



ripple2vec: Node Embedding with Ripple Distance of Structures

Jizhou Luo¹ · Song Xiao¹ · Shouxu Jiang¹ · Hong Gao¹ · YINUO Xiao¹

Received: 26 June 2021 / Revised: 8 February 2022 / Accepted: 21 March 2022 / Published online: 15 April 2022
© The Author(s) 2022

Abstract

Graph is a generic model of various networks in real-world applications. And, graph embedding aims to represent nodes (edges or graphs) as low-dimensional vectors which can be fed into machine learning algorithms for downstream graph analysis tasks. However, existing random walk-based node embedding methods often map some nodes with (dis)similar local structures to (near) far vectors. To overcome this issue, this paper proposes to implement node embedding by constructing a context graph via a new defined ripple distance over ripple vectors, whose components are the hitting times of fully condensed neighborhoods and thus characterize their structures as pure quantities. The distance is able to capture the (dis)similarities of nodes' local neighborhood structures and satisfies the triangular inequality. The neighbors of each node in the context graph are defined via the ripple distance, which makes the short random walks from a given node over the context graph only visit its similar nodes in the original graph. This property guarantees that the proposed method, named as ripple2vec, is able to map (dis)similar nodes to (far) near vectors. Experimental results on real datasets, where labels are mainly related to nodes' local structures, show that the results of ripple2vec behave better than those of state-of-the-art methods, in node clustering and node classification, and are competitive to other methods in link prediction.

Keywords Node embedding · Structural similarity · Ripple distance

1 Introduction

Graph is a generic model of various networks in real-world applications, *e.g.*, social network analysis [22], knowledge representation and inference [18], chemoinformatics and computational biology [24]. Effective graph analyses, often time- and space-consuming tasks due to the nature of graph data, help users better understand the structure of societies [22], languages [28], molecules and many others, which benefits lots of meaningful operations such as node classification [30], item recommendation [21], information propagation

[29] and performance prediction [2]. Many approaches have been proposed to perform such analyses [20–22, 25]. At the same time, recent years have witnessed a surge of successes in processing data with machine learning (ML) algorithms, which operate on structured data and often require to represent symbolic data as numerical vectors. In order to analyze graph data with ML algorithms, people turn to graph embedding, *i.e.*, representing nodes [4–7, 9] (edges [12] and graphs [11], *resp.*) in low-dimensional vector spaces such that the vectors of similar structures have small distances. This paper focuses on node embedding with direct embedding methods.

The goal of node embedding, given a graph $G(V, E)$ and an integer $d \in N$, is to find a mapping $f : V \rightarrow R^d$ such that $dis_G(u, v) \approx \|f(u) - f(v)\|_2$, where $dis_G(u, v)$ is the similarity between u and v . Usually, it is difficult to define dis_G to quantify precisely the semantic relationships and similarities between nodes in $G(V, E)$ [26]. For example, general dis_G s are adopted in traditional algorithms such as multi-dimensional scaling (MDS) [33], IsoMap [32], Laplacian eigenmap [31], and local linear embedding (LLE) [34], which suffer from high space and time cost. And, new ideas are developed in recent algorithms [4–9, 13, 15, 16].

Some algorithms follow the framework of random walk [4, 6] or its variants [13, 14, 17] to measure the proximities

✉ Jizhou Luo
luojizhou@hit.edu.cn

Song Xiao
15683609851@163.com

Shouxu Jiang
jsx@hit.edu.cn

Hong Gao
honggao@hit.edu.cn

YINUO Xiao
xiaoyinuo@hit.edu.cn

¹ Harbin Institute of Technology, Harbin, China

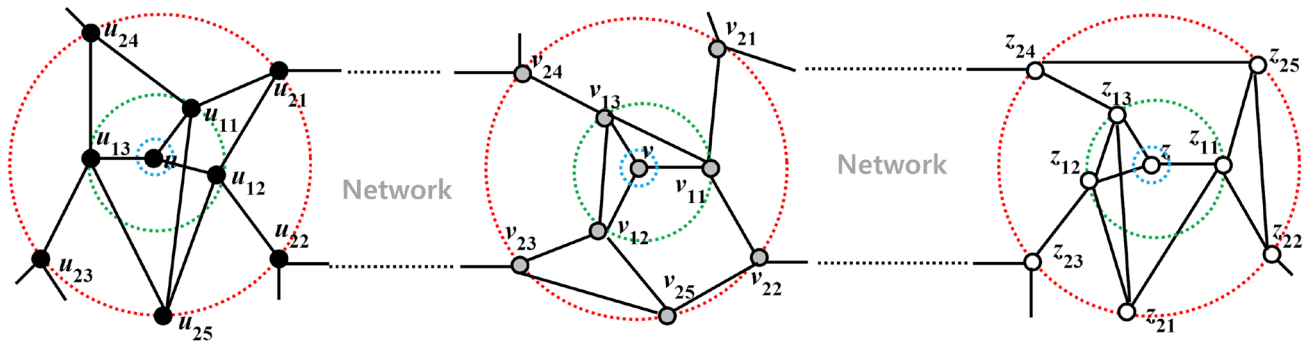


Fig. 1 Local structures caught by ripple vectors

between nodes with their co-occurrence on short random walks over the graph. However, such co-occurrences are based on the adjacency relationships instead of the similarity, which makes it hard to map nodes with similar local structures to vectors with small distances.

Instead, people sample random walks from context graphs, which are constructed via different similarity measures of local structures. For example, HONE [8] and RUM [23] construct weighted motif adjacency graphs by counting the occurrences of a set of network motifs. Such methods require user to provide suitable network motifs, which is usually a non-trivial task. Also, HARP [16] constructs a hierarchy of coarsened graphs by collapsing related nodes into “supernodes.” It still depends on the adjacency relationships of the original graph, while struc2vec [7] applies DTW algorithm on degree sequences to measure the similarity of local structures. Notice that DTW algorithm ignores partially the effects of connection patterns within neighborhoods. As a result, nodes with similar local structures may be mapped to far vectors.

Therefore, there is still much space to improve the way to capture the similarity of local neighborhoods. For example, consider the first (second *resp.*) -order neighborhoods $N_1(u)$, $N_1(v)$ ($N_2(u)$, $N_2(v)$ *resp.*) of nodes u , v in Fig. 1. Obviously, there are edges in both $N_1(v)$ and $N_2(v)$, but no edges in neither $N_1(u)$ nor $N_2(u)$. That is, $N_i(u)$ and $N_i(v)$ ($i = 1, 2$) have different local structures, which cannot be captured by applying DTW algorithm on degree sequences (like in [7]) because $N_i(u)$ and $N_i(v)$ have a same sequence. Besides, such differences cannot be easily captured explicitly by neither predefined graphlets (like in [8]) nor other existing methods, to the best of our knowledge.

Based on this observation and several wonderful existing ideas [1, 4, 7, 16], this paper proposes a metric (named as ripple distance) to capture above differences and leverages it to build a context graph (for node embedding), which helps map nodes with (dis)similar local structure to (far) near vectors.

Specifically, we first describe the local neighborhood structures of each node u as a ripple vector $r(u)$, where

each component $r_i(u)$ quantifies approximately the structure of the i th-order neighborhood $N_i(u)$ as the expected hitting time of a random walk in the Fully Condensed Abstract Neighborhood Graph $F_i(u)$. The intuitive meaning of $r_i(u)$ is the average steps of the random walk, which is determined inherently by the structure of $N_i(u)$, to escape $N_i(u)$ from $N_0(u) \cup \dots \cup N_{i-1}(u)$. For instance, Fig. 2 illustrates the FCANGs of $N_1(u)$, $N_1(v)$, $N_2(u)$ and $N_2(v)$ of node u , v in Fig. 1. Furthermore, $r(u) = \langle 3, 2.67, 7.2 \rangle$ and $r(v) = \langle 3, 4.0, 4.62 \rangle$ (See Sect. 4.2 for their computation). As a whole, ripple vector catches the structure of the nested local neighborhoods of each node, just like the shapes of water waves in diffusion sense the positions of the obstacles on the surface of the water.

Then, we propose to use ripple distance $r(u, v)$, which is defined as $\sum_0^k w_i (1 - \frac{\min(r_i(u), r_i(v))}{\max(r_i(u), r_i(v))})$, to measure the (dis)similarity of nodes' local structures. The configurability of weights brings a flexibility of our method to different applications. And ripple distance satisfies the triangular inequality, no matter how the weights are chosen. This lays a solid foundation for ripple2vec to map (dis)similar nodes to (far) near vectors.

Finally, with the help of the ripple distance, we construct a multilayer context graph G_c to implement node embedding by sampling short random walks from it. Node u in each i -th layer of G_c has $\log |V|$ neighbors corresponding to $\log |V|$ similar nodes (measured with the i -th-order ripple distance) of u in the original graph, which makes the co-occurring nodes in short random walks have high similarity. Efficient algorithm to compute the context graph is designed to guarantee its scalability.

