Clustering sequence graphs

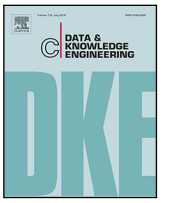
Haodi Zhong, Grigorios Loukides, Solon P Pissis

Data & Knowledge Engineering 138 (2022) 101981

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak



Clustering sequence graphs

Haodi Zhong [a], Grigorios Loukides [a,*], Solon P. Pissis [b,c]

[a] Department of Informatics, King's College London, London, UK
[b] CWI, Amsterdam, The Netherlands
[c] Vrije Universiteit, Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Sequence clustering
Graph clustering
Sequential data

ABSTRACT

In application domains ranging from social networks to e-commerce, it is important to cluster users with respect to both their relationships (e.g., friendship or trust) and their actions (e.g., visited locations or rated products). Motivated by these applications, we introduce here the task of clustering the nodes of a *sequence graph*, i.e., a graph whose nodes are labeled with strings (e.g., sequences of users' visited locations or rated products). Both string clustering algorithms and graph clustering algorithms are inappropriate to deal with this task, as they do not consider the structure of strings and graph simultaneously. Moreover, attributed graph clustering algorithms generally construct poor solutions because they need to represent a string as a vector of attributes, which inevitably loses information and may harm clustering quality. We thus introduce the problem of clustering a sequence graph. We first propose two pairwise distance measures for sequence graphs, one based on edit distance and shortest path distance and another one based on SimRank. We then formalize the problem under each measure, showing also that it is NP-hard. In addition, we design a polynomial-time 2-approximation algorithm, as well as a heuristic for the problem. Experiments using real datasets and a case study demonstrate the effectiveness and efficiency of our methods.

1. Introduction

Graph clustering [1] is a fundamental data mining task, which seeks to partition the nodes of an input graph, so that similar nodes form a group, referred to as cluster. The task is important in application domains such as social networks, where nodes represent users, edges represent friendship relationships between users and clustering aims to detect user communities [2]; or e-commerce, where nodes represent consumers, edges represent trust relationships between consumers and clustering aims to identify groups of consumers with bonds of trust among them [3]. The task is also important in medicine, as clustering patient similarity networks (i.e., graphs whose nodes represent patients and edges connect similar patients according to demographics or genetic mutations) allows identifying clinically homogeneous patient groups [4].

However, often in these application domains, the similarity among users does not depend only on user relationships but also on actions or events associated to users. Take for example a geo-social network, such as Foursquare, where users are associated with their history of visited locations. Two connected users may naturally be regarded as more similar in the network, if they have a similar history of visited locations [5,6]. For example, similarity with respect to both the sequence of visited locations and the social network structure is considered in recommendation [5] and location prediction [6]. Likewise, in e-commerce, two connected users may be regarded as more similar in a trust network, if they have a similar history of rated products [7]. Also, in medicine, two connected users may be regarded as more similar (e.g., with respect to disease progression), if they have a similar history of diagnoses [8].

* Corresponding author.
E-mail addresses: haodi.zhong@kcl.ac.uk (H. Zhong), grigorios.loukides@kcl.ac.uk (G. Loukides), solon.pissis@cwi.nl (S.P. Pissis).

https://doi.org/10.1016/j.datak.2022.101981
Received 19 June 2021; Received in revised form 30 December 2021; Accepted 2 January 2022
Available online 20 January 2022

Thus, motivated by these application domains, we introduce the problem of clustering a graph whose nodes are labeled with sequences of letters (i.e., strings). We refer to such graphs as *sequence graphs*. In the aforementioned application domains, an edge models a relationship between users. Alternatively, an edge in a sequence graph can model a relationship between strings (e.g., two nodes are connected, if their associated genomic sequences share sufficiently many substrings [9,10]).

While natural, the problem of clustering a sequence graph has not been considered before, to the best of our knowledge, although graphs with sequence-labeled nodes are being used extensively in bioinformatics [9]. In addition, existing clustering methods are not appropriate to address the problem. For example, algorithms for clustering an attributed graph [11–18], in which nodes are labeled with a vector of attribute values, are not appropriate for clustering a sequence graph. This is because to apply these algorithms one needs to convert a string into a vector of attribute values, which inevitably loses information and severely harms clustering quality, as we discuss in more detail in Section 2.

Contributions. Our work makes the following specific contributions:

1. We propose two measures to quantify the distance between a pair of nodes in a sequence graph. Both our measures take into account the similarity between node labels (strings) and the structural similarity of nodes; however, in different ways. Our first measure is a product metric [19], based on edit distance and shortest path distance, while our second measure is based on edit distance and SimRank [20]. Shortest path distance is efficient to compute but is based on a single path between two nodes. SimRank is less efficient to compute but takes into account all tours (paths that may have cycles) between two nodes u and v, and it aggregates similarities of multi-hop neighbors of u and v, which helps producing high-quality clustering results. We propose a fixed-point iteration algorithm for computing our SimRank-based measure. We also consider a proxy measure for each of our measures, which approximates edit distance between strings by efficiently embedding them into a Hamming distance space. This allows distance computation in linear time in the length of the two strings.

2. We formalize the problem of clustering a sequence graph under either of our measures, in the context of prototype-based clustering [1], where clusters are built around representative nodes. We consider two versions of our problem. The first is based on the k-center problem [21,22], in which the goal is to minimize the maximum distance between any node and its closest cluster representative. The second is based on the k-median problem [21,23], in which the goal is to minimize the sum of distances between each node and its closest representative. We show that both versions of our problem are NP-hard. For the first version, we design a 2-approximation algorithm and show that no polynomial-time algorithm achieves a better approximation guarantee. For the second version, we design a heuristic.

3. We compare our algorithms against four state-of-the-art attributed graph clustering algorithms [12,16–18], using two datasets from trust-aware e-commerce websites. Our results show that, unlike the competitors, our algorithms create high-quality clusters containing structurally similar nodes with similar strings (e.g., users with trust bonds who are similar with respect to their history of reviewed products). Our results also show that the proxy measures we introduce do not substantially affect quality, while greatly improving runtime efficiency. For instance, our algorithm that utilizes the proxy of the product metric was up to 8 times faster than the most efficient competitor.

4. We present a case study on phylogenetic trees [24]. A phylogenetic tree is often constructed from a set of strings, each representing the genomic sequence of an organism. It is a hierarchical representation of all clusterings of the genomic sequences of the organisms and is often modeled as a binary tree whose leaf nodes are labeled with the strings. Such a tree can thus be seen as a special type of a sequence graph. Given a phylogenetic tree and a positive integer k, the computational task we consider is to cluster the leaf nodes of the tree into k clusters, by taking into account both the tree topology and the sequences corresponding to leaf nodes. We can then evaluate how accurate this clustering is by comparing it to a ground truth clustering. If these two clusterings are similar, then the phylogenetic tree is meaningful. Our results indeed show that the measures we introduce (and the corresponding clustering algorithms) are a reliable way to evaluate whether a given phylogenetic tree is in accordance with a given ground truth clustering consisting of k clusters. On the other hand, we show that four state-of-the-art attributed graph clustering algorithms [12,16–18] are not suitable for this task. The results of the case study also indicate that our methods could potentially be useful in other bioinformatics applications, such as evaluating phylogenetic networks [25].

Paper Organization. The remaining of the paper is organized as follows. In Section 2, we review related work. In Section 3, we define some preliminary concepts. In Section 4 we present our distance measures for sequence graph clustering. In Section 5, we define two sequence clustering problems that we aim to solve and study their hardness. In Section 6, we present our algorithms for addressing these problems. In Section 7, we present an experimental evaluation of our algorithms. In Section 8, we present a case study to showcase the applicability of our algorithms. In Section 9, we conclude the paper.

2. Related work

Our work is related to sequence clustering and graph clustering, two data mining tasks with several applications [8,26–28]. Therefore, in the following, we briefly review algorithms for clustering a collection of sequences (strings), as well as algorithms for clustering a (non-attributed) graph (see [29,30] for surveys). Our work is also related to attributed graph clustering. In an attributed graph, each node is labeled with a vector of attribute values. We therefore review some recent works on attribute graph clustering and refer the reader to [31] for a survey. Last, we discuss the use of string-labeled graphs in bioinformatics.

Sequence Clustering and Graph Clustering. Algorithms for clustering a collection of sequences (strings) measure distance between sequences directly [32], or first project the sequences into a set of patterns (e.g., q-grams) and then measure distances on the

projected space [10,33]. Alternatively, they employ generative models for the input collection of sequences, which are used to obtain likelihood-based distances between sequences [34]. In any case, the distance measures are given as input to a clustering algorithm for vector data [30] to obtain clusters.

Algorithms for clustering a graph employ graph partitioning (e.g., they solve a minimum cut problem [35]), spectral clustering [36,37], or cohesive subgraph detection techniques [38]. Alternatively, the algorithms in [39,40] learn a node embedding into a vector space, which is then fed into a clustering algorithm for vector data.

Algorithms for clustering a collection of sequences [10,33,34], or for clustering a graph [1,38,39] are not appropriate for clustering sequence graphs, as we show in our experiments (Section 7). This is because they utilize either only the strings (i.e., they cluster the collection of labels in a sequence graph while ignoring the graph structure) or only the graph structure (i.e., they cluster a sequence graph while ignoring its labels), although both the strings and graph structure determine clustering quality. Also, we cannot convert a sequence graph to a graph with string distances as edge weights and then cluster the weighted graph. This is because our clustering problem needs distances between strings of nodes that are not connected, and it is generally not possible to compute these distances exactly by combining edge weights.

Attributed Graph Clustering. Algorithms for clustering an attributed graph utilize both the graph structure and the attribute values of nodes [11–18]. They employ, for example, graph convolution [18], matrix factorization [12,13], or attributed graph embedding into a vector space [16,17].

An example of a graph-convolution-based algorithm is Adaptive Graph Convolution (AGC) [18]. The AGC algorithm uses graph convolution to obtain smooth feature representations of node attributes and spectral clustering. The underlying assumption of AGC is that nodes that are close in the graph will be clustered together [18]. An example of a matrix-factorization-based algorithm is Text-Associated DeepWalk (TADW) [12]. This algorithm is based on DeepWalk [39], which uses textual attributes to supervise random walks on graphs. Examples of embedding-based algorithms are Text Enhanced Network Embedding (TENE) [16] and Binarized Attributed Network Embedding (BANE) [17]. Both of these algorithms aim at learning a low-dimensional vector representation for each node and its associated attributes in the attributed graph. BANE uses Weisfeiler–Lehman graph kernels [41] to encode dependencies between node edges and attributes into a binary code representation. This representation encodes first-order proximities [42] between nodes. TENE aims to jointly learn vector representations based on both first-order and second-order proximities [42], as well as on the text cluster membership matrix.

Although one can represent string labels as attribute vectors (e.g., by representing a string as a vector of q-grams and their frequencies [10] or their tf-idf scores [16]) and then apply an attributed graph clustering algorithm to our problem, such a representation inevitably loses information and thus severely degrades the quality of clustering, as we show in our experiments (Section 7). The reason it loses information is because one needs to assume a single order for the q-grams of all sequences to construct the vector representations of all sequences. However, the q-grams do not appear in the same order in all sequences. To illustrate this point, we provide the following example.

Example 1. Let $T_1 = \texttt{aba}$ and $T_2 = \texttt{bab}$ be two sequences. The 2-grams of both of these sequences are $\texttt{ab}$ and $\texttt{ba}$ and each of these 2-grams appears only once in T_1 or in T_2. Thus, T_1 and T_2 have the same vector representation $(1,1)$ where the first 1 denotes the frequency of $\texttt{ab}$ and the second 1 denotes the frequency of $\texttt{ba}$, assuming a lexicographic order of q-grams. Since T_1 and T_2 have the same vector representation, they are treated as equal by attributed graph clustering algorithms, although they are not. Similarly, consider a dataset $\{T_1, T_2, T_3\}$ with $T_3 = \texttt{ccc}$. The vector representation of both T_1 and T_2 when using td-idf scores instead of frequencies is $(\log_2(3/2), \log_2(3/2))$, so again T_1 and T_2 are treated as equal in similarity computation. Thus, the similarity information of sequences is not captured well after they are represented based on q-grams.

Sequence-labeled Graphs in Bioinformatics. In bioinformatics, graphs with sequence-labeled nodes have been used extensively in the following context: the nodes represent (short) DNA fragments read by sequencing technologies; and a weighted (directed) edge (u, v) represents the length of the suffix-prefix overlap between sequence u and sequence v. The goal is then to assemble these fragments into a candidate genome represented by some trail in the graph [9]. Let us stress that this task is not related to clustering.

3. Background

An *alphabet* Σ is a finite non-empty set of elements called *letters*; we denote its size by $|\Sigma|$. A *string* $T = T[0]T[1]\dots T[n-1]$ is a sequence of letters of *length* $|T| = n$ over Σ. For two positions i and j on T, we denote by $T[i\,..\,j] = T[i]\dots T[j]$ the *substring* of T that starts at position i and ends at position j of T. By ε we denote the *empty string* of length 0. We refer to a length-q substring of a string T as a q-gram (e.g., in Fig. 1, $\texttt{aab}$ is a 3-gram of string $\texttt{aaab}$).

