \leqslant \min\left(\min_{(u:v)\in E(C_p,C)} d(u,v), \min_{(u:v)\in E(C_q,C)} d(u,v)\right) \\ &\quad = \min_{(u:v)\in E(C_p\cup C_q,C)} d(u,v). \end{aligned} \tag{A8}$$

This proves that the constrained single-linkage dissimilarity has the reducibility property. The constrained single-linkage dissimilarity is therefore monotonic. ■

Note. The proof for the monotonicity of the constrained maximal-linkage dissimilarity (18) follows exactly the same steps. The monotonicity of the traditional single-linkage dissimilarity (13) follows from the monotonicity of the constrained single-linkage on a network N embedded into a complete network (graph) K_n and the monotonicity of Lehmann's dissimilarity (23) follows from the monotonicity of the constrained single-linkage using $d = d_l$.

3. Constrained average-linkage is monotonic

The proof for the monotonicity of the constrained average-linkage (19) follows similar steps like the proof for the constrained single-linkage in Sec. A 2. We assume that for disjoint clusters C_p, C_q, and C:

$$\begin{aligned} &D_{\text{ac}}(C_p,C_q) \leqslant D_{\text{ac}}(C_p,C) \quad \text{and} \\ &D_{\text{ac}}(C_p,C_q) \leqslant D_{\text{ac}}(C_q,C) \end{aligned} \tag{A9}$$

is true. By multiplying both sides of inequalities (A9) by $w[E(C_p,C)]w[E(C_p,C_q)]$ and $w[E(C_q,C)]w[E(C_p,C_q)]$, respectively, we obtain:

$$\begin{aligned} &w[E(C_p,C)] \sum_{(u:v)\in E(C_p,C_q)} d(u,v) \\ &\quad\leqslant w[E(C_p,C_q)] \sum_{(u:v)\in E(C_p,C)} d(u,v) \quad \text{and} \\ &w[E(C_q,C)] \sum_{(u:v)\in E(C_p,C_q)} d(u,v) \\ &\quad\leqslant w[E(C_p,C_q)] \sum_{(u:v)\in E(C_q,C)} d(u,v). \end{aligned} \tag{A10}$$

We then sum both the left and right sides of inequalities (A10) to get a single inequality:

$$\begin{aligned} &\{w[E(C_p,C)] + w[E(C_q,C)]\} \sum_{(u:v)\in E(C_p,C_q)} d(u,v) \\ &\leqslant w[E(C_p,C_q)]\left(\sum_{(u:v)\in E(C_p,C)} d(u,v) + \sum_{(u:v)\in E(C_q,C)} d(u,v)\right), \end{aligned} \tag{A11}$$

which simplifies to:

$$\begin{aligned} &w[E(C_p\cup C_q,C)] \sum_{(u:v)\in E(C_p,C_q)} d(u,v) \\ &\quad\leqslant w[E(C_p,C_q)] \sum_{(u:v)\in E(C_p\cup C_q,C)} d(u,v). \end{aligned} \tag{A12}$$

By dividing both sides of (A12) by $w[E(C_p,C_q)]w[E(C_p \cup C_q,C)]$ we get $D_{\text{ac}}(C_p,C_q) \leqslant D_{\text{ac}}(C_p \cup C_q,C)$ and the constrained average linkage is therefore monotonic. ■

APPENDIX B: ALGORITHM IMPLEMENTATION AND DATA STRUCTURES

Our implementation of the algorithm is as follows. In the beginning every link is put in its own cluster. The algorithm then enters into the main loop where it picks a random link cluster, using it as the beginning of a current chain of link clusters it advances toward. It then builds the chain in the direction of the lowest strictly decreasing dissimilarity between the cluster at the end of the chain and its adjacent clusters in the input network. To aid the process it constructs an incremental line subgraph along the chain in which the links represent dissimilarities between link clusters in the input network. When the algorithm encounters a pair of mutually most-similar clusters (reciprocal nearest neighbors) at the end of the chain, it removes them from that chain, joins them together into a new cluster, and updates the corresponding incremental line subgraph accordingly. It then continues to build the chain from its new ending, and so on. If the chain gets empty, the algorithm starts building a new chain from another randomly chosen unvisited cluster. Along the beginning of each new chain it disposes the current incremental line subgraph and begins building a new one. When ever two clusters get joined, the algorithm creates a new branching in the hierarchy, assigns the clusters to its branches and stores the dissimilarity between them accordingly. The algorithm finishes its work when the last two clusters are joined.

1. Incremental line subgraph

The incremental line subgraph is used as a cache of recently calculated dissimilarities between link clusters in the vicinity of link clusters in a current chain that the algorithm works with. We implemented it using two data structures. The first is a dictionary of dissimilarities between pairs of link clusters from the input network. In principle, the dictionary is used to store the weights of links in the partial line graph, which represent the pairs of adjacent clusters of links in the original network. It provides an efficient access to the cached dissimilarity of a given pair of link clusters. The second structure used is a dictionary of sets of link clusters adjacent to a given link cluster. It is used as an efficient structure to provide direct access to nodes adjacent to a selected node in the partial line graph.

a. Hierarchy representation

Hierarchy is implemented using the data structure, known as the parent pointer list of size $2m - 1$, in the form of an index array. In the beginning, every cluster, i.e., link, is assigned its own cell in the array at index from 1 to m. Then, in every iteration k of the main loop of the algorithm, two clusters being joined adapt the number of the current iteration increased by the number of links in the source network m: $k + m$. The cell in the list with the index $k + m$ is going to be used for the index to the parent of a new joined cluster in one of the following iterations. The cell represents a new branching in the hierarchy. The cells belonging to two joined clusters in the current iteration are filled with the index of their new parent, a joined cluster, $k + m$. In the last iteration of the algorithm, the last two clusters are joined together and their cells in the array are filled with index value or the hierarchy root, which is usually given the value 0.

b. Cluster structure

The cluster consists of a set of links, which were put into a cluster. Besides that, it contains a list of pointers to active nodes in which it is present and the list of frequencies of keywords characterizing it.

c. Structure to store active nodes

Our implementation of the algorithm keeps the list of active nodes. Each active node has its own list of pointers to clusters which interact in it.