Our main contributions include:

- We quantify the local structures of nodes in (un)weighted graphs as ripple vectors and define the ripple distance over them to measure the (dis)similarity between nodes.
- We implement a node embedding (named as ripple2vec) in the framework of struc2vec by adapting ripple distance

Table 1 Convention of notations

Notation	Description
$G(V, E)$	The input graph with node set V and edge set E
d_{max}	The maximum degree of G
$N_i(u)$	The i -th-order neighborhood of $u \in V$
$F_i(u)$	Fully condensed abstract neighborhood graph of $N_i(u)$
$T_i(u)$	Hitting time from source to sink in $F_i(u)$ or $G_i(u)$
$r_i(u)$	The i -th-order ripple vector of $u \in V$
$r_i(u, v)$	The i -th-order ripple distance between of u and v
$G_c(V_c, E_c)$	The multilayer context graph

to define the context graphs. It helps map (dis)similar nodes to (far) near vectors.

- Extensive experiments are conducted on real datasets to show the superiority of the ripple2vec, *i.e.*, it is stronger to capture explicitly the similarity of local structures, scales well and benefits downstream applications well.

Notations used in this paper are summarized in Table 1.

The remainder is organized as below. Section 2 summarizes related work. Section 3 formalizes the problem and presents explain our motivation further. Our solution is given in Sect. 4. And, Sect. 5 reports the experimental results, followed by the conclusion in Sect. 6.

2 Related Work

There are different types of node embedding algorithms (see [3, 26] for surveys and classification). Our method belongs to direct encoding algorithms.

2.1 Traditional Methods

Traditional dimension reduction algorithms, such as EigenMap [31], MDS [33], LLE [34] and IsoMap [32], can be used as node embedding algorithms. All these algorithms can be adapted to work coordinately with our ripple distance. For example, EigenMap and MDS can be applied directly on graph $G(V, E')$, where E' consists of all node pairs with their ripple distances as weights. And, LLE and IsoMap can be applied on graph $G(V, E'')$, where $E'' \subset E'$ only contains edges between each node with its k -nearest neighbors under the ripple distance. However, such adaptations suffer from high time cost of $\Omega(|V|^2)$ and are not affordable.

2.2 Random Walk-Based Methods

Many recent successful direct embedding algorithms take advantage of the framework of random walk (or its variants), which was first proposed in DeepWalk [4]. Its innovation is

to stochastically measure the proximity between nodes with their co-occurrence on short random walks over a graph. DeepWalk generated random walks in a DFS manner, while node2vec [6] in a BFS manner.

Instead of sampling random walks over the original graph, HONE [8] seizes the similarities of local structures by counting the occurrences of a set of network motifs (*i.e.*, graphlets or orbits) to form a weighted motif adjacency graph and then applies random walk approaches (*e.g.*, DeepWalk and node2vec) on it. Also, HARP [16] realizes a node embedding by generating iteratively a hierarchy of coarsened graphs (by collapsing related nodes into “supernodes”) and then applying random walk approaches (*e.g.*, DeepWalk and node2vec) on these coarsened graphs in an inverse order. Notice that the idea of using coarsened supernode to characterize local structures is explored in the definition of the ripple vector. Likewise, struc2vec [7] defines a multilayer graph with a recursively defined distance to formalize the similarities between local structures of nodes in the original graph and then applies on the multilayer graph the idea of random walk such that it has more opportunities to visit nodes with similar local structures. struc2vec utilizes DTW distance on ordered degree sequences of $N_k(u), N_k(v)$ to define the similarity of local structures of u, v , while ripple distance uses the difference between particular hitting times of the coarsened neighborhoods. This integration of collapsing neighborhood and random walking makes ripple distance more powerful in capturing the similarity of local structures. Moreover, the similar expressions of the distances in ripple2vec and struc2vec make it possible to adapt the framework of struc2vec to the ripple distance directly.

There are some other extensions of the random walk idea. For example, HOPE [13] extends DeepWalk to learn embeddings by using random walks that “skip” or “hop” over multiple nodes at each step, resulting in a proximity measure similar to GraRep [14], while Chamberlan et al. [17] modify the inner product of node2vec to use a hyperbolic distance measure (rather than Euclidean distance). Our ripple2vec aims to leverage the local structures caught precisely by the ripple distances and thus is perpendicular to these methods.

2.3 Other Methods

LINE [5] expresses the likelihood of the first and second proximity with the sigmoid function of the inner products of target vectors directly and then learns the target vectors by optimizing a loss function derived from the KL divergence metric between the empirical distribution. However, its strong assumption of conditional independence (*i.e.*, $Pr[N(u)|u] = \prod_{v_i \in N(u)} Pr[v_i]$) makes it hard to catch the dissimilarities between nodes of the same neighborhood with different structures, as ripple distance does.

SDNE [9] and DNGR [10] are representatives of deep learning methods (see [3, 26] for surveys). They associate each

node with a neighborhood vector which contains the proximities between this node and other nodes and learn node embedding with deep neural networks. DNGR defines the vector according to the pointwise mutual information of two nodes co-occurring on random walks, similar to DeepWalk [4] and node2vec [6]. SDNE simply chooses the node's adjacency vector as the vector. Both vectors may be replaced with the ripple distance vector (left as our future work).

Graph convolutional neural networks (GCNs) [30, 36] obtain node embedding by aggregating the features of neighbor nodes either in spectral domain or vertex domain. They leverage the local structures implicitly in downstream applications (e.g., [30]), while ripple2vec aims to exploit local structures explicitly. Graph kernel methods (see [19] for a recent survey) usually map graph to vectors by counting the appearance of substructures such as paths and trees, which is perpendicular to ripple2vec.

3 Problem Statement and Motivation

Node embedding aims to represent nodes of a graph in a low-dimensional Euclidean vector space. Formally, given a graph $G(V, E)$ with node set V and edge set E , the goal of node embedding is to learn a mapping function $f : V \rightarrow R^d$ ($d \ll |V|$) such that the distance $\|f(u) - f(v)\|_2$ explicitly preserves the similarity between the local structures of arbitrary $u, v \in V$.

The key point of node embedding is that the similarity between local neighborhood structures of nodes must be preserved essentially. However, different understandings of the size and the structure of the local neighborhood result in different definitions of the similarity, which further have great impacts on the quality of node embedding. For example, HONE [8] counts the occurrences of predefined graphlets to measure the similarity. DeepWalk [4] and node2vec [6] leverage the co-occurrences of nodes in short random walks to measure the similarity. SDNE [9] uses the first- and the second-order proximity to measure the similarity. struc2vec [7] aligns the sorted degree sequences to measure the similarity, and so on. These popular methods have two main disadvantages: (1) The similarity measures do not satisfy the triangular inequality in any way; (2) dissimilar nodes are often mapped to near vectors.

These issues motivate us to seek a new metric to capture the local neighborhood similarities such that (1) The dissimilarities (like in Fig. 1) not caught by existing methods can be easily captured by the new method; (2) it helps us realize a node embedding who maps similar nodes to near vectors and dissimilar nodes to far vectors with acceptable computation cost. This can be accomplished by requiring the new metric satisfies the triangular inequality. That is, nodes with more similar local structures have smaller distance, while nodes with more dissimilar local structures have larger distances. Since node embedding preserves similarities, a node embedding implemented with the new metric naturally maps (dis)similar nodes to (far) near vectors. To obtain such an embedding, we construct a multilayer context graph such that each node is adjacent to its most similar nodes. Therefore, the pioneering random walk framework over the context graph generates the context for each node (consisting of its most similar nodes) and results in an expected node embedding.

In summary, we devote to solve the following problem:

Input A weighted or unweighted graph $G(V, E)$, integer $d > 0$.

Output A learned mapping function: $f : V \rightarrow R^d$ such that nodes with (dis)similar local neighborhood structures are mapped to (far) near vectors.

Method Adapting the popular random walk framework to a context graph $G_c(V_c, E_c)$ derived from new similarity measure satisfying triangular inequality in some way.

4 The Proposed Model

This section presents our solution of the node embedding problem. The framework is presented in Subsection 4.1, and the key steps will be expanded in Subsections 4.2 and 4.3.

4.1 Framework of ripple2vec

The framework of ripple2vec (See Alg. 1) follows the generic framework of struc2vec [7] and runs in $O(d_{max}^K |V| + K^2 |V| \log^2 |V|)$ time.