Let $G = (V, E)$ be a simple graph,[1] where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. The set of neighbors of a node $u \in V$ is $n(u) = \{v \in V \mid (u, v) \in E\}$. The size $|n(u)|$ of $n(u)$ is the *degree* of u. A path p between two nodes u and v in G is a sequence of edges $e_1, \dots, e_{|p|}$ that joins a sequence of distinct nodes $(u, u_1, \dots, u_{|p|-1}, v)$ such that $e_1 = (u, u_1), e_2 = (u_1, u_2), \dots, e_{|p|} = (u_{|p|-1}, v)$. Since the nodes are distinct, p has no cycles. A tour is a path that may have cycles.

A *sequence graph* $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ is a tuple, where V is a set of nodes, $E \subseteq V \times V$ is a set of edges, $\mathcal{S}$ is a set of strings drawn from an alphabet $\Sigma_{\mathcal{S}}$, and $\mathcal{F} : V \to \mathcal{S} \cup \{\varepsilon\}$ is a function that outputs a possibly empty string. In particular, each node $u \in V$ is associated with a string $\mathcal{F}(u) \in \mathcal{S} \cup \{\varepsilon\}$. For example, in Fig. 1, $\mathcal{S} = \{\texttt{aaa}, \texttt{aaab}, \texttt{aabb}, \texttt{bbb}, \texttt{bbbc}\}$, $\Sigma_{\mathcal{S}} = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, and $\mathcal{F}(u_1) = \texttt{aaab}$.

[1] A simple graph is an unweighted, undirected graph with no loops or multiple edges.

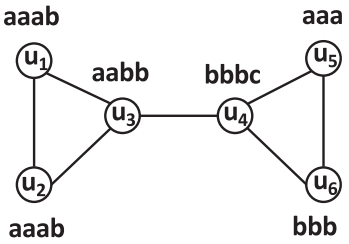


Fig. 1. A sequence graph.

A *k-clustering* of $\mathcal{G}$ for $k \in [1, |V|]$ is a partition of V into k subsets, called *clusters*. We may omit k from k-clustering when it is clear from the context.

The *edit distance* $d_E(U, V)$ between two strings U and V is defined as the minimum number of elementary edit operations (letter insertion, deletion, or substitution) to transform U to V. For U and V of equal length, the *Hamming distance* $d_H(U, V)$ is defined as the minimum number of substitutions to transform U to V. For example, $d_E(\texttt{aaab}, \texttt{aabb}) = d_H(\texttt{aaab}, \texttt{aabb}) = 1$. If U and V are not of the same length, we set $d_H(U, V) = \infty$, for completeness.

The *shortest path distance* $d_{SP}(u, v)$ for two nodes u and v of a graph is defined as the length of the shortest path between u and v. For completeness, we set $d_{SP}(u, v) = \infty$, if there is no path between u and v. For example, in Fig. 1, $d_{SP}(u_1, u_3) = 1$.

Given a constant $c \in (0, 1)$, referred to as *decay factor*, the *SimRank score* $SR(u, v)$ between u and v is defined as follows [20]:

$$SR(u,v) = \begin{cases} 1, & \text{if } u = v \\ 0, & \text{if } n(u) = \emptyset \text{ or } n(v) = \emptyset \\ \frac{c}{|n(u)||n(v)|} \sum_{u' \in n(u)} \sum_{v' \in n(v)} SR(u', v'), & \text{otherwise.} \end{cases} \tag{1}$$

The intuition behind SimRank is that two nodes are similar if they are reachable by similar nodes. SimRank aggregates similarities based on paths.

A *metric space* (X, d) is an ordered pair, where X is a set and d is a metric (also referred to as distance function) on X. For example, the set of strings S of a sequence graph $\mathcal{G}$ together with the edit distance d_E, which is a metric [43], define the metric space (S, d_E).

4. Distance measures for sequence graphs

In the following, we discuss our distance measures for sequence graphs.

4.1. The d_{ESP} measure

Given a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$, metric spaces (S, d_E) and (V, d_{SP}), nodes $u, v \in V$ and strings $\mathcal{F}(u), \mathcal{F}(v)$, the d_{ESP} (E is for Edit distance and SP for Shortest Path distance) measure is defined as:

$$d_{ESP}((\mathcal{F}(u), \mathcal{F}(v)), (u, v)) = \sqrt{(d_E(\mathcal{F}(u), \mathcal{F}(v)))^2 + (d_{SP}(u, v))^2}.$$

The d_{ESP} measure considers the distance between two nodes based on the edit distance between the strings of the nodes and the shortest path distance between the nodes. For example, in Fig. 1, $d_{ESP}(u_1, u_3) = \sqrt{2}$ because $d_E(\texttt{aaab}, \texttt{aabb}) = 1$ and $d_{SP}(u_1, u_3) = 1$. That is, d_{ESP} combines the two metric spaces (S, d_E) and (V, d_{SP}) into a metric space which measures similarity among string-labeled nodes, as if they were points in a 2D space. Note that d_{ESP} is a metric. This is because both edit distance and shortest path distance are metrics [43,44] and d_{ESP} is a 2-product metric [19] on the Cartesian product of the set of strings S and the set of nodes V in a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$.

4.2. The d_{ESR} measure

Our d_{ESR} (E is for Edit distance and SR is for SimRank) measure captures the intuition of SimRank, according to which u and v are similar when they are reachable by similar nodes. However, d_{ESR} differs from SimRank in that it also considers the similarity of the strings of u and v and the strings of their reachable nodes. For example, among two node pairs with the same SimRank score (e.g., u_1, u_5 and u_1, u_6 in Fig. 1), d_{ESR} treats the node pair having more similar strings with respect to edit distance as more similar (e.g., in Fig. 1, $d_{ESR}(u_1, u_5) < d_{ESR}(u_1, u_6)$).

d_{ESR} is based on ESR, a similarity score over a pair of nodes u and v, defined in Eq. (2). Specifically, we then define $d_{ESR}(u, v) = 1 - ESR(u, v)$.

$$ESR(u,v) = \begin{cases} 1, & \text{if } u = v \\ 0, & \text{if } n(u) = \emptyset \text{ or } n(v) = \emptyset \\ \frac{\sigma(u,v)}{|n(u)||n(v)|} \sum_{u' \in n(u)} \sum_{v' \in n(v)} ESR(u', v'), & \text{otherwise,} \end{cases} \tag{2}$$

where $\sigma(u,v) = (1 - \frac{d_E(\mathcal{F}(u),\mathcal{F}(v))}{\max(|\mathcal{F}(u)|,|\mathcal{F}(v)|)}) \cdot (1-\gamma)$ is a similarity score between strings, which is computed by subtracting the normalized edit distance from 1 and multiplying by $(1-\gamma)$ for a small real number $\gamma > 0$. This ensures that $\sigma(u,v) < 1$ for any pair of nodes (u,v), which is required for the iterative computation of ESR (see Theorem 1). Importantly, it also ensures that the multiplication does not substantially change similarity (i.e., the values of $\sigma(u,v)$ and of $(1 - \frac{d_E(\mathcal{F}(u),\mathcal{F}(v))}{\max(|\mathcal{F}(u)|,|\mathcal{F}(v)|)})$ are nearly equal).

Note that ESR cannot be derived from SimRank by setting $c = \sigma(u,v)$ in Eq. (1), since in SimRank $c > 0$ [20] while $\sigma(u,v)$ may be 0. Furthermore, ESR is not a simple weighted version of SimRank with $c = 1$ and weight $\sigma(u,v)$, since $c < 1$ in Eq. (1). ESR is also different from SemSim [45], which was developed for attributed graphs and assumes a large value of normalized semantic similarity between any pair of values.[2] Also, it is easy to see that ESR differs from d_{ESP} in that it considers all reachable nodes from u and v as well as their strings, while d_{ESP} considers only the strings of u and v.

In the following, we show that a fixed-point iteration method, similar to that developed for SimRank [20], can compute ESR. Specifically, in Theorem 1, we define a function $R_\ell(u,v)$, for a node pair u,v and an integer $\ell \geq 0$, which quantifies the similarity of u and v. Next we prove that $R_\ell(u,v)$ can be computed iteratively and that the values of $R_\ell(u,v)$ converge to $ESR(u,v)$. Clearly, our measure can also benefit from efficiency optimizations of SimRank (e.g., [46]) and be extended to address the "zero-similarity" problem [47].

Theorem 1. *Let $u,v \in V$ be a pair of nodes in G and $R_\ell(u,v)$ be a recursive function, defined as follows for some integer $\ell \geq 0$:*

$$R_{\ell+1}(u,v) = \begin{cases} 1, & u = v \\ 0, & n(u) = \emptyset \text{ or } n(v) = \emptyset \\ \frac{\sigma(u,v)}{|n(u)||n(v)|} \sum_{u' \in n(u)} \sum_{v' \in n(v)} R_\ell(u',v'), & otherwise \end{cases} \tag{3}$$

with $R_0(u,v) = \begin{cases} 1, & for\ u = v \\ 0, & otherwise \end{cases}$. *Then, for each $u,v \in V$, there exists a unique solution to Eq. (3) such that* $\lim_{\ell\to\infty} R_\ell(u,v) = ESR(u,v)$.

Proof. We first show the following three properties for R_ℓ:

I Symmetry: $R_\ell(u,v) = R_\ell(v,u)$.
II Maximum self similarity: $R_\ell(u,u) = 1$.
III Monotonicity: $0 \leq R_\ell(u,v) \leq R_{\ell+1}(u,v) \leq 1$.

Then, we will use these properties for proving that the solution of R_ℓ always *exists* and is *unique* to complete the proof.

Symmetry and Maximum self similarity: Properties I and II hold due to the definition of R_ℓ.

Monotonicity: For $u = v$, $R_0(u,v) = R_1(u,v) = \cdots = 1$, so clearly Property III holds. For $u \neq v$ with $n(u) = \emptyset$ or $n(v) = \emptyset$, Property III holds due to the definition of R_ℓ. In any other case, we show that Property III also holds by induction. (Induction base) For $\ell = 0$, the definition of R_ℓ implies:

$$0 \leq R_0(u,v) \leq R_1(u,v) \leq 1. \tag{4}$$

(Induction hypothesis) For an integer $\ell > 0$, $0 \leq R_{\ell-1}(u,v) \leq R_\ell(u,v) \leq 1$ holds.

From the induction hypothesis, it holds that:

$R_{\ell+1}(u,v) - R_\ell(u,v) = \frac{\sigma(u,v)}{|n(u)||n(v)|} \cdot \sum_{u' \in n(u)} \sum_{v' \in n(v)} \left(R_\ell(u',v') - R_{\ell-1}(u',v')\right)$.

$R_\ell(u',v') \geq R_{\ell-1}(u',v')$ holds for any integer $\ell > 0$ and any $u',v' \in V$. Specifically, for $u' = v'$, as well as for u',v' such that $n(u') = \emptyset$ or $n(v') = \emptyset$, $R_\ell(u',v') \geq R_{\ell-1}(u',v')$ holds from the definition of R_ℓ. In any other case, $R_\ell(u',v') \geq R_{\ell-1}(u',v')$ holds from the inductive hypothesis. Also, $\frac{\sigma(u,v)}{|n(u)||n(v)|} \geq 0$ holds by the definition of normalized edit similarity. Thus, $R_{\ell+1}(u,v) - R_\ell(u,v) \geq 0 \Rightarrow R_{\ell+1}(u,v) \geq R_\ell(u,v)$ holds for $\ell > 0$. Therefore, it suffices to show that $0 \leq R_\ell(u,v)$ and $R_{\ell+1}(u,v) \leq 1$ hold, for $\ell > 0$. The first inequality holds, due to the induction hypothesis. For the second inequality, we have:

$$R_{\ell+1}(u,v) = \frac{\sigma(u,v)}{|n(u)||n(v)|} \sum_{u' \in n(u)} \sum_{v' \in n(v)} R_\ell(u',v') \tag{5}$$

$$\leq \frac{\sigma(a,b)}{|n(a)||n(b)|} \cdot |n(a)||n(b)| \cdot 1 \tag{6}$$

$$\leq 1. \tag{7}$$

Eq. (5) is due to Eq. (3). Eq. (6) is due to the induction hypothesis. Eq. (7) is due to the definition of $\sigma(\cdot,\cdot)$. Therefore, $0 \leq R_\ell(u,v) \leq R_{\ell+1}(u,v) \leq 1$ holds for $\ell > 0$, and thus we have shown by induction that $0 \leq R_\ell(u,v) \leq R_\ell(u,v) \leq 1$ holds for any $\ell \geq 0$.

Existence: For $u = v$, or for $u \neq v$ such that $n(u) = \emptyset$ or $n(v) = \emptyset$, it is clear that $\lim_{\ell\to\infty} R_{\ell+1}(u,v) = ESR(u,v)$. In any other case, by the Monotone Convergence Theorem [48, Section 2.5.1, Theorem 1], we know that the series $\{R_\ell(u,v)\}$ for any u,v converges, since

[2] Specifically, SemSim uses a decay factor (alike c in SimRank) that must be smaller than the semantic similarity between the attribute vectors of any two nodes. The latter is zero, or near-zero, in sequence graphs, as they contain empty strings, or strings that are largely different, which makes SemSim inapplicable to our setting.

it is non-decreasing and bounded due to Property III. Furthermore, by Corollary 1 in [48, Section 2.5.1], we know that $\{R_\ell(u,v)\}$ tends to its least upper bound. Thus, it holds $\lim_{\ell\to\infty} R_{\ell+1}(u,v) = \lim_{\ell\to\infty} R_\ell(u,v)$. Therefore,

$$\lim_{\ell\to\infty} R_{\ell+1}(u,v) = \frac{\sigma(u,v)}{|n(u)||n(v)|} \cdot \lim_{\ell\to\infty} \sum_{u'\in n(u)} \sum_{v'\in n(v)} R_\ell(u',v') \tag{8}$$

$$= \frac{\sigma(u,v)}{|n(u)||n(v)|} \cdot \sum_{u'\in n(u)} \sum_{v'\in n(v)} \lim_{\ell\to\infty} R_\ell(u',v'). \tag{9}$$

Eq. (8) is due to the Sum Rule of limits [48, Section 5.1.3, Theorem 3]. Let $R(u,v) = \lim_{\ell\to\infty} R_{\ell+1}(u,v)$. Then, from Eq. (9), we have $R(u,v) = \frac{\sigma(u,v)}{|n(u)||n(v)|} \cdot \sum_{u'\in n(u)} \sum_{v'\in n(v)} R(u',v')$. This shows that the limit of $R_\ell(\cdot,\cdot)$ with respect to ℓ satisfies the ESR equation for $u \neq v$ with $n(u) \neq \emptyset$ and $n(v) \neq \emptyset$. Thus, we have shown existence.