[1] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, Proc. Natl. Acad. Sci. USA **101**, 2658 (2004).

[2] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, Nature **466**, 761 (2010).

[3] T. S. Evans and R. Lambiotte, Phys. Rev. E **80**, 016105 (2009).

[4] J. Yang and J. Leskovec, *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12 (ACM, New York, 2012), pp. 3:1–3:8.

[5] S. Fortunato, Phys. Rep. **486**, 75 (2010).

[6] J. Xie, S. Kelley, and B. K. Szymanski, ACM Comput. Surv. **45**, 43 (2013).

[7] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, Nature **435**, 814 (2005).

[8] S. Fortunato and A. Lancichinetti, *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '09* (ICST, Brussels, Belgium, 2009), pp. 27:1–27:2.

[9] J. Baumes, M. K. Goldberg, M. S. Krishnamoorthy, M. Magdon-Ismail, and N. Preston, in *IADIS AC*, edited by N. Guimarães and P. T. Isaías (IADIS, Algarve, Portugal, 2005), pp. 97–104.

[10] S. Gregory, New J. Phys. **12**, 103018 (2010).

[11] J. Xie and B. K. Szymanski, arXiv:1202.2465.

[12] P. D. Meo, E. Ferrara, G. Fiumara, and A. Provetti, arXiv:1108.1502.

[13] A. Ferligoj and V. Batagelj, Psychometrika **47**, 413 (1982).

[14] G.-J. Kim, K.-Y. Whang, M.-S. Kim, H.-S. Lim, K.-H. Lee, and B. S. Lee in *Second International Conference on the Applications of Digital Information and Web Technologies, 2009. ICADIWT'09* (IEEE, London, 2009), pp. 438–445.

[15] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech: Theor. Exp. (2008) P10008.

[16] A. Ferligoj and V. Batagelj, Psychometrika **48**, 541 (1983).

[17] V. Batagelj, A. Ferligoj, and A. Mrvar, Hierarchical clustering in large networks http://perso.uclouvain.be/vincent.blondel/workshops/2008/files/batagelj.pdf (2008).

[18] F. Murtagh, *Multidimensional Clustering Algorithms* (Physica-Verlag, Vienna, 1985), pp. 59–88.

[19] C. de Rham, *La classification hiérarchique ascendante selon la méthode des voisins réciproques, Les Cahiers de l'Analyse des Données*, Dunod, 5 (1980), pp. 135–144, http://www.numdam.org/item?id=CAD_1980__5_2_135_0.

[20] L. Fu, D. Sun, and L. R. Rilett, Comput. Oper. Res. **33**, 3324 (2006).

[21] M. Bruynooghe, Stat. Anal. Données no. 3, 24 (1977).

[22] P. Jaccard, Bull. Soc. Vaud. Sci. Natur. **37**, 547 (1901).

[23] C. Herrera and P. J. Zufiria, in *Proceedings of the 2011 IEEE Network Science Workshop, NSW'11* (IEEE Computer Society, Washington, DC, USA, 2011), pp. 167–172.

[24] A.-L. Barabási, *Network Science*, Chap. 4 (2012), http://barabasilab.neu.edu/networksciencebook/download/network_science_december_ch4_2013.pdf .

[25] A. A. Hagberg, D. A. Schult, and P. J. Swart, in *Proceedings of the 7th Python in Science Conference (SciPy2008)* (Pasadena, CA, USA, 2008), pp. 11–15.

[26] V. Batagelj and M. Zaveršnik, Adv. Data Anal. Class. **5**, 129 (2011).

[27] J. Leskovec and A. Krevl, SNAP Datasets: Stanford large network dataset collection http://snap.stanford.edu/data (2014).

[28] C. Fellbaum, ed., *WordNet: An Electronic Lexical Database* (MIT Press, Cambridge, MA, 1998).

[29] J. Bodlaj and V. Batagelj, Molec. Inform. **33**, 514 (2014).

[30] G. Tibély, L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, Phys. Rev. E **83**, 056125 (2011).

[31] M. K. Goldberg, M. Hayvanovych, and M. Magdon-ismail, in *Proceedings of the 2010 IEEE Second International Conference on Social Computing, SOCIALCOM '10* (IEEE, Minneapolis, Minnesota, 2010), pp. 303–308.

[32] G. Salton and C. Buckley, Inform. Process. Manag. **24**, 513 (1988).

[33] V. Batagelj and A. Ferligoj, *Data Analysis*, edited by W. Gaul, O. Opitz, M. Schrader (Springer, Berlin, 2000), pp. 3–15.

[34] B. J. T. Morgan and A. P. G. Ray, Appl. Stat. **44**, 117 (1995).

[35] A. Fernández and S. Gómez, J. Class. **25**, 43 (2008).

[36] http://www.abelium.eu/razvoj/net.plexor