Algorithm 1: ripple2vec ($G(V, E), D$)

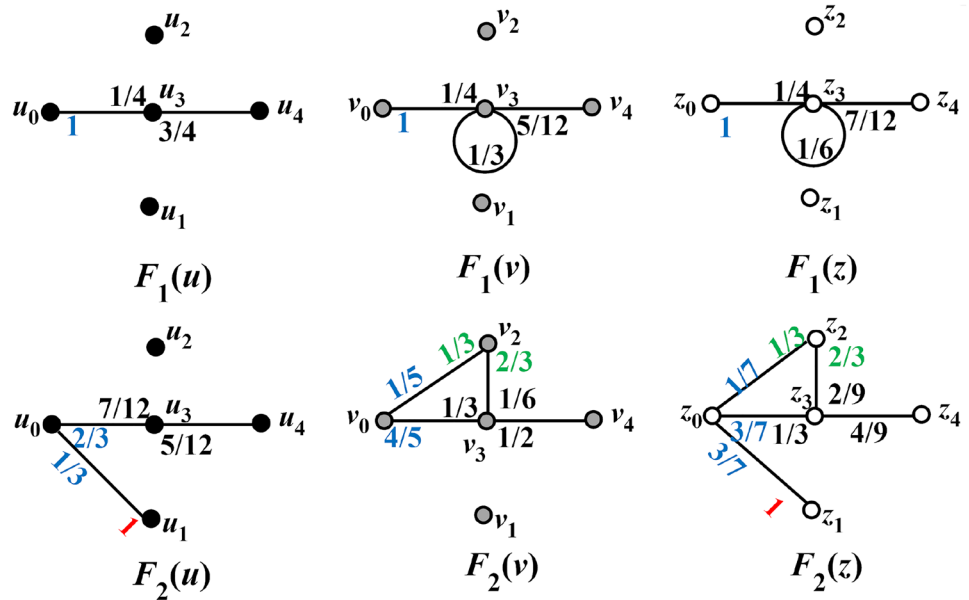
Input: Graph $G(V, E)$, dimension D of target vectors

Output: Embedded vectors $vector_{|V| \times D}$ of all nodes

```

1  $R_{|V| \times (K+1)} \leftarrow rippleVector(G(V, E), K);$  // Sec. 4.2
2  $G_c(V_c, E_c) \leftarrow contextGraph(V, K, R);$  // Sec. 4.3
3 for  $u \in V$  do
4    $context(u) \leftarrow p$  short random walks, with length at most  $l$ , starting from  $u$  in
     the  $K$ -th layer of  $G_c$ .
5  $vectors \leftarrow word2vec(\cup_{u \in V} context(u));$ 
6 return  $vectors$ ;
```

Fig. 2 The FCANGs of nodes u, v, z in Fig. 1



It first computes, by invoking `rippleVector` (See Subsection 4.2), a $K+1$ -dimensional vectors for all nodes to depict their K th-order local neighborhood structures (Line 1). Usually, $K = 4$ is enough (See Sect. 5), because ripple vectors catch approximately the local neighborhood structures well, and higher order of local structures is often meaningless as shown in [6].

Then, it invokes `contextGraph` (see Sect. 4.3) to construct a weighted multilayer context graph $G_c(V_c, E_c)$ (Line 2), in which all nodes in V appear in each layer. In each i -th layer, u has exactly $\log |V|$ neighbors, which are the nearest neighbors in the i -th-order ripple distance. Edge weights among every node pair within each layer are defined in a same way as `struc2vec`.

After that, it generates context for each node by sampling p random walks with length at most l (Line 3, Line 4). Notice that each random walk from u contains only u 's similar nodes from G .

Finally, it accomplishes the mapping with the well-known learning algorithm `word2vec` [28] (Line 5).

4.2 Ripple Vectors and Ripple Distance

This subsection develops a metric, which is integrated into `ripple2vec` in next subsection, to measure the similarity of the structures of the local neighborhoods of any pair of nodes. Our statement focuses on unweighted graphs, although it can be directly extended to weighted graphs. Subsection 4.2.1 introduces the ripple vectors. And, Subsection 4.2.2 discusses the ripple distance.

4.2.1 Ripple Vector

Given a graph $G(V, E)$, let $\text{diam}(G)$ be the diameter of G , i.e., the maximum length of the shortest paths between two nodes of G . For $\forall u \in V$ and $0 \leq k \leq \text{diam}(G)$, the collection of vertices exactly k -hop away from u in G is called as the k -th-order neighborhood of u and denoted as $N_k(u)$. For example, consider the graph in Fig. 1. We have $N_0(u) = \{u\}$, $N_1(u) = \{u_{11}, u_{12}, u_{13}\}$, and $N_2(u) = \{u_{21}, u_{22}, u_{23}, u_{24}, u_{25}\}$.

Intuitively, we intend to quantify the structure of $N_k(u)$ as a single value $T_k(u)$. And, all $T_k(u)$ s, together with $d_u(G)$, consist of the ripple vector $r(u)$ of u . Therefore, if $N_k(u)$ is viewed as a ripple k -hop away from u and $T_k(u)$ quantifies the structure of $N_k(u)$, then the ripple vector $r(u)$ characterizes the whole structure of $N_0(u) \cup \dots \cup N_{\text{diam}(G)}(u)$, just like water waves in diffusions on the surface of the water area.

The structure of each $N_k(u)$ ($0 < k < \text{diam}(G)$) is determined by three kinds of edges, i.e., the edges between $N_{k-1}(u)$ and $N_k(u)$, the edges between $N_k(u)$ and $N_{k+1}(u)$, and edges within $N_k(u)$. In what follows, we first represent this structure approximately by constructing a 5-node weighted undirected graph $F_k(u)$, which is called as Fully Condensed Abstract Neighborhood Graph (or FCANG in short), and then quantify it as a single value.

$F_k(u)$ has 5 nodes u_0, u_1, u_2, u_3, u_4 . Each u_i is a subset of V .

- $u_0 = N_0(u) \cup \dots \cup N_{k-1}(u)$;
- $u_4 = N_{k+1}(u)$;
- $u_3 = \{v \in N_k(u) \mid (v, x) \in E \text{ for some } x \in u_4\}$;
- $u_2 = \{v \in N_k(u) - u_3 \mid (v, v') \in E \text{ for some } v' \in u_3\}$;
- $u_1 = N_k(u) - u_3 - u_2$;

Edges in $F_k(u)$ can be added as below. If $u_i \neq \emptyset$ ($i = 1, 2, 3$), then there is an edge (u_0, u_i) . If there is an edge in $G(V, E)$ with both endpoints from u_i , then there is a self-circle at u_i . Moreover, if $u_2 \neq \emptyset$ ($u_3 \neq \emptyset$ resp.), then edge (u_2, u_3) ((u_3, u_4) resp.) exists. Actually, each edge (u_i, u_j) in $F_k(u)$ represents a type of connections near $N_k(u)$. Therefore, it can be weighted as the number n_{ij} of edges in $G(V, E)$ with one endpoint from u_i and the other from u_j .

Example 1 Figure 2 illustrates the first- and the second-order FCANGs of nodes u, v, z in Fig. 1, where the weights are not given explicitly. Particularly, in $F_2(z)$, $z_1 = \{z_{21}\}$, $z_2 = \{z_{25}\}$ and $z_3 = \{z_{22}, z_{23}, z_{24}\}$. Since $z_i \neq \emptyset$ and there is no edge in $G(V, E)$ with both endpoints from z_i ($i = 1, 2, 3$), the edge set of $F_2(z)$ is $\{(z_0, z_1), (z_0, z_2), (z_0, z_3), (z_2, z_3), (z_3, z_4)\}$. In contrast, in $F_1(z)$, $z_1 = \emptyset$, $z_2 = \emptyset$ and $z_3 = N_1(z)$. Since edge (z_{12}, z_{13}) of $G(V, E)$ has both endpoints from z_3 , the edge set of $F_2(z)$ is $\{(z_0, z_3), (z_3, z_3), (z_3, z_4)\}$.

The structure of the FCANG $F_k(u)$ can be quantified in the mass as the expected hitting time of random walk in $F_k(u)$ from u_0 to u_4 , which intuitively is the average random walk steps to escape from $N_k(u)$. It is well known that the hitting time is inherently affected by the structure of $F_k(u)$.

Definition 1 In graph $G(V, E)$, for $\forall u \in V$ and $1 \leq k \leq \text{diam}(G)$, the k -th-order escaping time of u , denoted as $T_k(u)$, is defined as the expected hitting time of random walk from u_0 to u_4 in $F_k(u)$.

Definition 2 In graph $G(V, E)$, for $\forall u \in V$ and $1 \leq k \leq \text{diam}(G)$, the k -th-order ripple vector of u , denoted as $r_k(u)$, is defined as the vector $\langle d_G(u), T_1(u), \dots, T_k(u) \rangle$.

$T_k(u)$ ($k > 0$) can be computed with the closed formula derived in [1]. This can be done by first filling the state transition probability matrix $B_k(u)$ such that $B_k(u)(a, b) = \frac{w(a, b)}{\sum_{(a, b') \in E_k(u)} w(a, b')}$ and then taking $T_k(u)$ as the sum of u 's row of matrix $(I - B_k(u))^{-1}$, which can be computed in $O(1)$ time. Specifically, given edge weights $n_{01}, n_{02}, n_{03}, n_{11}, n_{22}, n_{32}, n_{33}, n_{43}$ in $F_k(u)$, the matrix $B_k(u)_{4 \times 4}$

can be constructed by updating 0 matrix as below. For the first row, $B_k(u)(0, a) = \frac{n_{0a}}{n_{01} + n_{02} + n_{03}}$ for $a = 1, 2, 3$. For the second row, if $n_{01} > 0$, then $B_k(u)(1, 0) = \frac{n_{01}}{n_{01} + 2n_{11}}$ and $B_k(u)(1, 1) = \frac{2n_{11}}{n_{01} + 2n_{11}}$. For the third row, if $n_{02} > 0$, then $B_k(u)(2, a) = \frac{n_{a2}}{n_{02} + 2n_{22} + n_{32}}$ for $a = 0, 3$ and $B_k(u)(2, 2) = \frac{2n_{22}}{n_{02} + 2n_{22} + n_{32}}$. For the last row, $B_k(u)(3, a) = \frac{n_{a3}}{n_{03} + n_{23} + 2n_{33} + n_{43}}$ for $a = 0, 2$, and $B_k(u)(3, 3) = \frac{2n_{33}}{n_{03} + n_{23} + 2n_{33} + n_{43}}$.