Uniqueness: let $M = \max_{(u,v)} |s(u,v) - s'(u,v)|$, where $s(u,v)$ and $s'(u,v)$ are two solutions to the ESR, for some pair $u, v \in V$. It suffices to show $M = 0$, as this means $s(u,v)$ and $s'(u,v)$ are the same (i.e., the solution is unique). Let $M = |s(a,b) - s'(a,b)|$ for some $a, b \in V$. For $a = b$, $M = |1-1| = 0$ due to Property II. For $a \neq b$ such that $n(u) = \emptyset$ or $n(v) = \emptyset$, $M = 0$. For a, b such that $n(a) \neq \emptyset$ and $n(b) \neq \emptyset$, we distinguish two cases for $\sigma(a,b)$. For $\sigma(a,b) = 0$, clearly $M = 0$. For $\sigma(a,b) > 0$, we have:

$$M = |s(a,b) - s'(a,b)| \tag{10}$$

$$= |\frac{\sigma(a,b)}{|n(a)||n(b)|} \cdot \sum_{u'\in n(a)} \sum_{v'\in n(b)} (s(u',v') - s'(u',v'))| \tag{11}$$

$$= |\frac{\sigma(a,b)}{|n(a)||n(b)|}| \cdot |\sum_{u'\in n(a)} \sum_{v'\in n(b)} (s(u',v') - s'(u',v'))| \tag{12}$$

$$\leq \frac{\sigma(a,b)}{|n(a)||n(b)|} \cdot \sum_{u'\in n(a)} \sum_{v'\in n(b)} |s(u',v') - s'(u',v')| \tag{13}$$

$$\leq \frac{\sigma(a,b)}{|n(a)||n(b)|} \cdot |n(a)||n(b)|M \tag{14}$$

$$\leq \sigma(a,b) \cdot M. \tag{15}$$

Eq. (10) is by the definition of M. Eq. (11) is by Eq. (2). Eq. (12) is by the multiplicativity of absolute value. Eq. (13) is by the triangle inequality of absolute value. Eq. (14) is because each $|s(u',v') - s'(u',v')| \leq M$ by the definition of M. Since $\sigma(a,b) > 0$ in the case we examine, $\sigma(a,b) < 1$ by definition, and $M \geq 0$ by definition, Eq. (15) (i.e., $M \leq \sigma(a,b) \cdot M$) holds if and only if $M = 0$. To see this, let $M > 0$. Then, $\frac{M}{M} = 1 \leq \sigma(a,b)$, which does not hold by the definition of $\sigma(\cdot,\cdot)$. □

4.3. Proxy measures for d_{ESP} and d_{ESR}

Since exact edit distance computation requires quadratic time (assuming the Strong Exponential Time Hypothesis (SETH) is true [49]), d_{ESP} and d_{ESR} are expensive to compute. Therefore, we propose a proxy $d_{\widehat{ESP}}$ and $d_{\widehat{ESR}}$ for d_{ESP} and d_{ESR}, respectively. Instead of considering the strings of a pair of nodes, the proxies consider embeddings of these strings into Hamming distance space. An embedding from a metric space M_1 to a metric space M_2 is a map of points from M_1 to M_2 such that distances are preserved up to some factor D known as *the distortion*. We employ the CGK algorithm [50], which provides a probabilistic embedding with *linear* distortion. The CGK algorithm runs in *linear* time, and if the edit distance between two input strings is K, then the Hamming distance between their embeddings is between $K/2$ and $O(K^2)$ with *good* probability [50]. By incorporating CGK instead of edit distance in our algorithms, we substantially improve their efficiency without substantially degrading effectiveness, as it will be shown in Section 7.

5. Sequence graph k-Center and k-Median

We present two clustering problems for sequence graphs inspired by the k-Center and the k-Median problems [21–23].

5.1. Sequence graph k-Center (SGC)

The SGC problem, defined in Problem 1 below, requires finding k nodes, referred to as *centers*, such that the maximum distance of any node to a closest center with respect to d_{ESP} is minimized.

Problem 1 (*Sequence Graph k-Center (SGC)*)**.** Given a sequence graph $\mathcal{G} = (V, E, S, \mathcal{F})$ and an integer $k \in [1, |V|]$, find a subset $C \subseteq V$ of k nodes such that $\max_{u\in V} \min_{c\in C} d_{ESP}(u,c)$ is minimized.

A clustering $\{V_1, \ldots, V_k\}$ of $\mathcal{G}$ is obtained from a solution C to SGC, by assigning into each cluster V_i, $i \in [1,k]$, a center $c_i \in C$ and also every node $u \notin C$ that is closer to this center compared to other centers (i.e., every u such that $d_{ESP}(u,c_i) < d_{ESP}(u,c_j)$, for each $j \neq i$). If a node is in equal distance from multiple clusters, it is assigned into an arbitrarily selected cluster containing one of these centers.

The *decision version* of SGC asks whether there exists a subset $C \subseteq V$ of k nodes such that

$$\max_{u \in V} \min_{c \in C} d_{ESP}(u, c) \leq B,$$

for a given real number B.

We show that SGC is NP-hard by showing that its decision version is NP-complete. We prove this result via a reduction from the decision version of metric k-Center [22], which is known to be NP-complete [22].

Problem 2 (*Metric k-Center (Decision Version) [22]*)**.** Given a metric space (P, d), where P is a set of n points and $d : P \times P \to \mathbb{R}^+$ is a distance function, an integer $k \in [1, n]$, and a real number B, decide whether there exists a subset $C \subseteq P$ of k points such that $\max_{p \in P} \min_{c \in C} d(p, c) \leq B$.

Lemma 1. *The decision version of SGC is NP-complete.*

Proof. The decision version of SGC is clearly in NP. In the following, we show that it can be reduced from the decision version of metric k-Center. Given any instance $\mathcal{I}_{kC}$ of the decision version of metric k-Center, we construct an instance $\mathcal{I}_{SGC}$ of the decision version of SGC in polynomial time, by creating a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ as follows: (I) We add a node u into V, for each point $p \in P$, where P is the set of points in $\mathcal{I}_{kC}$. (II) We set $E = \emptyset$. (III) We set $\mathcal{S} = \{\varepsilon\}$. (IV) We define $\mathcal{F}$ such that $\mathcal{F}(u) = \varepsilon$, for each $u \in V$. We also set k and B in $\mathcal{I}_{SGC}$ to k and B in $\mathcal{I}_{kC}$, respectively, and $d_{ESP}((\mathcal{F}(u), \mathcal{F}(v)), (u, v)) = d(p, p')$, for every pair $(p, p') \in P$ corresponding to pair $(u, v) \in V$. This completes the construction of $\mathcal{I}_{SGC}$.

In the following, we prove that $\mathcal{I}_{SGC}$ has a positive answer if and only if $\mathcal{I}_{kC}$ has a positive answer.
($\Rightarrow$) If $\mathcal{I}_{SGC}$ has a positive answer, there is a subset $C \subseteq V$ of k nodes such that $\max_{u \in V} \min_{c \in C} d_{ESP}(u, c) \leq B$. These nodes correspond to a subset $C \subseteq P$ of k points such that $\max_{p \in P} \min_{c \in C} d(p, c) \leq B$. Thus, $\mathcal{I}_{kC}$ has a positive answer.
($\Leftarrow$) If $\mathcal{I}_{kC}$ has a positive answer, then there is a subset $C \subseteq P$ of k points such that $\max_{p \in P} \min_{c \in C} d(p, c) \leq B$. These points correspond to a subset $C \subseteq V$ of k nodes such that $\max_{u \in V} \min_{c \in C} d_{ESP}(u, c) \leq B$. Thus, $\mathcal{I}_{SGC}$ has a positive answer. □

Due to Lemma 1, we obtain Theorem 2 directly.

Theorem 2. *SGC is NP-hard.*

Proof. The statement follows from Lemma 1. □

We also consider the following variant of SGC which uses d_{ESR} instead of d_{ESP}:

Problem 3 (*Sequence Graph k-Center (SGC) with d_{ESR}*)**.** Given a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ and an integer $k \in [1, |V|]$, find a subset $C \subseteq V$ of k nodes such that $\max_{u \in V} \min_{c \in C} d_{ESR}(u, c)$ is minimized.

The NP-hardness of the variant follows from a similar reduction to that of Lemma 1. The main change is that we reduce from the decision version of the k-center problem in which d is a dissimilarity function (not necessarily a metric) [21] and that we use d_{ESR} instead of d_{ESP}.

5.2. Sequence graph k-Median (SGM)

The Sequence Graph k-Median (SGM) problem requires finding k nodes, referred to as *representatives*, such that the sum of distances between nodes and their closest representative with respect to d_{ESP} is minimized.

Problem 4 (*Sequence Graph k-Median (SGM)*)**.** Given a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ and an integer $k \in [1, |V|]$, find a set $C \subseteq V$ of k nodes such that $\sum_{u \in V} \min_{c \in C} d_{ESP}(u, c)$ is minimized.

Then, a clustering $\{V_1, \ldots, V_k\}$ is obtained from C as in the SGC problem.

We prove that SGM is NP-hard by reducing the decision version of SGM from the decision version of the metric k-Median problem, which is NP-complete [23]. The decision version of SGM asks whether there exists a subset $C \subseteq V$ of k nodes such that $\sum_{u \in V} \min_{c \in C} d_{ESP}(u, c) \leq B$, for a given real number B.

Problem 5 (*Metric k-Median (Decision Version) [23]*)**.** Given a metric space (P, d), where P is a set of n points and $d : P \times P \to \mathbb{R}^+$ is a distance function, an integer $k \in [1, n]$, and a real number T, decide whether there exists a subset $C \subseteq P$ such that $\sum_{p \in P} \min_{c \in C} d(p, c) \leq T$.

Lemma 2. *The decision version of SGM is NP-complete.*

Proof. Given any instance $\mathcal{I}_{kM}$ of the decision version of k-Median, we construct an instance $\mathcal{I}_{SGM}$ of the decision version of SGM in polynomial time, by creating a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ as follows: (I) We add a node u into V, for each point $p \in P$, where P is the set of points in $\mathcal{I}_{kM}$. (II) We set $E = \emptyset$. (III) We set $\mathcal{S} = \{\varepsilon\}$. (IV) We define $\mathcal{F}$ such that $\mathcal{F}(u) = \varepsilon$ for each $u \in V$. We

also set k in $\mathcal{I}_{SGM}$ to k in $\mathcal{I}_{kM}$, and $d_{ESP}(u,v) = d(p,p')$, for every pair $(p,p') \in P$ corresponding to pair $(u,v) \in V$. Last, we set B in $\mathcal{I}_{SGM}$ to T. This completes the construction.

In the following, we prove that $\mathcal{I}_{SGM}$ has a positive answer if and only if $\mathcal{I}_{kM}$ has a positive answer.
($\Rightarrow$) If $\mathcal{I}_{SGM}$ has a positive answer, then there is a subset $C \subseteq V$ of k nodes such that $\sum_{u\in V} \min_{c\in C} d_{ESP}(u,c) \leq B$. These nodes correspond to a subset $C \subseteq P$ of k points such that $\sum_{p\in P} \min_{c\in C} d(p,c) \leq T$. Thus, $\mathcal{I}_{kM}$ has a positive answer.
($\Leftarrow$) If $\mathcal{I}_{kM}$ has a positive answer, then there is a subset $C \subseteq P$ of k points such that $\sum_{p\in P} \min_{c\in C} d(p,c) \leq T$. These points correspond to a subset $C \subseteq V$ of k nodes such that $\sum_{u\in V} \min_{c\in C} d_{ESP}(u,c) \leq B$. Thus, $\mathcal{I}_{SGM}$ has a positive answer. □

Due to Lemma 2, we obtain Theorem 3 directly.

Theorem 3. *SGM is NP-hard.*

Proof. The statement follows from Lemma 2.

We also consider a variant of SGM which uses d_{ESR} instead of d_{ESP}:

Problem 6 (*Sequence Graph k-Median (SGM) with d_{ESR}*)**.** Given a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ and an integer $k \in [1, |V|]$, find a set $C \subseteq V$ of k nodes such that $\sum_{u\in V} \min_{c\in C} d_{ESR}(u,c)$ is minimized.

The decision version of Problem 6 asks whether there exists a subset $C \subseteq V$ of k nodes such that $\sum_{u\in V} \min_{c\in C} d_{ESR}(u,c) \leq B$, for a given real number B. Below, we provide a reduction from the decision version of k-Median [21], which implies that Problem 6 is NP-hard.