Example 2 Consider the FCANG $F_2(z)$ in Fig. 2. We count the edges in Fig. 1 and obtain $n_{01} = 3$, $n_{02} = 1$, $n_{03} = 3$, $n_{11} = 0$, $n_{22} = 0$, $n_{32} = 2$, $n_{33} = 0$ and $n_{43} = 4$. Therefore, we have

$$B_2(z) = \begin{matrix} & \begin{matrix} z_0 & z_1 & z_2 & z_3 \end{matrix} \\ \begin{matrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{matrix} & \begin{pmatrix} 0 & \frac{3}{7} & \frac{1}{7} & \frac{3}{7} \\ 1 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{2}{3} \\ \frac{1}{3} & 0 & \frac{2}{9} & 0 \end{pmatrix} \end{matrix}$$

as shown in Fig. 2. Thus, $T_2(z) = ((I - B_2(z))^{-1}(1, 1, 1, 1)^T)_1 = 8.5$. Similarly, we have $T_1(z) = 3.14$. Therefore, the second-order ripple vector of z is $r_2(z) = \langle 3, 3.14, 8.5 \rangle$. Likewise, $r_2(u) = \langle 3, 2.67, 7.2 \rangle$, and $r_2(v) = \langle 3, 4.0, 4.62 \rangle$. According to these vectors, u, v, z in Fig. 1 can be distinguished from each other easily.

Algorithm 2 computes the K th-order ripple vectors for all nodes by extending the procedure above directly. It is noticeable that n_{ii} is increased when each endpoint of an edge is considered. Therefore, factor 2 disappears in the formula to update $B_i(u)$. Now notice that the inverse of $B_i(u)$ can be computed in $O(1)$, each $T_i(u)$ is obtained in $O(|N_i(u)|)$ time. Since $|N_i(u)| \leq d_{\max}^i$ where $d_{\max}$ is the maximum degree in $G(V, E)$, Alg. 2 runs in $O(d_{\max}^K |V|)$ time. For example, when $K = 2$, it runs in $O(d_{\max}^2 |V|)$ time.

Remark Ripple vector can also be defined without splitting $N_k(u)$ into u_1, u_2 and u_3 , which makes matrix $B_k(u)$ have rank $|N_k(u)|$, and Alg. 2 have prohibitively high cost to apply.

Algorithm 2: rippleVector ($G(V, E), K$)**Input:** Graph $G(V, E)$, dimension K of ripple vectors**Output:** FCRVs of nodes in V with dimension at most K

```

1  $k \leftarrow K$ ;
2 for each  $u \in V$  do
3    $T_0(u) = d_G(u)$ ,  $N(u) = \{u\}$ ,  $i \leftarrow 0$ ;
4   repeat
5      $i \leftarrow i + 1$ ;
6      $N_i(u) \leftarrow \{v | v \in V, (u, v) \in E, v \notin N_u\}$ ;
7     If  $N_i(u) = \emptyset$  then break;
8      $u_4 \leftarrow V - N(u) - N_i(u)$ ;
9      $u_3 \leftarrow \{v \in N_i(u) | (v, x) \in E \text{ for some } x \in u_4\}$ ;
10     $u_2 \leftarrow \{v \in N_i(u) - u_3 | (v, v') \in E \text{ for some } v' \in u_3\}$ ;
11     $u_1 \leftarrow N_i(u) - u_2 - u_3$ ;  $u_0 \leftarrow N(u)$ ;
12    let  $n_{01}, n_{02}, n_{03}, n_{11}, n_{22}, n_{32}, n_{33}, n_{43}$  be 0s;
13    for each  $v$  in  $N_i(u)$  do
14      for each  $(v, x)$  in  $E$  do
15        switch  $v \in u_p, x \in u_q$  do
16          case  $\langle q, p \rangle$ :  $n_{qp} \leftarrow n_{qp} + 1$ ;
17     $B \leftarrow 0_{4 \times 4}$ ;  $t_1 \leftarrow n_{01} + n_{11}$ ;  $t_2 \leftarrow n_{02} + n_{22} + n_{23}$ ;
18     $t_3 \leftarrow n_{03} + n_{23} + n_{33} + n_{43}$ ;
19     $B(0, p) \leftarrow \frac{n_{0p}}{n_{01} + n_{02} + n_{03}}$  for  $p = 1, 2, 3$ ;
20    if  $n_{01} > 0$  then  $B(1, a) \leftarrow \frac{n_{a1}}{t_1}$  for  $a = 0, 1$ ;
21    if  $n_{02} > 0$  then  $B(2, a) \leftarrow \frac{n_{a2}}{t_2}$  for  $a = 0, 2, 3$ ;
22     $B(3, 0) \leftarrow \frac{n_{03}}{t_3}$ ;  $B(3, 2) \leftarrow \frac{n_{23}}{t_3}$ ;  $B(3, 3) \leftarrow \frac{n_{33}}{t_3}$ ;
23     $B \leftarrow (I - B)$ 's inverse matrix;
24     $T_i(u) \leftarrow$  sum the 1st row of  $B$ ;
25     $N(u) \leftarrow N(u) \cup N_i(u)$ ;
26  until  $N_i(u) = \emptyset$  or  $i > k$ ;
27 return  $\langle T_0(u), T_1(u), \dots, T_k(u) \rangle$  for  $\forall u \in V$ ;

```

4.2.2 Ripple Distance

We first define the distance and then prove it satisfies the triangular inequality and present some implications.

The ripple vector quantifies node's hierarchical local structures as its components. Therefore, the difference between ripple vectors, which is a weighted sum of the difference of each component, measures the similarity of local structures of nodes. To obtain 0 distance for identical vectors, the difference of the corresponding component of two ripple vectors considers the ratio of the minimal value to the maximal value.

Definition 3 In graph $G(V, E)$, for $\forall u, v \in V$ and $0 \leq k \leq \text{diam}(G)$, the k -th-order ripple distance between u and v w.r.t weights $\mathbf{w} = \langle w_1, \dots, w_k \rangle$ ($w_i > 0$), denoted as $r_k^{(w)}(u, v)$ or $r_k(u, v)$ in short when $\mathbf{w}$ is clear, is defined as $\sum_{i=0}^k w_i \cdot [1 - \frac{\min(r_i(u), r_i(v))}{\max(r_i(u), r_i(v))}]$.

The ripple distance can focus on the differences in one or several decomposed neighborhoods by adjusting the aggregation weights, which results in a great flexibility of its applications. We do not discussed how to set the weights and take $\mathbf{w} = \langle \frac{100}{k+1}, \frac{100}{k+1}, \dots, \frac{100}{k+1} \rangle$ for $r_k(u, v)$ in all experiments.

No matter what $\mathbf{w}$ is, the distances satisfy the triangular inequality.

Theorem 1 If $T_k(u) > 0$ for $\forall u \in V$ in graph $G(V, E)$, then the ripple distance is really a distance of the vector set $\{r_k(u) | r_k(u) = \langle T_0(u) = d_G(u), T_1(u), \dots, T_k(u) \rangle, u \in V\}$.

Proof Obviously, we have (1) $r_k(u, v) \geq 0$; (2) $r_k(u, v) = r_k(v, u)$; (3) $r_k(u, v) = 0$ if and only if $r_k(u) = r_k(v)$. We only need to verify the triangular inequality.

First of all, notice that $1 - \frac{\min(a, b)}{\max(a, b)} - \frac{\min(b, c)}{\max(b, c)} + \frac{\min(a, c)}{\max(a, c)} \geq 0$ for $\forall a, b, c > 0$. In fact, assume $a \leq b \leq c$ without loss of generality, we have $1 - \frac{\min(a, b)}{\max(a, b)} - \frac{\min(b, c)}{\max(b, c)} + \frac{\min(a, c)}{\max(a, c)} = 1 - \frac{a}{b} - \frac{b}{c} + \frac{a}{c} = (1 - \frac{a}{b})(1 - \frac{b}{c}) \geq 0$.

For $\forall x, y, z \in V$, we have $r_k(x, y) + r_k(y, z) - r_k(x, z) = \sum_{i=0}^k w_i \cdot [1 - \frac{\min(T_i(x), T_i(y))}{\max(T_i(x), T_i(y))} - \frac{\min(T_i(y), T_i(z))}{\max(T_i(y), T_i(z))} + \frac{\min(T_i(x), T_i(z))}{\max(T_i(x), T_i(z))}] \geq 0$.

Remark (1) With the ripple distance rather than the Euclidean distance, ripple vectors generated by Alg. 2 realize a node embedding. (2) Ripple distances between $\forall u, v \in V$ form a similarity matrix, on which the traditional dimension reduction algorithms can be applied to implement a node embedding in $O(|V|^2)$ time.

4.3 Multi-layer Context Graph

Algorithm 3: contextGraph ($R_{|V| \times (K+1)}, G(V, E)$)

Input: Ripple vectors $R_{|V| \times (K+1)}$, graph $G(V, E)$
Output: constructed multi-layer context graphs $G_c(V_c, E_c)$

```

1 for  $i \leftarrow 0$  To  $K$  do
2    $L_i \leftarrow \{\langle R_{j,i}, j \rangle | j \in V\}$  sorted in ascending order of  $R_{j,i}$ ;
3   for each  $u \in V$  do                                     // find neighbors in all layers
4      $D[] \leftarrow \{\}$ ;  $Rounds \leftarrow 0$ ;
5     for  $i \leftarrow 0$  To  $K$  do                               // each layer
6        $count \leftarrow 0$ ;  $new \leftarrow \emptyset$ ;  $L_i.scanInit(R_{u,i})$ ;
7       for  $j \leftarrow 0$  to  $Rounds$  do                       // missed scans in  $L_i$ 
8          $v \leftarrow L_i.next()$ ;
9         if  $v \notin D[]$  then  $new.append(\langle r_i(u, v), v \rangle)$ ;
10      for  $\langle dis, v, * \rangle$  in  $D[]$  do                         // update distances
11        update  $\langle dis, v, * \rangle$  to  $\langle dis + w_i \cdot (1 - \frac{\min(R_{u,i}, R_{v,i})}{\max(R_{u,i}, R_{v,i})}), v, N \rangle$ ;
12        if  $\frac{\min(R_{v,j}, R_{u,j})}{\max(R_{v,j}, R_{u,j})} \geq L_j.th$  for  $\forall j \leq i$  then
13           $count++$ ; update  $\langle dis, v, N \rangle$  to  $\langle dis, v, Y \rangle$ ;
14      for  $\langle dis, v \rangle$  in  $new$  do                             // append new nodes
15         $D.append(\langle dis, v, N \rangle)$ ;
16        if  $\frac{\min(R_{v,j}, R_{u,j})}{\max(R_{v,j}, R_{u,j})} \geq L_j.th$  for  $\forall j \leq i$  then
17           $count++$ ; update  $\langle dis, v, N \rangle$  to  $\langle dis, v, Y \rangle$ ;
18      while  $count < \log_2 |V|$  and  $|D| < (i+1) \cdot \log_2 |V|$  do
19        for  $j \leftarrow 0$  to  $i$  do                             // parallel scan
20           $v \leftarrow L_j.next()$ ;
21          if  $v \notin D[]$  then  $D.append(\langle r_i(u, v), v, N \rangle)$ ;
22          for  $\langle dis, v, N \rangle \in D$  do
23            if  $\frac{\min(R_{v,j}, R_{u,j})}{\max(R_{v,j}, R_{u,j})} \geq L_j.th$  for  $\forall j \leq i$  then
24               $count++$ ; update  $\langle dis, v, N \rangle$  to  $\langle dis, v, Y \rangle$ ;
25           $Rounds++$ ;
26      add  $\log_2 |V|$  nodes in  $D$  with smallest  $dis$  as the neighbors of  $u$  in layer  $i$ ;
27 Add edges between different layers;                       // as in struc2vec

```

This subsection presents the algorithm contextGraph, which constructs for ripple2vec (see Subsection 4.1) a multilayer context graph $G_c(V_c, E_c)$ such that (1) the number of layers is the same as the dimension of the ripple vectors (see Subsection 4.2); (2) each node u in $G(V, E)$ has a counterpart node in each i -th layer with $\log |V|$ nodes of smallest ripple distances $r_i(u, \cdot)$ as its neighbors; and (3) counterpart nodes of the same $u \in V$ in different layers are connected as in the context graph of struc2vec [7]. So, the key of contextGraph is to find the neighborhood $N^{(i)}(u)$ of each $u \in V$ in each i -th layer.

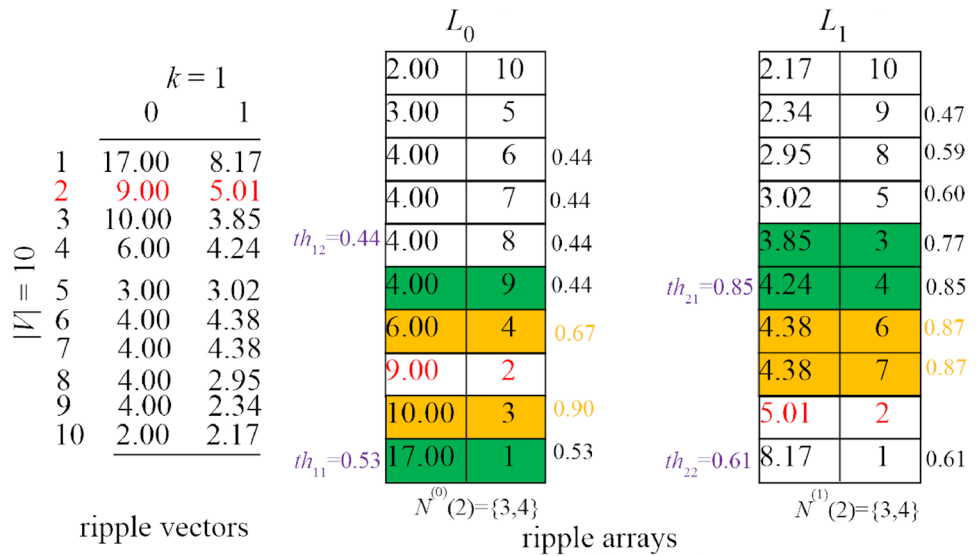
A straightforward way to complete this task is to compute, for each i , $r_i(u, v)$ for all v s and take $\log |V|$ nearest nodes among them as u 's neighbors in the i -th layer. However, this method takes a prohibitively high cost of $O(|V|^2)$.

Fortunately, the ripple distance is a ‘‘monotone’’ function in the sense that the more similar the local structures are, the smaller the distance between them is, which is implied by the triangular inequality. And, this monotonicity can be utilized to speed up the computation with the help of the threshold algorithm [35], which aims to find t largest

aggregation results from a list of sorted arrays by scanning them parallelly until a stop condition is satisfied. During parallel scanning, aggregated results of all seen objects are obtained via random access of data. These results are stored, and the highest ones are managed with a max-heap of a fixed size t . Once there are t seen objects whose scores in each array are higher than the current lowest thresholds of the same arrays, the scan terminates and objects in the heap are the final results. We refer the readers to [35] to learn more details. We adapt the threshold algorithm to compute $G_c(V_c, E_c)$ by reorganizing each component of all ripple vectors as a sorted array and scanning all arrays parallelly and bidirectionally.

Precisely, let $R_{|V| \times (K+1)}$ be the ripple vectors and L_i be the sorted array storing $\{\langle R_{u,i}, u \rangle | u \in V\}$ for $0 \leq i \leq K$. Additionally, L_i has some attributes to memorize information. For instance, $L_i.tVal$ is the value which defines the starting position of the parallel scan in L_i . $L_i.up$ ($L_i.down$ resp.) is the position of the parallel scan in upward (downward resp.) direction. $L_i.uRatio$ and $L_i.dRatio$ are the corresponding ratio at position $L_i.up$ and $L_i.down$. $L.th$ memorizes the

Fig. 3 Illustration of the running of Alg. 3



current lowest threshold of the scan in L . Each array can use `scanInit()` to initialize the scan with a given value and `next()` to get next tuple of the scan. These two procedures are as below.

Procedure $L.scanInit(tVal)$

1. $p \leftarrow \text{index of } tVal \text{ in } L$;
2. $L.dRatio \leftarrow (p > 1) ? \frac{L[p-1].ripple}{tVal} : 0$;
3. $L.uRatio \leftarrow (p < |V|) ? \frac{tVal}{L[p+1].ripple} : 0$;
4. $L.up \leftarrow p + 1$; $L.down \leftarrow p - 1$;
5. $L.th \leftarrow \max(L.uRatio, L.dRatio)$; $L.tVal \leftarrow tVal$;

Procedure $L.next()$

1. if $L.uRatio \geq L.dRatio$ then
2. $L.up \leftarrow L.up + 1$; $p \leftarrow L.up$;
3. $L.uRatio \leftarrow (L.up \leq |V|) ? \frac{L.tVal}{L[L.up].ripple} : 0$
4. Else
5. $L.down \leftarrow L.down - 1$; $p \leftarrow L.down$;
6. $L.dRatio \leftarrow (L.down \geq 1) ? \frac{L[L.down].ripple}{L.tVal} : 0$
7. $L.th \leftarrow \max(L.dRatio, L.uRatio)$;
8. return $L[p].node$;

Algorithm `contextGraph` is sketched in Alg. 3. It first transcribes ripple vectors $R_{|V| \times (K+1)}$ into sorted arrays $L_0, \dots, L_K$ (Line 1-Line 2). Then, it computes the neighborhood $N^{(i)}(u)$ in each $layer_i$ for each $u \in V$ (Line 3-Line 26). To do so, it initializes D to record the seen neighbors and *Rounds* to account the rounds of the parallel scan (Line 4). After that, it computes $\log |V|$ nearest neighbors with $r_i(\cdot, \cdot)$ for $i = 0, 1, \dots, K$ sequentially (Line 6-Line 25) and takes $\log_2 |V|$ nodes with smallest distance with u as u 's neighbors

in $layer_i$ (Line 26). Finally, it adds edges between different layers in a same way as in `struc2vec`.