Lemma 3. *The decision version of Problem 6 is NP-complete.*

Proof. Given any instance $\mathcal{I}_{kM}$ of the decision version of k-Median, we construct an instance $\mathcal{I}_{P2}$ of the decision version of Problem 6 in polynomial time, by creating a sequence graph $\mathcal{G} = (V, E, \mathcal{S}, \mathcal{F})$ as follows: (I) We add a node u into V, for each point $p \in P$, where P is the set of points in the decision version of k-Median. (II) We set $E = \emptyset$. (III) We set $\mathcal{S} = \{\varepsilon\}$. (IV) We define $\mathcal{F}$ such that $\mathcal{F}(u) = \varepsilon$ for each $u \in V$. We also set k in $\mathcal{I}_{P2}$ to k in $\mathcal{I}_{kM}$, and set $d_{ESR}(u,v) = d(p,p')/(\max_{(p,p')\in P} d(p,p') + 1)$, for every pair $(p,p') \in P$ corresponding to pair $(u,v) \in V$, where $d : P \times P \to \mathbb{R}^+$ is the dissimilarity function in the decision version of k-Median. For brevity, we denote $\max_{(p,p')\in P} d(p,p')+1$ by μ. The division with μ ensures that a value in $[0,1]$ is assigned to d_{ESR} and is needed because d_{ESR} takes values in $[0,1]$, whereas d takes values in $\mathbb{R}^+$. Clearly, μ can be computed in polynomial time. Last, we set B in $\mathcal{I}_{P2}$ to $\frac{T}{\mu}$. This completes the construction.

In the following, we prove that $\mathcal{I}_{P2}$ has a positive answer if and only if $\mathcal{I}_{kM}$ has a positive answer.
($\Rightarrow$) If $\mathcal{I}_{P2}$ has a positive answer, then there is a subset $C \subseteq V$ of k nodes such that $\sum_{u\in V} \min_{c\in C} d_{ESR}(u,c) \leq B$. These nodes correspond to a subset $C \subseteq P$ of k points such that $\sum_{p\in P} \min_{c'\in C} d(p,c') \leq T$, since $d_{ESR}(u,c) = \frac{d(p,c')}{\mu}$, for every pair of nodes $(u,c) \in C$ corresponding to a pair of points $(p,c') \in C$, and $B = \frac{T}{\mu}$. Thus, $\mathcal{I}_{kM}$ has a positive answer.
($\Leftarrow$) If $\mathcal{I}_{kM}$ has a positive answer, then there is a subset $C \subseteq P$ of k points such that $\sum_{p\in P} \min_{c'\in C} d(p,c') \leq T$. These points correspond to a subset $C \subseteq V$ of k nodes such that $\sum_{u\in V} \min_{c\in C} d_{ESR}(u,c) \leq B$, since $d(p,c') = d_{ESR}(u,c) \cdot \mu$, for every pair of points $(p,c') \in C$ corresponding to a pair of nodes $(u,c) \in C$, and $T = B \cdot \mu$. Thus, $\mathcal{I}_{P2}$ has a positive answer. □

6. Algorithms for clustering sequence graphs

We present algorithms for the SGC and SGM problems. These algorithms are presented with d_{ESP} but they can also use d_{ESR}, or the proxies of these measures, instead (see Section 7).

6.1. Approximation algorithm for SGC

We begin by showing a polynomial-time approximation-preserving reduction from SGC to (the optimization version of) metric k-Center [22] defined below. This allows approximating SGC within a 2-factor.

Problem 7 (*Metric k-Center [22]*)**.** Given a metric space (P,d), where P is a set of n points and $d : P \times P \to \mathbb{R}^+$ is a distance function, and an integer $k \in [1,n]$, find a subset $C \subseteq P$ of k points such that $\max_{p\in P} \min_{c\in C} d(p,c)$ is minimized.

Lemma 4. *SGC can be reduced to metric k-Center.*

Proof. Given any instance $\mathcal{I}_{SGC}$ of SGC, we construct an instance $\mathcal{I}_{kC}$ of metric k-Center in polynomial time, as follows: (I) We construct a set P of points by adding into an initially empty set P a point p, for each node $u \in V$ in the graph of $\mathcal{I}_{SGC}$. (II) We set k and B in $\mathcal{I}_{kC}$ to k and B in $\mathcal{I}_{SGC}$, respectively. (III) We set $d(p,p') = d_{ESP}((\mathcal{F}(u),\mathcal{F}(v)),(u,v))$, for every pair of nodes $(u,v) \in V$ corresponding to a pair of points $(p,p') \in P$. This completes the construction of $\mathcal{I}_{kC}$.

In the following, we prove the correspondence between a solution S_{kC} to $\mathcal{I}_{kC}$ and a solution S_{SGC} to $\mathcal{I}_{SGC}$.

($\Rightarrow$) If S_{kC} is a solution to $\mathcal{I}_{kC}$, then there is a subset $C \subseteq P$ of k points such that $\max_{p\in P} \min_{c\in C} d(p,c)$ is minimum. These points correspond to a subset $C \subseteq V$ of k nodes such that $\max_{u\in V} \min_{c\in C} d_{ESP}(u,c)$ is minimum. Thus, S_{SGC} is a solution to $\mathcal{I}_{SGC}$.
($\Leftarrow$) If S_{SGC} is a solution to $\mathcal{I}_{SGC}$, then there is a subset $C \subseteq V$ of k nodes such that $\max_{u\in V} \min_{c\in C} d_{ESP}(u,c)$ is minimum. These nodes correspond a subset $C \subseteq P$ of k points such that $\max_{p\in P} \min_{c\in C} d(p,c)$ is minimum. Thus, S_{kC} is a solution to $\mathcal{I}_{kC}$. □

Theorem 4. *SGC can be approximated within a factor of* 2*, for any* $\epsilon > 0$.

Proof. The statement follows from Lemma 4.

We also show that SGC cannot be approximated within a $2-\epsilon$ factor, for any $\epsilon > 0$, by reducing metric k-Center to SGC.

Theorem 5. *SGC cannot be approximated within a* $2-\epsilon$ *factor, for any* $\epsilon > 0$.

Proof. Metric k-Center cannot be approximated within a $2-\epsilon$ factor for any $\epsilon > 0$ [22]. Thus, it suffices to reduce k-Center to SGC.

Given any instance $\mathcal{I}_{kC}$ of metric k-Center, we construct an instance $\mathcal{I}_{SGC}$ of SGC in polynomial time, as follows. First, we create a sequence graph $\mathcal{G} = (V, E, S, \mathcal{F})$ as follows: (I) We add a node u into V, for each point $p\in P$, where P is the set of points in $\mathcal{I}_{kC}$. (II) We set $E = \emptyset$. (III) We set $S = \{\varepsilon\}$. (IV) We define $\mathcal{F}$ such that $\mathcal{F}(u) = \varepsilon$, for each $u \in V$. We also set k in $\mathcal{I}_{SGC}$ to k in $\mathcal{I}_{kC}$, and $d_{ESP}((\mathcal{F}(u), \mathcal{F}(v))) = d(p,p')$, for every pair $(p,p') \in P$ corresponding to pair $(u,v) \in V$. This completes the construction of $\mathcal{I}_{SGC}$.

The correspondence between a solution S_{SGC} to $\mathcal{I}_{SGC}$ and a solution S_{kC} to $\mathcal{I}_{kC}$ holds due to Lemma 4. □

Therefore, we develop SGC-APPROX, a 2-approximation algorithm for SGC which is based on the algorithm of Gonzalez [51] for metric k-Center. Note that, by Theorem 5, it is not possible to design a polynomial-time approximation algorithm for SGC with better approximation ratio than that of SGC-APPROX.

Algorithm 1 SGC-APPROX($\mathcal{G}$, k)

Input: Sequence graph $\mathcal{G}(V, E, S, \mathcal{F})$ and the number of clusters k
Output: a set of clusters $\mathcal{C}$
1: Select an arbitrary node c from V
2: Add cluster $\{c\}$ into an empty set of clusters $\mathcal{C}$
3: Add node c into an empty set C
4: **while** $|C| < k$ **do**
5: Select node u such that $\min_{u\in V\setminus C, c\in C} d_{ESP}(u,c)$ is maximized
6: Add cluster $\{u\}$ into set of clusters $\mathcal{C}$
7: Add node u into set C
8: **end while**
9: **for** each node $u \in V \setminus C$ **do**
10: Add u into the cluster in $\mathcal{C}$ whose center c has minimum $d_{ESP}(u,c)$
11: **end for**
12: **return** $\mathcal{C}$

Our algorithm works as follows (see Algorithm 1 for the pseudocode). It adds an arbitrary node c into an initially empty set of clusters $\mathcal{C}$ and into an auxiliary set C that will contain the selected centers (Lines 1 to 3). Then, it performs $k-1$ iterations (Lines 4 to 8). In each iteration, it finds a node that is as far as possible from its closest node in C with respect to d_{ESP}. This node is selected as a center and is added into $\mathcal{C}$ and into C. After that, SGC-APPROX constructs and returns a clustering comprised of a center and its closest nodes with respect to d_{ESP}, breaking ties arbitrarily (Lines 9 to 12).

The approximation guarantee of SGC-APPROX is given below:

Theorem 6. *SGC-APPROX finds a solution* $\mathcal{C}$ *to SGC with*

$$\max_{u\in V} \min_{c\in C} d_{ESP}(u,c) \leq 2 \cdot \max_{u\in V} \min_{c\in C^*} d_{ESP}(u,c),$$

where C^* *is an optimal solution to SGC, in* $O(((\max_{s\in S} |s|)^2 + |E|)k|V|)$ *time.*

Proof. The approximation guarantee of SGC-APPROX follows from that the algorithm of Gonzalez [51] has an approximation factor of 2 for the metric k-Center problem and from Theorem 4.

The time complexity of SGC-APPROX is $O(((\max_{s\in S} |s|)^2 + |E|)k|V|)$. This is because, in each of the $O(k)$ iterations, the algorithm computes d_{ESP} between u and v, where u is one of the $O(k)$ nodes in C and v is one of the $O(|V|)$ nodes that are not contained in C, and each such computation takes $O((\max_{s\in S} |s|)^2 + |E|)$ time: $O((\max_{s\in S} |s|)^2)$ for computing d_E [52]; and $O(|E|)$ for computing d_{SP} using the algorithm of Thorup [53]. □

6.2. Heuristic for SGM

We propose SGM-HEUR (see Algorithm 2 for the pseudocode), a heuristic based on an efficient and effective version of the k-medoids algorithm [54]. Our heuristic selects k nodes with a smallest score $\sum_{v\in V} \frac{d_{ESP}(u,v)}{\sum_{v'\in V} d_{ESP}(v,v')}$, treats each such node as a cluster

representative (medoid), and adds each other node into the cluster of its closest representative (Lines 1 to 8). After that, it updates the representatives and the clusters, as k-medoids does, until the clusters do not change or M iterations are performed (Lines 9 to 21). Last, it returns the set of clusters (Line 22). SGM-HEUR requires $O([|V|^2 \cdot ((\max_{s\in S}|s|)^2 + |E|)] + [|V|kM])$ time. The term in the first (respectively, second) pair of square brackets is the time for computing d_{ESP} for all node pairs (respectively, the time for deriving the clustering).

Algorithm 2 SGM-HEUR($\mathcal{G}$, M, k)

Input: Sequence graph $\mathcal{G}(V, E, S, \mathcal{F})$, maximum number of iterations M, and the number of clusters k
Output: a set C of k clusters

```
    /* Select initial medoids */
 1: Create empty set C
 2: C ← set of k nodes in V with a smallest ∑_{v∈V'} d_ESP(v,v') / ∑_{v'∈V} d_ESP(v,v')
    /* Construct initial clusters */
 3: while C ≠ ∅ do
 4:     Select a node u ∈ C arbitrarily
 5:     Create a cluster c = {u} with medoid μ(c) = u
 6:     Add c into C
 7:     Remove c from C
 8: end while
 9: Add each u ∈ V into a cluster c ∈ C such that d_ESP(u, μ(c)) is minimum
10: S_0 ← ∑_{u∈c : c∈C} d_ESP(u, μ(c))
11: S_m ← 0, for each m ∈ [1, M]
12: for each m ∈ [1, M] do
    /* Update medoids */
13:     for each c ∈ C do
14:         μ(c) ← node u ∈ c with a minimum ∑_{u'∈c} d_ESP(u, u')
15:     end for
    /* Assign nodes into clusters */
16:     Add each u ∈ V into a cluster c ∈ C such that d_ESP(u, μ(c)) is minimum
17:     S_m ← ∑_{u∈c : c∈C} d_ESP(u, μ(c))
18:     if S_m = S_{m−1} then
19:         break
20:     end if
21: end for
22: return C
```

7. Experimental evaluation

In this section, we evaluate our algorithms with respect to effectiveness and efficiency. We also demonstrate that ESR and $\widehat{ESR}$ and hence d_{ESR} and $d_{\widehat{ESR}}$ converge fast.