When computing $N^{(i)}(u)$, it first scans *Rounds* nodes in L_i (Line 7-Line 9), which guarantees that a same number of nodes in each of $L_0, \dots, L_i$ have been scanned. The new met nodes are recorded in *new*. Then, it updates each $i - 1$ -th-order ripple distance in D to the i -th-order distances and rechecks the condition: $\frac{\min(R_{v,j}, R_{u,j})}{\max(R_{v,j}, R_{u,j})} \geq L_j.th$ holds for each $j \leq i$ and counts the number of seen nodes v which satisfy this condition (Line 10-Line 13). After that, it computes the i -th-order distances for all nodes in *new*, appends them to D and counts the number of nodes satisfying the same condition (Line 14-Line 17). If $count \geq \log |V|$, i.e., at least $\log |V|$ seen nodes satisfy the condition, then $N^{(i)}(u)$ has been obtained. Else, the algorithm scans parallelly all arrays $L_0, \dots, L_i$ until $count \geq \log |V|$ (Line 18-Line 25). For each new met node v , it appends $\langle r_i(u, v), v, N \rangle$ to D (Line 19-Line 21). After each round of parallel scan, it checks for each unlabeled node in D whether it satisfies the condition and increases *count* if necessary (Line 22-Line 24).

Notice that the condition $|D| < (i + 1) \times \log_2 |V|$ in Line 18 limits the rounds executed in parallel scans. Therefore, the algorithm only finds for each node $\log |V|$ near neighbors rather than exact $\log |V|$ nearest neighbors, which guarantees its scalability (see Sect. 5).

Example 3 Ignore the second condition in Line 18, Fig. 3 illustrates the computation of the 2-nearest neighbors of node 2, whose ripple vector and positions are all marked with red color, in $layer_0$ and $layer_1$ with Alg. 3 and the given data. The i -th components of all ripple vectors are sorted in array L_i ($i = 0, 1$).

The computation of $N^{(0)}(2)$ scans L_0 only. After two nodes are scanned (as the yellow shaded cells in L_0), both H and D record $\{\langle 3, 0.1 \rangle, \langle 4, 0.33 \rangle\}$. And, $L_0.th = th_{11} = 0.53$. Since $\frac{\min(R_{v,0}, R_{2,0})}{\max(R_{v,0}, R_{2,0})} \geq 0.53$ holds for both $v = 3$ and $v = 4$, we know $N^{(0)}(2) = \{3, 4\}$.

The computation of $N^{(1)}(2)$ scans both L_0 and L_1 . Since two nodes in L_0 have been scanned when $N^{(0)}(2)$ is computed, we first scan two nodes in L_1 (shown as the yellow shaded cells in L_1). And, two new nodes are met, *i.e.*, $new = \{6, 7\}$. After we update distances in D and append nodes 6, 7 to D , we get $D = \{\langle 3, 0.33 \rangle, \langle 4, 0.48 \rangle, \langle 6, 0.69 \rangle, \langle 7, 0.69 \rangle\}$, and $H = \{\langle 3, 0.33 \rangle, \langle 4, 0.48 \rangle\}$. At this time, $L_1.th = th_{21} = 0.85$. Since $\frac{\min(R_{v,0}, R_{2,0})}{\max(R_{v,0}, R_{2,0})} < 0.53$ for $v = 6, 7$ and $\frac{\min(R_{v,1}, R_{2,1})}{\max(R_{v,1}, R_{2,1})} < 0.85$ for $v = 3, 4$, the algorithm has to scan L_0 and L_1 parallelly further. After two rounds of parallel scan, we get the results shown as the green shaded cells in L_0 and L_1 . $L_0.th = th_{12} = 0.44$ and $L_1.th = th_{22} = 0.61$, H does not change, and D becomes $\{\langle 3, 0.33 \rangle, \langle 4, 0.48 \rangle, \langle 6, 0.69 \rangle, \langle 7, 0.69 \rangle, \langle 1, 0.86 \rangle, \langle 9, 1.09 \rangle\}$. Now, since both $\frac{\min(R_{v,0}, R_{2,0})}{\max(R_{v,0}, R_{2,0})} \geq 0.44$ and $\frac{\min(R_{v,1}, R_{2,1})}{\max(R_{v,1}, R_{2,1})} \geq 0.61$ hold for $v = 3, 4$, we get $N^{(1)}(2) = \{3, 4\}$.

Algorithm 3 runs in $O(K^3|V| \log^2 |V|)$ time, where K is the dimension of ripple vectors. In fact, there are K layers in G_c . Each node in a layer has $\log |V|$ neighbors in that layer. Computation of these $\log |V|$ neighbors for counterpart nodes in different layers is shared according to the definition of the ripple distance (see Line 5-Line 25), where at most $K \log |V|$ Kth-order ripple distances are computed, each of which needs $O(K \log |V|)$ operations ($|D| < (K + 1) \log |V|$). Totally, Line 1-Line 26 spend $O(K^2|V| \log^2 |V|)$ time. Line 27 adds edges across successive layers. Since there are totally $2K|V|$ such edges (see [7]) and the weight of each edge is determined in $\log |V|$ time, Line 27 needs a total time of $O(K|V| \log |V|)$.

5 Experiments

This section evaluates ripple2vec's performance on various settings. Subsection 5.1 evaluates the capability to capture the (dis)similarities of local structures. Subsection 5.2 evaluates performance in classical tasks. Subsection 5.3 evaluates performance in network alignment. Subsection 5.4 shows the effects of the dimension K of ripple vectors and the method of setting weights. Subsection 5.5 evaluates its scalability.

We compare ripple2vec with 7 algorithms, *i.e.*, DeepWalk [4], node2vec [6], LINE [5], SDNE [9], struc2vec [7], GCN [36] and GAlign [37]. To evaluate the roles of Alg. 3, we adapt the ripple distance to struc2vec without any other changes (labeled as ripple + struc2vec). Additionally,

ripple2vec with the second condition in Line 17 of Alg. 3 canceled (labeled as ripple + unlimited) is also considered.

All algorithms are implemented in Python 3 and run with five threads on an Inspur Server with Intel Xeon 128x2.3GHz CPU and 3Tb RAM running CentOS7 Linux as its operating system.

5.1 Capability to Capture Similarities

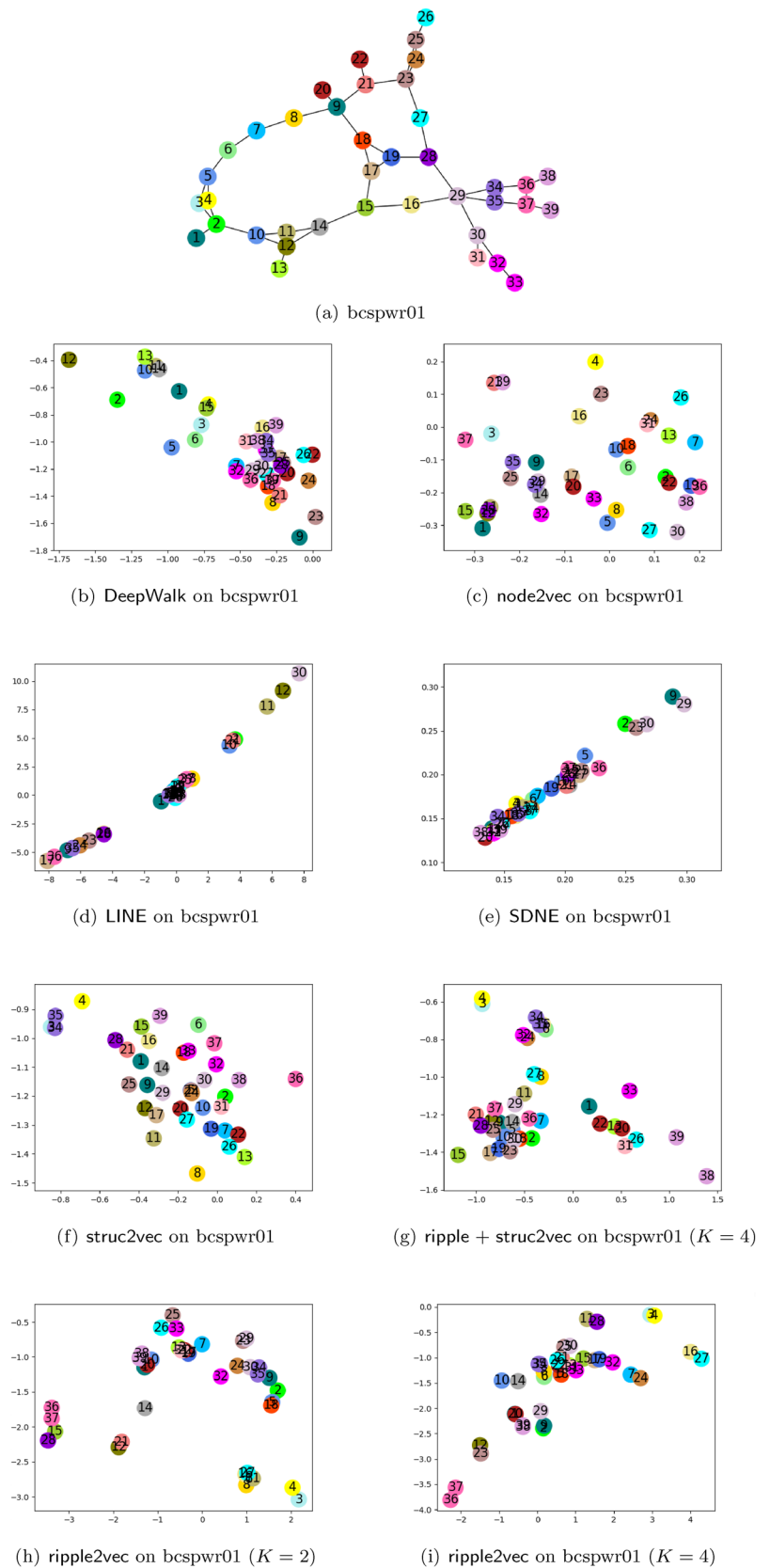
We conduct experiments on bcspwr01¹ in Fig. 4(a) to evaluate ripple2vec's capability to capture (dis)similarities of the local neighborhood structures. Bcspwr01 depicts the skeleton of a fish with 39 vertices and 46 edges (see Fig. 4(a)). Since all these vertices play different roles on the body of the fish, most of them have different local neighborhood structures. We expect that nodes with similar neighborhood structures (*e.g.*, 34 and 35) are mapped to near points in the plain, while dissimilar nodes are mapped to distinguishable points. Experimental results are shown in Fig. 4, from which we have following two observations.

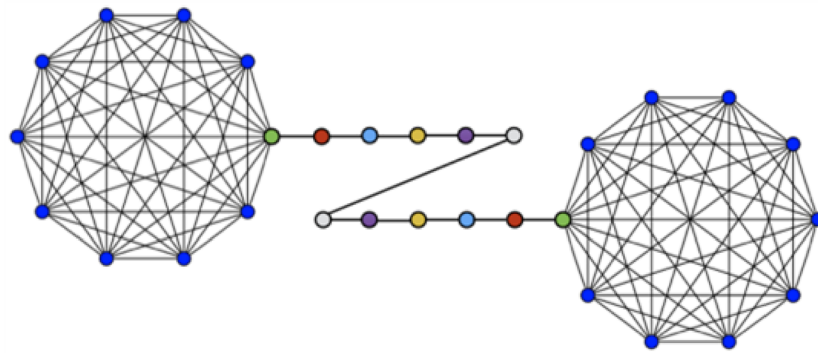
Firstly, ripple2vec performs better than these state-of-the-art methods. In fact, for each of the existing methods, it maps some nodes with similar local neighborhood structures to further vectors and some dissimilar nodes to nearer vectors. For instance, nodes 38 and 39 have the same decomposed neighborhoods, which are different from those of nodes 16, 31 and 34 (see Fig. 4a). However, in the node embedding f generated by DeepWalk, $f(38)$ is closer to $f(16)$, $f(31)$, $f(34)$ than to $f(39)$ (see Fig. 4b). Similar phenomenon can also be observed for SDNE (see Fig. 4e). For node2vec, $f(38)$ and $f(39)$ are almost the farthest vectors (see Fig. 4(c)). Similarly, nodes 34 and 35 with a same local structure are mapped to further vectors by LINE (see Fig. 4(d)). Their weaker abilities stem from the fact that they are designed to draw the context of nodes without explicit consideration of structural similarity. On the contrary, struc2vec applies DTW algorithm on the sorted degree sequences of the decomposed neighborhoods to catch the (dis)similarity of local structures. Thus, it scatters all nodes better than the previous methods (see Fig. 4f). However, some dissimilar nodes (such as 18 and 33) are mapped to nearer vectors, while some similar nodes (such as 38 and 39) are mapped to further vectors. In contrast, ripple2vec uses ripple distance over ripple vectors to capture the (dis)similarity of local structures. It almost always maps similar nodes to near vectors and dissimilar nodes to far vectors, no matter what the value of K is (see Fig. 4h) and 4(i)).

Secondly, we find that if we replace the distance in the framework of struc2vec with the ripple distance, then the

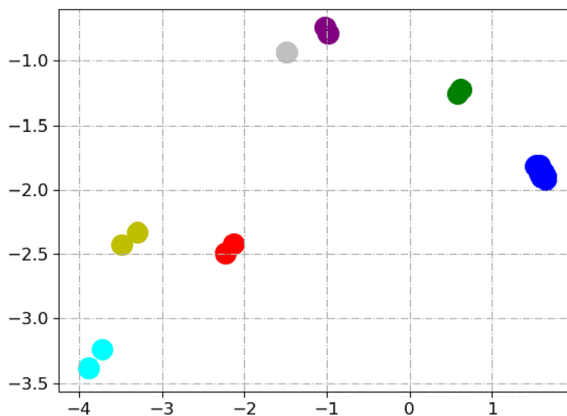
¹ http://yifanhu.net/GALLERY/GRAPHS/GIF_SMALL/HB@bcspwr01.html

Fig. 4 2D embeddings of bcspwr01 obtained by different algorithms





(a) barbell (2,10)



(b) struc2vec on barbell

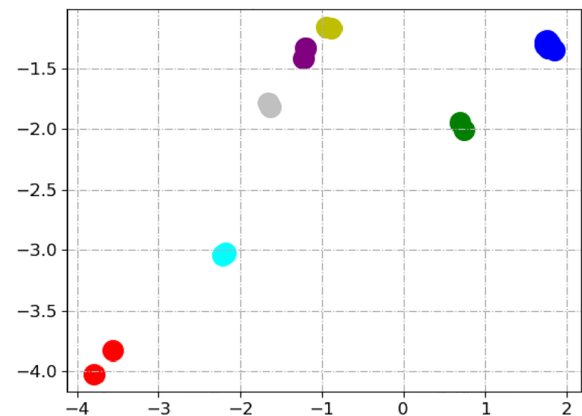

 (c) ripple2vec on barbell ($K = 4$)

Fig. 5 2D embeddings of barbell obtained by different algorithms

resulting method (*i.e.*, ripple + struc2vec) has a weaker ability to capture similarities of local neighborhood structures (see Fig. 4(g)) than ripple2vec. In this case, some similar nodes (e.g., 3 and 4, 38 and 39) are mapped to further vectors than those in ripple2vec. This indicates that the context graph constructed in Sect. 4.3 is meaningful.

Both observations can be verified on the Barbell-(2,10) and the mirrored karate. Figures 5 and 6 present only partial results. For full results, we refer the readers to our technique report. Barbell-(2,10) [7] is the graph obtained by connecting two 10-cliques with a path of 10 nodes (Fig. 5a). Both struc2vec and ripple2vec can distinguish the differences in the local neighborhood structures (5b and 5c). The mirrored karate [7] is the graph obtained by connecting two isomorphic karate networks, where ripple2vec behaves much better than struc2vec. For example, ripple2vec almost always maps any pair of mirrored nodes to the same vectors, while struc2vec does not (*e.g.*, nodes 12 and 67, 30 and 36). Besides, some dissimilar nodes not captured by struc2vec are caught by ripple2vec (*e.g.*, nodes 35 and 36, 38 and 39).

5.2 Performance in Classical Analysis Tasks

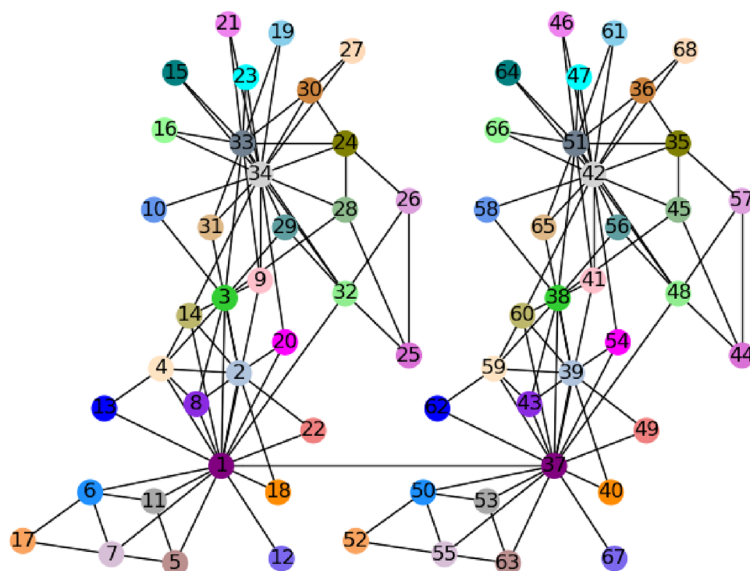
The capability of ripple2vec to map nodes with similar local structures to near vectors can be leveraged for node clustering when the clusters of nodes are more related to their local structures than to other features. To verify this, we conduct experiments on three air-traffic networks: Brazilian air-traffic network², American air-network³ and European air-traffic network⁴, which are unweighted, undirected networks with nodes corresponding to airports, edges indicating the existence of commercial flights and the label on each node marking the airport's level of activity (measured in flights and people, totally 4 levels are used). Table 2 reports the statistics of each dataset (see [7] and its GitHub link⁵ for more details). We conduct experiments to observe the behaviors of ripple2vec in node classification and link prediction.