Evaluated Methods. We tested SGC-APPROX and SGM-HEUR with d_{ESP}, $d_{\widehat{ESP}}$, d_{ESR}, or $d_{\widehat{ESR}}$. We report results for SGC-APPROX with d_{ESP} (referred to as CA) and with $d_{\widehat{ESP}}$ (referred to as CA_E), as well as for SGM-HEUR with d_{ESR} (referred to as MH) and with $d_{\widehat{ESR}}$ (referred to as MH_E). The use of d_{ESR} and $d_{\widehat{ESR}}$ in SGC-APPROX did not substantially help quality but reduced efficiency, while the use of d_{ESP} and $d_{\widehat{ESP}}$ in SGM-HEUR did not substantially help efficiency but reduced quality; thus we omit these results. In all our algorithms, we used a normalized version of d_{ESP}, where d_E and d_{SP} is divided by its maximum value, and a similarly normalized version of $d_{\widehat{ESP}}$. We compared our algorithms against four state-of-the-art attributed graph clustering methods which employ different techniques (see Section 2): (I) Text-Associated DeepWalk (TADW) [12], (II) Text Enhanced Network Embedding (TENE) [16], (III) Binarized Attributed Network Embedding (BANE) [17], and (IV) Adaptive Graph Convolution (AGC) [18].

To use these methods, we first constructed the set $\mathcal{Q} = \cup_{s\in S} Q_s$, where Q_s is the set of q-grams of a string $s \in S$, and then embedded the string $\mathcal{F}(u)$ of each node $u \in V$ into an attribute vector $\mathcal{A}_u$ such that $\mathcal{A}_u[i]$ is equal to: (I) 1, if $\mathcal{F}(u)$ contains the q-gram with lexicographic rank[3] i in $\mathcal{Q}$, and 0 otherwise, (II) the frequency in $\mathcal{F}(u)$ of the q-gram with lexicographic rank i in $\mathcal{Q}$, or (III) the tf-idf score in S of the q-gram with lexicographic rank i in $\mathcal{Q}$. Note that such embeddings have already been used in the literature, for instance, in [10,16,33]. We report results for the best embedding method for each competitor and q. The real-valued embedding constructed by TADW or TENE was fed into k-means, following [55], while the binary embedding of BANE was fed into k-medoids [54] (with Hamming distance). We also implemented variants of CA and MH that represent a string using a vector of q-grams, as TADW and TENE do, reporting results for the best embedding method and q. We refer to the variant of CA (respectively, MH) as CA^{vec} (respectively, MH^{vec}). Methods for clustering strings (k-medoids [54], k-means [1]) or graphs (spectral clustering [1]) performed worse than [12,16,17]; thus we omit their results. We summarize all tested methods in Table 1, for ease of reference.

[3] The *lexicographic rank* of a string in a set of strings is the rank of the string in the lexicographically sorted list of all the strings in the set.

Table 1
Summary of the methods used in experiments.

Acronym	Description
CA	SGC-APPROX with d_{ESP}
CA_E	SGC-APPROX with $d_{\widehat{ESP}}$
MH	SGM-HEUR with d_{ESR}
MH_E	SGM-HEUR with $d_{\widehat{ESR}}$
CA^{vec}	Variant of CA with vector of q-grams
MH^{vec}	Variant of MH with vector of q-grams
TADW	Text-Associated DeepWalk [12]
TENE	Text Enhanced Network Embedding [16]
BANE	Binarized Attributed Network Embedding [17]
AGC	Adaptive Graph Convolution [18]

Table 2
Datasets characteristics.

Dataset	# nodes	# edges	Avg., max. degree	Alphabet size $\lvert\Sigma_S\rvert$	Avg., max. string length
CIAO	2,375	43,690	36.792, 453	6	15.185, 868
EPIN	22,165	286,822	25.881, 2,032	27	41.609, 5,357

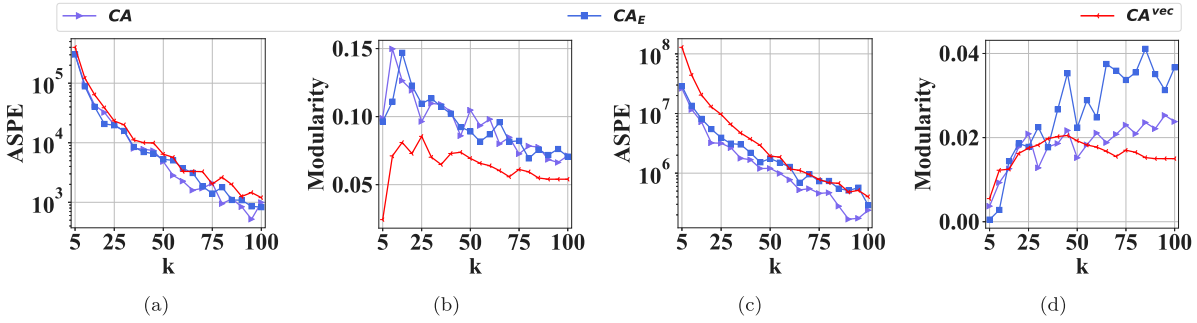


Fig. 2. Comparison of our algorithms against variants that represent strings as attribute vectors: (a) ASPE vs. k for CIAO. (b) Modularity vs. k for CIAO. (c) ASPE vs. k for EPIN. (d) Modularity vs. k for EPIN.

Datasets and Setup. We used the Ciao (CIAO) and Epinions (EPIN) datasets from https://www.cse.msu.edu/~tangjili/datasetcode/truststudy.htm.

In these datasets, each user (node) is associated with a (potentially empty) sequence of reviewed products (string), and an edge connects users who trust each other. Table 2 summarizes the characteristics of the datasets we used.

Clustering quality was quantified using: (I) the Average Sum of Pairwise Edit distances (ASPE), defined as $\frac{1}{k}\sum_{c\in\mathcal{C}}\sum_{u,v\in c} d_E(\mathcal{F}(u),\mathcal{F}(v))$; and (II) Modularity [2], a well-established measure of network (graph) clustering quality, expressed as the fraction of the edges that fall within the given clusters minus the expected fraction if edges were distributed at random. Small ASPE and large Modularity values are preferred. The speed of convergence of ESR (see Theorem 1) was measured using Average Relative Difference (ARD) [45], defined as $\frac{1}{|V|^2}\sum_{u,v\in V}\frac{R_{\ell+1}(u,v)-R_\ell(u,v)}{R_\ell(u,v)}$, where ℓ and $\ell+1$ are two consecutive iterations of the fixed-point iteration algorithm. ARD was also used with $\widehat{ESR}$ (defined as $\widehat{ESR}(u,v)=1-d_{\widehat{ESR}}(u,v)$) and SimRank.

By default, we used $k=15$, $q=2$, $\gamma=10^{-9}$, and $\ell=5$. We also used the default value $M=300$ [54] and default parameter values for competitors. The proxy measures were implemented following [56]. All results are averaged over 10 runs. All algorithms were implemented in Python and executed on an Intel i9 at 3.70 GHz with 64 GB RAM. Our source code is available at https://rebrand.ly/SGcode.

String vs. Vector Representation. We show that representing strings as attribute vectors has a negative impact on clustering quality via comparing our algorithms to the variants CA^{vec} and MH^{vec}. As can be seen in Figs. 2 and 3, CA^{vec} resulted in higher (worse) ASPE than both CA and CA_E. For example, the average ASPE for CA^{vec} was higher than that of CA (respectively, CA_E) by 104% (respectively, 56%) on average (over the two datasets). Also, CA^{vec} resulted in lower (worse) Modularity than both CA and CA_E. For example, the average Modularity for CA^{vec} was lower than that of CA (respectively, CA_E) by 18% (respectively, 33%) (over the two datasets). These results show the benefit of measuring similarity by using strings directly as our algorithms do. Since CA^{vec} and MH^{vec} performed worse than CA and MH, we do not report results for them in the remaining of this section.

Clustering Quality. We show that our algorithms created clusters containing nodes with similar strings, as ASPE is lower than that of most competitors (see Fig. 4(a)), which are also structurally similar, as Modularity is higher than that of most competitors

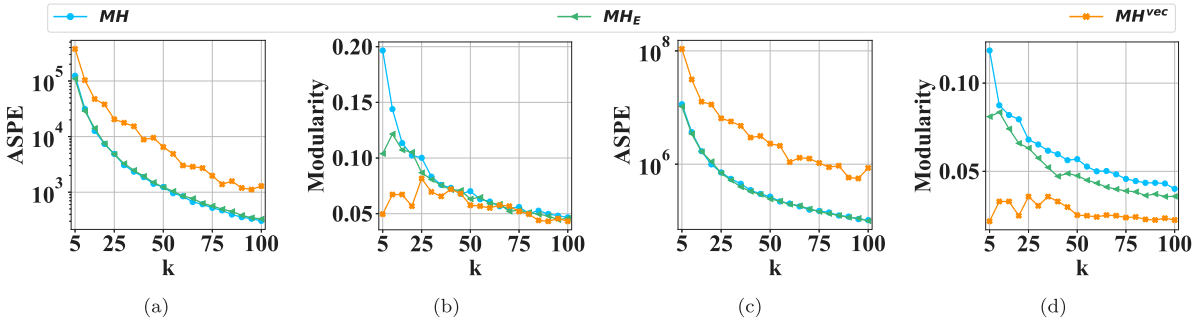


Fig. 3. Comparison of our algorithms against variants that represent strings as attribute vectors: (a) ASPE vs. k for CIAO. (b) Modularity vs. k for CIAO. (c) ASPE vs. k for EPIN. (d) Modularity vs. k for EPIN.

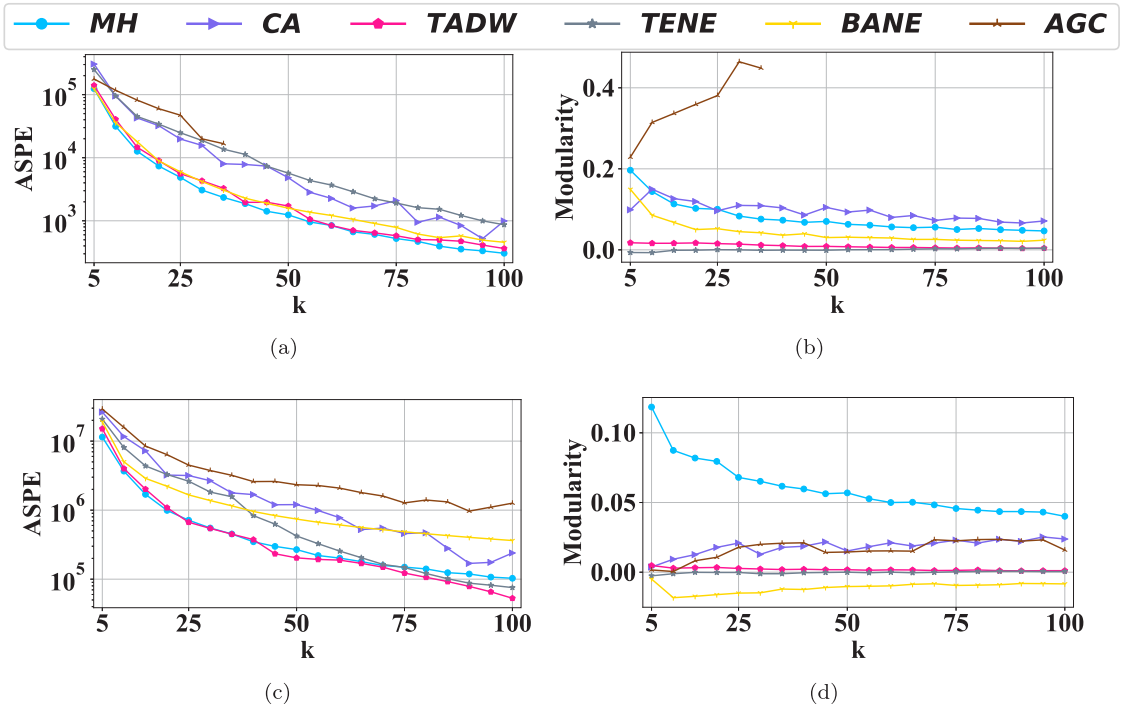


Fig. 4. Comparison of our algorithms against state-of-the-art algorithms for attributed graph clustering: (a) ASPE and (b) Modularity vs. k for CIAO. (c) ASPE and (d) Modularity vs. k for EPIN.

(see Fig. 4(b)). In addition, the use of proxy measures by our algorithms did not substantially affect clustering quality, as CA_E and MH_E performed very similarly to CA and MH, respectively (see Fig. A.8 in Appendix A). On the other hand, the competitors achieved a worse clustering than our algorithms, when considering both ASPE and Modularity together, and some were worse in both of these measures. For example, TENE performed poorly in terms of both ASPE and Modularity, while BANE and TADW performed the worst in terms of Modularity and worse than MH and MH_E in terms of ASPE. AGC performed the worst in terms of ASPE but the best in terms of Modularity for $k < 36$. For $k \geq 36$, AGC did not terminate because the eigenvalue decomposition it employs to find the convolution failed. Similar results were obtained for the EPIN dataset (see Figs. 4(c) and 4(d)).

The reason that AGC performed poorly with respect to ASPE is that it favors Modularity by design, since it assumes that nodes that are close in the graph will likely be clustered together, as mentioned in Section 2. This assumption is not necessarily true. In fact, in our setting, AGC created clusters comprised of nodes that are close in the graph but have quite dissimilar strings and this led to poor clusters in terms of ASPE. The reason that TADW performed poorly with respect to Modularity is that it supervises random walks based on attribute vectors, which resulted in clusters with nodes that are far apart in the graph. The reason that BANE (respectively, TENE) performed poorly in terms of Modularity is that it does not use higher than first (respectively, second) order proximities to capture the distance of nodes in the graph. However, such proximities are important to consider [42] because good clusters may be constructed based on higher than second order proximities.

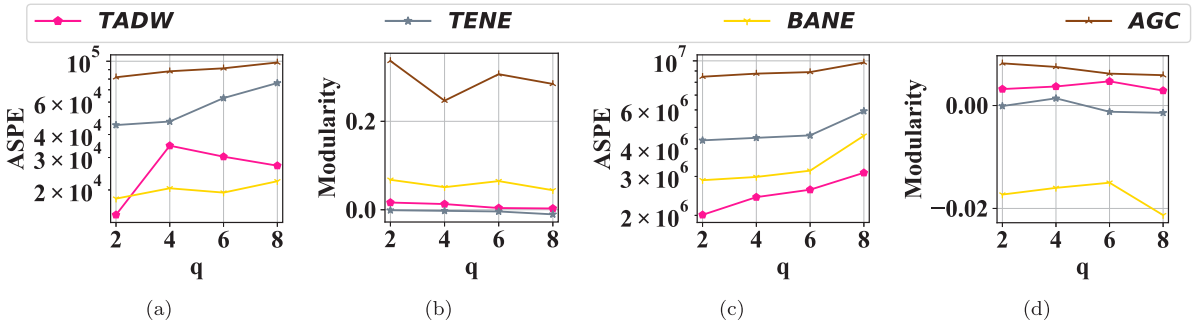


Fig. 5. Impact of q on clustering quality for competitors: (a) ASPE and (b) Modularity vs. q for CIAO. (c) ASPE and (d) Modularity vs. q for EPIN.

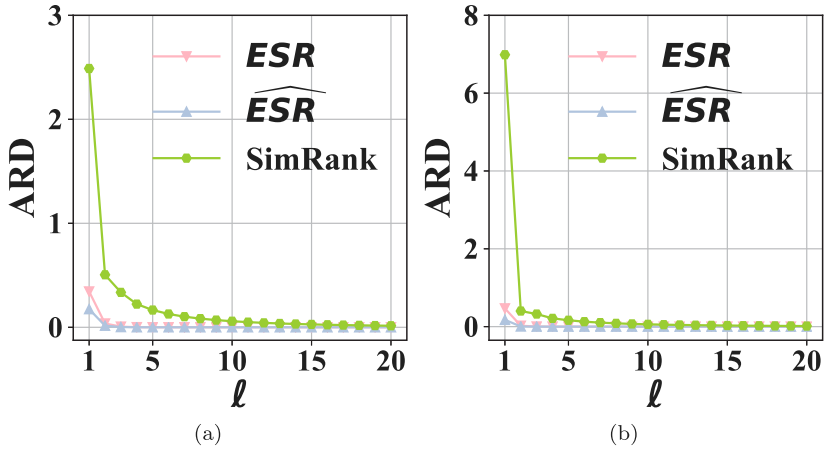


Fig. 6. Convergence speed for our measures and SimRank: ARD vs. ℓ (number of iterations of fixed-point iteration algorithm) for: (a) CIAO and (b) EPIN.

The good performance of our methods is due to three factors: (I) d_{ESR} and d_{ESP} can capture graph distance and string distance in a unified manner. (II) Unlike the competitors, our methods use the strings directly in similarity measurements instead of representing strings as attribute vectors, which may lose similarity information. (III) Unlike TENE and BANE, which only use first or first and second order proximities, our methods employ measures that capture distance between two nodes based also on longer paths.

Note that the values of ASPE and Modularity depend on the similarity between strings and the similarity between nodes, respectively. Thus, it may not be possible to create a clustering with both low ASPE and high Modularity, when close nodes have different strings and vice versa.

Besides, we also examined the impact of q (length of q-gram used in the vector representation of a string by the competitors). Figs. 5 (c)–(d) show that cluster quality in EPIN became worse as q increased. This is because the number of distinct q-grams (i.e., $|\mathcal{Q}|$) increases and thus ASPE increased as well. Thus, the default value $q = 2$ we used is a fair choice. The results for CIAO were similar (see Figs. 5 (a)–(b)).

Convergence. We show that ESR and $\widehat{ESR}$ converge faster than SimRank, which helps efficiency (see Fig. 6) This is attributed to the impact of string similarity (e.g., $\sigma(u, v) = 0$ implies $R_{\ell+1}(u, v) = 0$ in Eq. (3)). In fact, the ARD scores for ESR and $\widehat{ESR}$ were smaller than 10^{-3} after 5 iterations, while those for SimRank were an order of magnitude larger even after 20 iterations.

Runtime. We examined the runtime of all methods for varying number of nodes (see Fig. 7) CA_E and MH_E were much more efficient than all competitors. For instance, CA_E was up to 8 and 3 times faster than TENE, the fastest competitor, in the experiment of Fig. 7(a) and 7(b), respectively. As expected, CA_E and MH_E were faster than CA and MH, since the last two algorithms need to compute edit distance instead of the more efficient to compute proxy measures. For example, MH_E was two orders of magnitude faster than MH in the case of clustering CIAO and even faster in the case of clustering EPIN. In addition, CA_E was approximately two times faster than CA. The impact of using the proxy measure in the algorithms for SGC was less significant compared to the algorithms for SGM, because in the former there are fewer edit distance computations, as in SGC there is no need to compute all pairwise distances between strings. CA was faster than MH, as expected by the complexity analysis (see Section 6). For example, CA was more than 50 times faster than MH in the case of the CIAO dataset and more than two order of magnitudes faster in the case of the EPIN dataset.

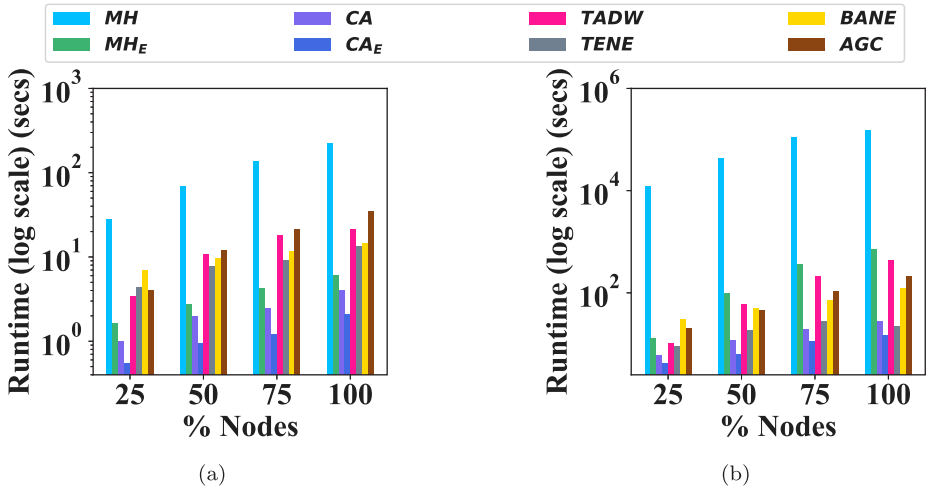


Fig. 7. Efficiency for our methods and the competitors: Runtime vs. % of nodes for: (a) CIAO and (b) EPIN.

8. Case study: Clustering phylogenetic trees

As discussed in Introduction, an edge in a sequence graph can model a relationship between users or a relationship between strings. In Section 7, we demonstrated the effectiveness of our approach in applications where edges represent relationships between users and the input data are modeled as a graph. We now proceed to presenting a case study that highlights the effectiveness of our approach when edges represent relationships between strings and the input data are modeled as a *phylogenetic tree* [24].

In particular, we consider the domain of bioinformatics and the application of evaluating the quality of a phylogenetic tree [24]. A phylogenetic tree is a rooted or unrooted leaf-labeled bifurcating (binary) tree that represents evolutionary relationships among biological organisms [24]. A phylogenetic tree can be inferred from a set of strings, each representing the genomic sequence of an organism. Each leaf of the tree corresponds to a different organism and is labeled with a string representing the genomic sequence of the organism, while each non-leaf node u corresponds to a cluster comprised of all strings associated with the leaves of the subtree rooted at u. Thus, a phylogenetic tree is a hierarchical representation of all clusterings of the strings of its leaves.

A phylogenetic tree T whose leaves correspond to a set of strings S can be constructed by different methods (e.g., by agglomerative hierarchical clustering methods [24]). To evaluate the quality of T, one can compare it with a ground truth clustering C of S. Let k be the number of clusters in C. Clearly, T cannot be compared with C directly, since the former is a binary tree (i.e., a 2D structure), whereas the latter is a partition of S into k clusters (i.e., a 1D structure). Therefore, one needs to first "flatten" T, by creating a clustering C' of its leaves that has k clusters, and then compare C' with C. If these two clusterings are similar, T is of high quality, as it accurately reflects the evolutionary relationships between the organisms corresponding to the strings of S according to the ground truth clustering.

It is easy to see that T can be modeled as a sequence graph $\mathcal{G} = (V, E, S, \mathcal{F})$ with V (respectively, E) being the set of nodes (respectively, edges) of T, S being the set of strings S corresponding to the leaves of T (i.e., the leaf labels), and $\mathcal{F}$ being a function that associates each leaf of T with is corresponding string in S and each non-leaf node in T with the empty string. Thus, we can construct C' by first clustering the sequence graph corresponding to T with k equal to the number of clusters in the ground truth clustering, and then creating, for each resultant cluster c, a cluster c' in C' that is comprised of the non-empty strings corresponding to the nodes in c. After that, we can compare C' with the ground truth clustering C using measures that compare clusterings (e.g., the measures in [57–59]).

In what follows, we first discuss the data and setup we used and then the results of our case study.

Data and Setup. We used three datasets, referred to as Ebolavirus (EBOL), Influenza (INFL), and Coronavirus (COR). The characteristics of these datasets are summarized in Table 3. In these datasets, each record is a genomic sequence of a different virus type (e.g., in EBOL, there are 59 records and each record corresponds to a different type of Ebolavirus). All genomic sequences were downloaded from the NCBI GenBank [60] based on their accession numbers provided in [61].

For each dataset, we obtained the phylogenetic tree from [62], and the ground truth clustering from the NCBI GenBank [60] using the BioPython library [63]. Specifically, each cluster in the ground truth clustering is comprised of all sequences with the same value in the *Organism* field, for EBOL and INFL, or all sequences with the same value in the last element of the *Taxonomy* field for COR (since all sequences in this dataset had the same value in *Organism*). It can be readily verified from Figs. B.9, B.10, and B.11 in Appendix B that the phylogenetic trees we used are in accordance with the ground truth clustering. That is, each ground truth cluster contains leaves that are close together in the phylogenetic tree.

Table 3
Datasets characteristics.

Dataset	# of leaves	# of edges	Avg., max. degree	Alphabet size $\lvert\Sigma_S\rvert$	Avg., max. string length	Ground truth clusters
EBOL	59	116	1.98, 3	4	18,932, 18,961	5
INFL	38	74	1.97, 3	4	1,406, 1,467	5
COR	34	66	1.97, 3	4	27,567, 31,357	9

Table 4
ACC for different methods applied with $k = 5$ on EBOL and INFL, and with $k = 9$ on COR. A × denotes that a method did not produce a score, because it did not produce k clusters. The values for the best performing method are in bold.

Methods	EBOL	INFL	COR
CA	**0.904**	**0.947**	**0.941**
CA_E	0.805	0.894	0.882
MH	0.814	0.845	0.756
MH_E	0.712	0.753	0.729
TADW	0.627	0.447	0.382
TENE	0.576	0.394	0.352
BANE	×	0.368	×
AGC	0.423	0.317	0.352

Table 5
NMI for different methods applied with $k = 5$ on EBOL and INFL, and with $k = 9$ on COR. A × denotes that a method did not produce a score, because it did not produce k clusters. The values for the best performing method are in bold.

Methods	EBOL	INFL	COR
CA	**0.894**	**0.962**	**0.957**
CA_E	0.880	0.857	0.918
MH	0.837	0.884	0.720
MH_E	0.701	0.835	0.701
TADW	0.426	0.269	0.389
TENE	0.317	0.255	0.434
BANE	×	0.313	×
AGC	0.435	0.322	0.376

Table 6
Macro-F_1 for different methods applied with $k = 5$ on EBOL and INFL, and with $k = 9$ on COR. A × denotes that a method did not produce a score, because it did not produce k clusters. The values for the best performing method are in bold.

Methods	EBOL	INFL	COR
CA	**0.901**	**0.943**	**0.886**
CA_E	0.717	0.889	0.848
MH	0.751	0.840	0.762
MH_E	0.693	0.708	0.709
TADW	0.483	0.376	0.375
TENE	0.371	0.401	0.317
BANE	×	0.321	×
AGC	0.401	0.345	0.286

We constructed a clustering C' from the sequence graph corresponding to a phylogenetic tree T by applying one of our methods (CA, CA_E, MH, and MH_E) or a competitor (TADW, TENE, BANE, AGC), configured as in Section 7.

To measure similarity between C' and the ground truth clustering, we used three well-established measures that compare similarity between two clusterings based on their labels: Clustering Accuracy (ACC) [57], Normalized Mutual Information (NMI) [58], and macro-F_1 score [59]. These measures take values in $[0, 1]$ with larger values indicating a more accurate (i.e., closer to the ground truth) clustering.

ACC is computed based on Eq. (16):

$$\mathrm{ACC}(C', C) = \frac{\sum_{i=1}^{|S|} \mathbf{1}\left(l_i = m\left(c_i\right)\right)}{|S|}, \tag{16}$$

where S is the set of strings in the input sequence graph, l_i is the ground truth label of the ith string in the input sequence graph, c_i is the id of the cluster where this string belongs in C' that is used as clustering label, $m()$ is the optimal mapping function that permutes clustering labels to match the ground truth labels,[4] and $\mathbf{1}()$ outputs 1 if its argument is true and 0 otherwise.

NMI is computed based on Eq. (17):

$$\mathrm{NMI}(C, C') = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} n_{ij} \log_2 \frac{n_{ij}}{n_i \hat{n}_j}}{\sqrt{\left(\sum_{i=1}^{k} n_i \log_2 \frac{n_i}{n}\right)\left(\sum_{j=1}^{k} \hat{n}_j \log_2 \frac{\hat{n}_j}{n}\right)}}, \tag{17}$$

where n_i denotes the number of strings in the ith cluster in C', $\hat{n}_j$ denotes the number of strings belonging to the jth cluster in the ground truth clustering C, n_{ij} is the number of strings belonging both in the ith cluster in C' and in the jth ground truth cluster, and k is the number of clusters.

The macro-F_1 measure is based on the F_1 measure. F_1 assumes a setting where there are only two different labels, namely 0 and 1, in the ground truth clustering, and it is computed based on Eq. (18):

$$F_1 = 2 \cdot \frac{\mathrm{Prec} \cdot \mathrm{Rec}}{\mathrm{Prec} + \mathrm{Rec}}, \tag{18}$$

where $\mathrm{Prec} = \frac{TP}{TP+FP}$ and $\mathrm{Rec} = \frac{TP}{TP+FN}$. In turn, TP denotes the number of strings with label 1 in both the ground truth clustering and C', while FN (respectively, FP) denotes the number of strings with label 1 (respectively, 0) in the ground truth clustering and 0 (respectively, 1) in C'. When the ground truth clustering contains L labels, the macro-F_1 measure is defined based on Eq. (19):

$$\sum_{c=1}^{L} F_1(c)/L, \tag{19}$$

where $F_1(c)$ is the F_1 score obtained for a two-label setting, in which 1 is the label of cluster c and 0 is the label of any other cluster.

We used the default parameters of Section 7 for all methods. All experiments ran on the PC mentioned in Section 7.

Clustering Quality. Since the phylogenetic trees we used are in accordance with the ground truth (see Figs. B.9, B.10, and B.11 in Appendix B), we expect that a good clustering method for evaluating a phylogenetic tree would have a high value (close to 1) in ACC, NMI, and macro-F_1. The higher the value, the better the clustering method, as the clustering it constructs is more similar to the ground truth clustering.

Tables 4, 5, and 6 show that the clusterings created by our methods are substantially more similar to the ground truth clustering compared to those created by the competitors. CA was the best performing method, outperforming MH in all tested cases, due to its objective function. Specifically, its ACC, NMI, and macro-F_1 scores were 15.6%, 15.2%, and 16% larger on average (over all datasets) than those of MH, respectively. In addition, the use of proxy measures in our methods did not substantially affect clustering quality. This is encouraging as our methods using proxy measures, namely CA_E and MH_E, are more efficient, as discussed in Section 7.

On the other hand, the competitors did not perform well. For example, the ACC, NMI, and macro-F_1 scores for CA were 91.8%, 159.5%, and 121.2% larger on average (over all datasets) than the best competitor TADW. The main reason for the poor performance of competitors is that, in the application we consider, only the leaves of a phylogenetic tree have non-empty strings associated with them, while a large number of non-leaf nodes are associated with the empty string. This leads the competitors to construct clusters with leaf nodes associated with different strings, as their assumptions (close nodes in the tree should be clustered together for AGC, and low order proximities are sufficient to cluster nodes in the tree for BANE and TENE) are invalidated. Last, note that in the case of EBOL and COR, BANE did not produce k clusters, since it learned fewer than k binary code representations.

9. Conclusion

This work introduced the problem of clustering sequence graphs and studied variants of the problem based on the k-center and k-median problems. We first proposed a product metric and a SimRank-based measure to capture distance between two nodes of a sequence graph, as well as a proxy for each measure. We then proposed an approximation algorithm and a heuristic, which generally outperform attribute-based graph clustering methods, as shown experimentally. Last, we proposed a methodology that successfully applies our measures (and the corresponding clustering algorithms) to evaluate whether a given phylogenetic tree is in accordance with a given ground truth clustering.

CRediT authorship contribution statement

Haodi Zhong: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Grigorios Loukides:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Supervision, Project administration. **Solon P. Pissis:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing – review & editing.

[4] The optimal mapping function can be computed based on the Hungarian algorithm.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Grigorios Loukides reports financial support was provided by King's College London. Haodi Zhong reports financial support was provided by King's College London. Solon P. Pissis reports financial support was provided by National Research Institute for Mathematics and Computer Science.

Acknowledgments

H. Zhong is supported by a CSC Scholarship, UK. G. Loukides is supported by the Leverhulme Trust, UK RPG-2019-399 project. S. P. Pissis is partially supported by the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 956229 and by the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 872539.

Appendix A. The impact of proxy measures on clustering quality

See Fig. A.8.

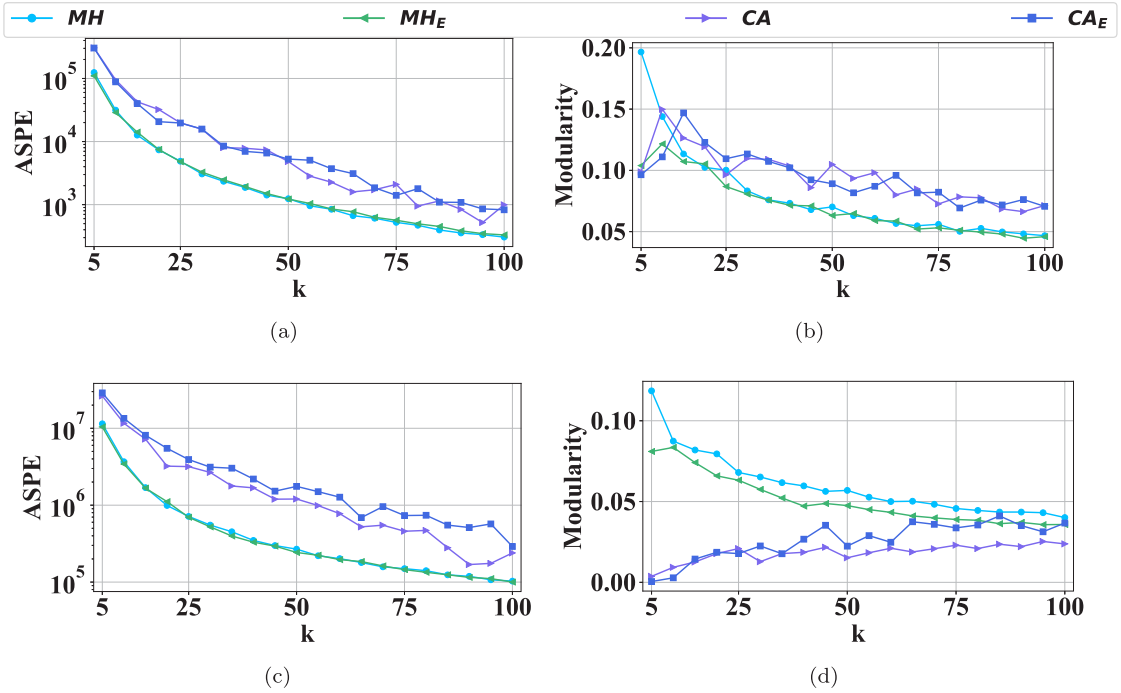


Fig. A.8. Comparison of MH and CA against MH_E and CA_E: (a) ASPE and (b) Modularity vs. k for CIAO. (c) ASPE and (d) Modularity vs. k for EPIN.

Appendix B. Phylogenetic trees with the ground truth

See Figs. B.9–B.11.

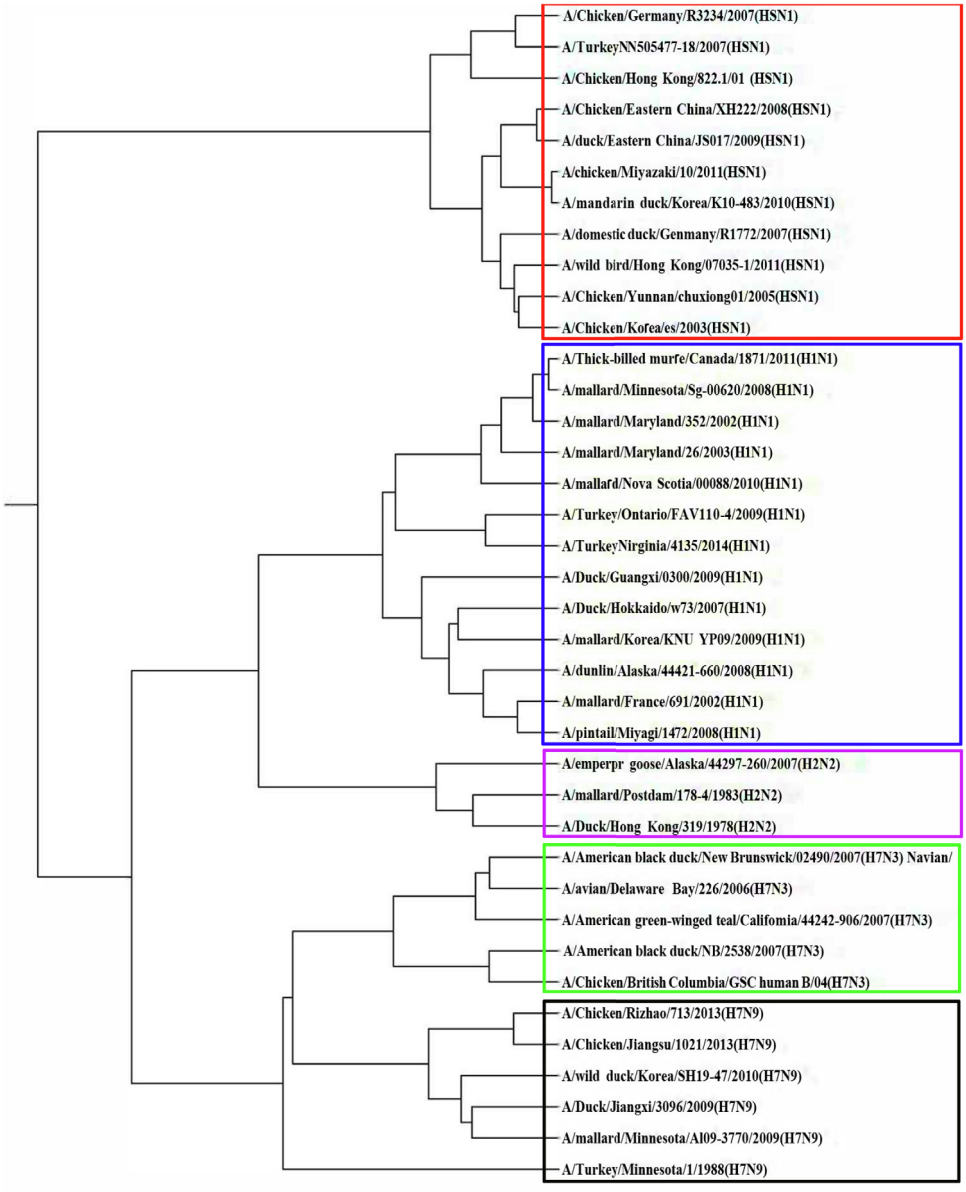


Fig. B.9. Phylogenetic tree of INFL with the ground truth. The sequences are shown as leaves. Each cluster in the ground truth clustering is represented with a differently colored rectangle.

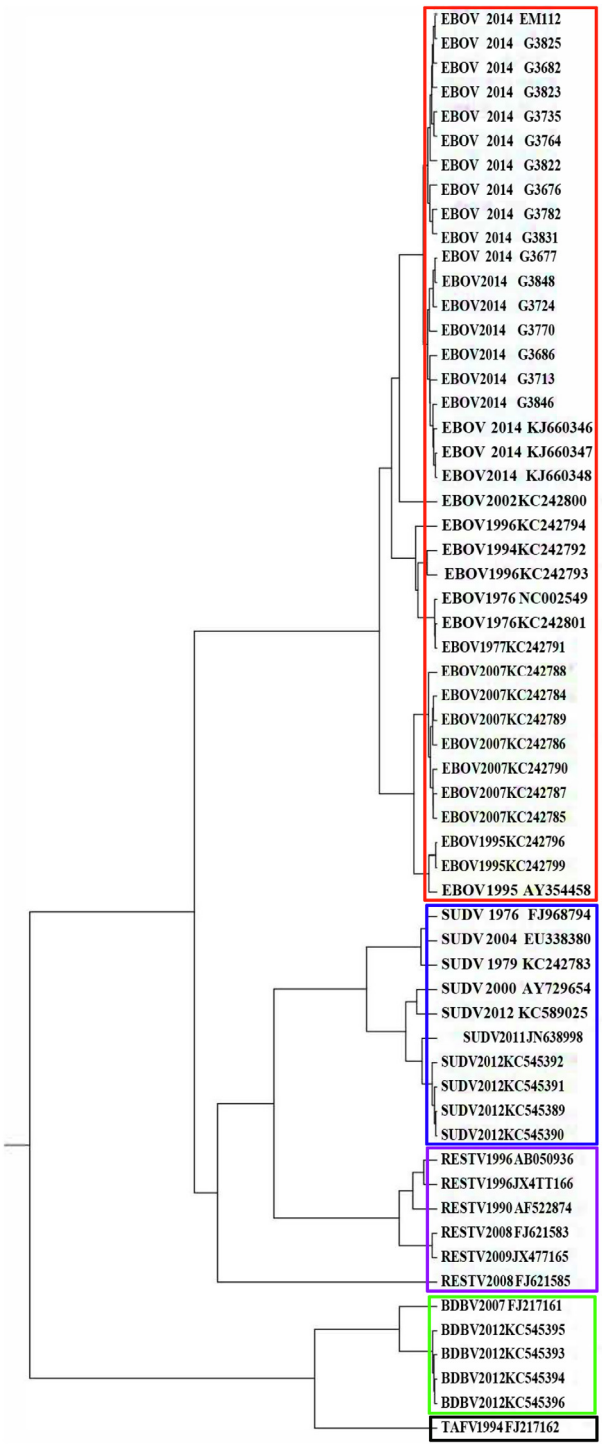


Fig. B.10. Phylogenetic tree of EBOL with the ground truth. The sequences are shown as leaves. Each cluster in the ground truth clustering is represented with a differently colored rectangle.

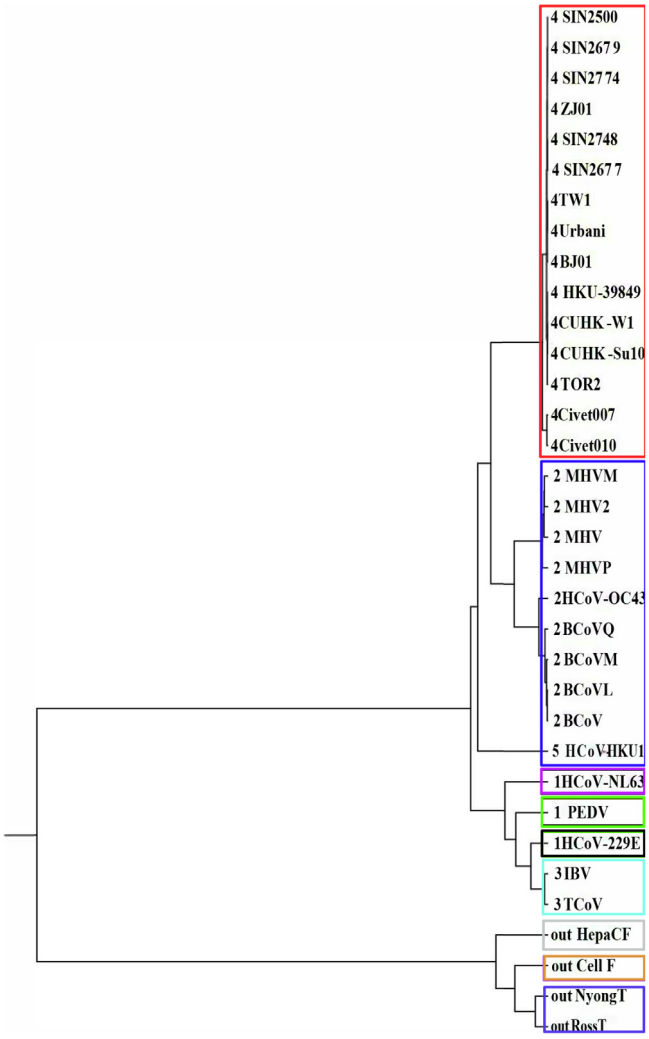


Fig. B.11. Phylogenetic tree of COR with the ground truth. The sequences are shown as leaves. Each cluster in the ground truth clustering is represented with a differently colored rectangle.

References

[1] P. Tan, M. Steinbach, A. Karpatne, V. Kumar, Introduction to Data Mining, second ed., Pearson, 2018.
[2] M.E. Newman, Modularity and community structure in networks, Proc. Nat. Acad. Sci. 103 (23) (2006) 8577–8582.
[3] G. Guo, J. Zhang, D. Thalmann, A. Basu, N. Yorke-Smith, From ratings to trust: An empirical study of implicit trust in recommender systems, in: SAC, 2014, pp. 248–253.
[4] B. Wang, A.M. Mezlini, F. Demir, M. Fiume, Z. Tu, M. Brudno, B. Haibe-Kains, A. Goldenberg, Similarity network fusion for aggregating data types on a genomic scale, Nature Methods 11 (3) (2014) 333–337.
[5] H. Gao, J. Tang, H. Liu, Exploring social-historical ties on location-based social networks, in: AAAI, 2012.
[6] X. Yu, A. Pan, L.-A. Tang, Z. Li, J. Han, Geo-friends recommendation in gps-based cyber-physical social network, in: ASONAM, 2011, pp. 361–368.
[7] Y. Matsuo, H. Yamamoto, Community gravity: measuring bidirectional effects by trust and rating on online social networks, in: WWW, 2009, pp. 751–760.
[8] H. Zhong, G. Loukides, S.P. Pissis, Clustering demographics and sequences of diagnosis codes, IEEE J. Biomed. Health Inform. (2021) http://dx.doi.org/10.1109/JBHI.2021.3129461.
[9] E.W. Myers, The fragment assembly string graph, Bioinformatics 21 (suppl_2) (2005) ii79–ii85.
[10] S. Jun, G. Sims, G.A. Wu, S. Kim, Whole-proteome phylogeny of prokaryotes by feature frequency profiles: An alignment-free method with optimal feature resolution, Proc. Natl. Acad. Sci. USA 107 (1) (2010) 133–138.
[11] R. Xia, Y. Pan, L. Du, J. Yin, Robust multi-view spectral clustering via low-rank and sparse decomposition, in: AAAI, 2014, pp. 2149–2155.
[12] C. Yang, Z. Liu, D. Zhao, M. Sun, E.Y. Chang, Network representation learning with rich text information, in: IJCAI, 2015, pp. 2111–2117.
[13] X. Wang, D. Jin, X. Cao, L. Yang, W. Zhang, Semantic community identification in large attribute networks, in: AAAI, 2016, pp. 265–271.
[14] L. Akoglu, H. Tong, B. Meeder, C. Faloutsos, PICS: Parameter-free identification of cohesive subgroups in large attributed graphs, in: SDM, 2012, pp. 439–450.

[15] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, C. Zhang, Adversarially regularized graph autoencoder for graph embedding, in: IJCAI, 2018, pp. 2609–2615.
[16] S. Yang, B. Yang, Enhanced network embedding with text information, in: ICPR, 2018, pp. 326–331.
[17] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, C. Zhang, Binarized attributed network embedding, in: ICDM, 2018, pp. 1476–1481.
[18] X. Zhang, H. Liu, Q. Li, X. Wu, Attributed graph clustering via adaptive graph convolution, in: IJCAI, 2019, pp. 4327–4333.
[19] S. Avgustinovich, D. Fon-Der-Flaass, Cartesian products of graphs and metric spaces, Eur. J. Comb. 21 (7) (2000) 847–851.
[20] G. Jeh, J. Widom, SimRank: a measure of structural-context similarity, in: KDD, 2002, pp. 538–543.
[21] D. Hochbaum, When are NP-hard location problems easy? Ann. Oper. Res. 1 (1984) 201–214.
[22] V.V. Vazirani, K-center, in: Approximation Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 47–53, http://dx.doi.org/10.1007/978-3-662-04565-7_5.
[23] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems. II: The p-Medians, SIAM J. Appl. Math. 37 (3) (1979) 539–560.
[24] T. Warnow, Computational Phylogenetics: An Introduction to Designing Methods for Phylogeny Estimation, first ed., Cambridge University Press, USA, 2017.
[25] D.H. Huson, C. Scornavacca, A survey of combinatorial methods for phylogenetic networks, Genome Biol. Evol. 3 (2011) 23–35.
[26] Q. Zou, G. Lin, X. Jiang, X. Liu, X. Zeng, Sequence clustering in bioinformatics: an empirical study, Brief. Bioinform. 21 (1) (2020) 1–10.
[27] H.-J. Li, Z. Bu, Z. Wang, J. Cao, Dynamical clustering in electronic commerce systems via optimization and leadership expansion, IEEE Trans. Ind. Inf. 16 (8) (2019) 5327–5334.
[28] L. Sheugh, S.H. Alizadeh, A novel 2D-graph clustering method based on trust and similarity measures to enhance accuracy and coverage in recommender systems, Inform. Sci. 432 (2018) 210–230.
[29] C.C. Aggarwal, H. Wang, A survey of clustering algorithms for graph data, in: Managing and Mining Graph Data, Springer, 2010, pp. 275–301.
[30] R. Xu, D. Wunsch, Survey of clustering algorithms, IEEE Trans. Neural Netw. 16 (3) (2005) 645–678.
[31] C. Bothorel, J.D. Cruz, M. Magnani, B. Micenkova, Clustering attributed graphs: Models, measures and methods, Netw. Sci. 3 (3) (2015) 408–444.
[32] J.A. Carriço, M. Crochemore, A.P. Francisco, S.P. Pissis, B. Ribeiro-Gonçalves, C. Vaz, Fast phylogenetic inference from typing data, Algorithms Mol. Biol. 13 (1) (2018) 4:1–4:14.
[33] V. Guralnik, G. Karypis, A scalable algorithm for clustering sequential data, in: ICDM, 2001, pp. 179–186.
[34] T. Xiong, S. Wang, Q. Jiang, J.Z. Huang, A new Markov model for clustering categorical sequences, in: ICDM, 2011, pp. 854–863.
[35] H.N. Djidjev, M. Onus, Scalable and accurate graph clustering and community structure detection, IEEE Trans. Parallel Distrib. Syst. 24 (5) (2013) 1022–1029.
[36] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 22 (8) (2000) 888–905.
[37] J. Laeuchli, Fast community detection with graph sparsification, in: PAKDD, 2020, pp. 291–304.
[38] J. Pei, D. Jiang, A. Zhang, On mining cross-graph quasi-cliques, in: KDD, 2005, pp. 228–238.
[39] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: KDD, 2014, pp. 701–710.
[40] B. Rozemberczki, R. Davies, R. Sarkar, C. Sutton, Gemsec: Graph embedding with self clustering, in: ASONAM, 2019, pp. 65–72.
[41] N. Shervashidze, P. Schweitzer, E.J. Van Leeuwen, K. Mehlhorn, K.M. Borgwardt, Weisfeiler-lehman graph kernels., J. Mach. Learn. Res. 12 (9) (2011).
[42] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, Knowl.-Based Syst. 151 (2018) 78–94.
[43] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, 2007.
[44] M. Potamias, F. Bonchi, C. Castillo, A. Gionis, Fast shortest path distance estimation in large networks, in: CIKM, ACM, 2009, pp. 867–876.
[45] B. Youngmann, T. Milo, A. Somech, Boosting SimRank with semantics, in: EDBT, 2019, pp. 37–48.
[46] W. Yu, W. Zhang, X. Lin, Q. Zhang, J. Le, A space and time efficient algorithm for SimRank computation, World Wide Web 15 (2012) 327–353.
[47] W. Yu, X. Lin, W. Zhang, J. Pei, J.A. Mccann, SimRank*: Effective and scalable pairwise similarity search based on graph topology, VLDBJ 28 (3) (2019) 401–426.
[48] D.A. Brannan, A First Course in Mathematical Analysis, Cambridge University Press, 2006, http://dx.doi.org/10.1017/CBO9780511803949.
[49] A. Backurs, P. Indyk, Edit distance cannot be computed in strongly subquadratic time (Unless SETH is false), in: STOC, 2015, pp. 51–58.
[50] D. Chakraborty, E. Goldenberg, M. Koucký, Streaming algorithms for embedding and computing edit distance in the low distance regime, in: STOC, 2016, pp. 712–725.
[51] T.F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoret. Comput. Sci. 38 (1985) 293–306.
[52] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1) (1974) 168–173.
[53] M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time, J. ACM 46 (3) (1999) 362–394.
[54] H.-S. Park, C.-H. Jun, A simple and fast algorithm for K-medoids clustering, Expert Syst. Appl. 36 (2009) 3336–3341.
[55] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, C. Zhang, Attributed graph clustering: A deep attentional embedding approach, in: IJCAI, 2019, pp. 3670–3676.
[56] H. Zhang, Q. Zhang, Embedjoin: Efficient edit similarity joins via embeddings, in: KDD, 2017, pp. 585–594.
[57] Y. Yang, D. Xu, F. Nie, S. Yan, Y. Zhuang, Image clustering using local discriminant models and global integration, IEEE Trans. Image Process. 19 (10) (2010) 2761–2773.
[58] A. Amelio, C. Pizzuti, Is normalized mutual information a fair measure for comparing community detection methods? in: ASONAM, 2015, pp. 1584–1585.
[59] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[60] National Center for Biotechnology Information (NCBI), https://www.ncbi.nlm.nih.gov/.
[61] Y. Li, L. He, R.L. He, S.S.-T. Yau, Supplementary material of 'A novel fast vector method for genetic sequence comparison', Sci. Rep. 7 (1) (2017) 1–11, https://static-content.springer.com/esm/art%3A10.1038%2Fs41598-017-12493-2/MediaObjects/41598_2017_12493_MOESM1_ESM.pdf.
[62] Y. Li, L. He, R.L. He, S.S.-T. Yau, A novel fast vector method for genetic sequence comparison, Sci. Rep. 7 (1) (2017) 1–11.
[63] J. Chang, Biopython: Tutorial and Cookbook, Self-publishing, 2020.

Haodi Zhong is a Ph.D. student at King's College London. His research interests are in data mining and biomedical informatics.

Grigorios Loukides is an Associate Professor at King's College London. His research interests are in data privacy, data mining, and biomedical informatics.

Solon P. Pissis is a Senior Researcher at CWI. His research focuses on theory of algorithms and their application in data mining.