² <http://www.anac.gov.br/>

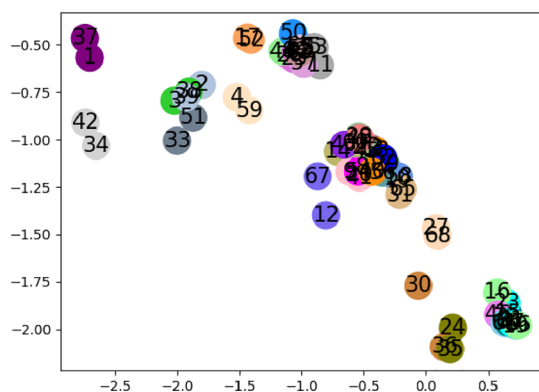
³ <https://transtats.bts.gov/>

⁴ <http://ec.europa.eu/>

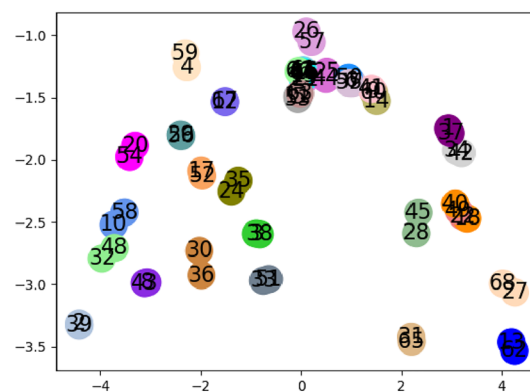
⁵ <https://github.com/leoribeiro/struc2vec>



(a) Karate-mirrored Network



(b) struc2vec on Karate-mirrored

(c) ripple2vec on Karate-mirrored ($K = 4$)**Fig. 6** 2D embeddings of karate-mirrored obtained by different algorithms**Table 2** Statistics of networks

Networks	Nodes	Edges
Brazil	131	1038
USA	1190	13599
Europe	399	5995

All algorithms map nodes to 128-dimensional vectors. For LINE and SDNE, we take default parameters. For DeepWalk, node2vec, struc2vec and ripple2vec, we take 80 walks with length 40 for each node and 8 as Skip-Gram window in word2vec. Both optimizations in [7] are considered. GCN takes the identity matrix as its input features. We fix $K = 4$ (see Subsection 5.4 for changing K). The embedded vectors of each dataset are partitioned into training data and test data in different ways by setting each $i \in [0, 10]$ as a

random seed correspondingly. Each experiment is repeated 5 times, and the average behaviors are reported.

For node clustering, we run k -means algorithm on the embedded vectors to cluster all nodes into 4 classes. The performance is evaluated as the adjusted Rand index (ARI), which ranges in $[-1, 1]$ with larger values indicating stronger consistencies between the clustering results and the actual clusters. Table 3 reports the experimental results. Notice that ripple2vec outperforms other algorithms remarkably, which verifies the facts that ripple2vec is stronger to map nodes with similar local structures into near vectors.

For node classification, we take labels in training data as known and classify nodes in test data by finding their k -NN ($k = 11$) neighbors in training data. The performance is evaluated as average accuracy and average macro F_1 score. Table 4 reports the results. Notice that ripple2vec is

Table 3 Average adjusted Rand index of k-means algorithm on node embeddings generated by different algorithms

Dataset	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
Brazil	ARI	0.4887	0.03	0.1526	0.0187	0.1599
	Metric	ripple-unlimited	ripple + struct	SDNE	struc2vec	
	ARI	0.4951	0.4433	0.2884	0.4581	
USA	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
	ARI	0.2181	0.0899	0.1399	0.0777	0.2093
	Metric	ripple-unlimited	ripple + struct	SDNE	struc2vec	
Europe	ARI	0.2855	0.2078	0.2609	0.2358	
	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
	ARI	0.3255	0.0026	0.1038	0.0173	0.1937
	Metric	ripple-unlimited	ripple + struct	SDNE	struc2vec	
	ARI	0.3224	0.2459	0.2036	0.1943	

Bold values mark the winners of the compared algorithms on each dataset

Table 4 Performance of node classification with k -NN algorithm on node embeddings generated by different algorithms

Dataset	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
Brazil	acc	0.7966	0.4074	0.2929	0.3501	0.4552
	macroF1	0.7926	0.3689	0.2519	0.3222	0.4131
	Metric	ripple-unlimited	ripple + struct	SDNE	struc2vec	
	acc	0.7953	0.7872	0.4006	0.7764	
	macroF1	0.7893	0.7780	0.3306	0.7671	
	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
USA	acc	0.6325	0.5385	0.4174	0.4747	0.6061
	macroF1	0.5988	0.5282	0.4150	0.4702	0.5875
	Metric	ripple-unlimited	ripple + struct	SDNE	struc2vec	
	acc	0.6208	0.6116	0.6321	0.6144	
	macroF1	0.5912	0.6028	0.6237	0.6067	
	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
Europe	acc	0.5652	0.4568	0.4340	0.3613	0.5616
	macroF1	0.5399	0.4482	0.3995	0.3532	0.5293
	Metric	ripple-unlimited	ripple + struct	SDNE	struc2vec	
	acc	0.5572	0.5341	0.4125	0.5347	
	macroF1	0.5537	0.5280	0.3889	0.5255	

Bold values mark the winners of the compared algorithms on each dataset and each metric

competitive to struc2vec on all datasets. It is better than GCN on Brazil and competitive to GCN on other datasets. And, it always outperforms all other algorithms. These observations verify further the facts that ripple2vec is stronger to map nodes with similar local structures into near vectors.

For link prediction, we train a logistic regression model with the training datasets and use it to predict the edges in the corresponding test datasets. The performance is evaluated as average accuracy and average area under curve (AUC). Table 5 reports the experimental results. Notice that ripple2vec is competitive to all other algorithms, although the logistic regression model uses nonlinear activation function to eliminate some advantages of mapping nodes with similar local structures into near vectors.

Moreover, in all three experiments above, we notice that the second condition in Line 17 of Alg. 3 has a little bit

of impacts on the performance (comparing the results of ripple2vec with ripple + unlimited). However, it affects the scalability of the algorithm greatly. On the contrary, adapting directly the ripple distance to the framework of struc2vec behaves worse than ripple2vec in both node clustering and node classification (comparing the results of ripple2vec with ripple + struct). Both observations demonstrate the effectiveness of the design in Sect. 4.3.

5.3 Performance in Network Alignment

In addition to node classification, node cluster and link prediction, ripple vector can also be used in network alignment work. To show this, we use ripple vector to improve the GAlign [37](the improved algorithm denoted by RAlign).

Table 5 Performance of link prediction with logistic regression algorithm on node embeddings generated by different algorithms

Dataset	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
Brazil	acc	0.7854	0.7363	0.7518	0.7159	0.7618
	ROC	0.8696	0.8228	0.8292	0.7947	0.8378
	Metric	ripple -unlimited	ripple + struct	SDNE	struc2vec	
	acc	0.7851	0.7847	0.7798	0.7909	
USA	ROC	0.8694	0.8731	0.8537	0.8755	
	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
	acc	0.8298	0.8030	0.7775	0.6959	0.7926
	ROC	0.9121	0.8851	0.8626	0.7725	0.8828
Europe	Metric	ripple -unlimited	ripple + struct	SDNE	struc2vec	
	acc	0.8325	0.8358	0.8307	0.8358	
	ROC	0.9135	0.9160	0.9101	0.9159	
	Metric	ripple2vec	DeepWalk	LINE	node2vec	GCN
	acc	0.7894	0.7532	0.7339	0.7207	0.7343
	ROC	0.8682	0.8314	0.8125	0.7914	0.8133
	Metric	ripple -unlimited	ripple + struct	SDNE	struc2vec	
	acc	0.7911	0.7929	0.7882	0.7945	
	ROC	0.8664	0.8671	0.8630	0.8724	

Bold values mark the winners of the compared algorithms on each dataset and each metric

Table 6 Datasets used in network alignment

Networks	Nodes	Edges	Attributes
Allmovie	6011	124709	14
Imdb	5713	119073	14
Douban Online	3906	8164	538
Douban Offline	1118	1511	538
ppi-1	591	3104	50
ppi-2	591	3093	50
blogCatalog-1	10312	267522	39
blogCatalog-2	10312	267189	39

GAlign is a state-of-the-art method, which is an unsupervised network alignment method, but the accuracy of the network alignment is not worse than the supervised network alignment method. In the RAlign algorithm, we denote the alignment matrix obtained by the GAlign algorithm through node attribute training as S1 and then use the ripple vector of each node as an additional attribute of each node and use the GAlign algorithm to train the alignment matrix on the additional attributes, denoted as S2. The weighted summation of S1 and S2 is used to obtain the alignment matrix S, and finally, S is used for network alignment.

Table 7 Network alignment comparison

Dataset	Metric	RAlign	GAlign	IsoRank	FINAL	REGAL
Allmovie-Imdb	Acc_1	0.648	0.625	0.5522	0.6039	0.0954
	MAP	0.7158	0.6945	0.6141	0.7088	0.1909
	Acc_5	0.7938	0.7741	0.6742	0.8324	0.2771
	Acc_{10}	0.8386	0.8237	0.7302	0.8737	0.3932
Douban Online–Offline	Acc_1	0.4324	0.4344	0.1374	0.3486	0.048
	MAP	0.5438	0.5414	0.2526	0.4864	0.0999
	Acc_5	0.6728	0.6674	0.3832	0.657	0.1341
	Acc_{10}	0.7638	0.7620	0.5151	0.7765	0.2011
ppi	Acc_1	0.6349	0.5976	0.0778	0.1423	0.3776
	MAP	0.6920	0.6523	0.1456	0.2079	0.4505
	Acc_5	0.7533	0.7083	0.2296	0.2884	0.5237
	Acc_{10}	0.8010	0.7479	0.2865	0.3359	0.5844
blogCatalog	Acc_1	0.4809	0.4405	0.0305	0.0076	0.3224
	MAP	0.5296	0.4897	0.0578	0.023	0.378
	Acc_5	0.5795	0.5408	0.0737	0.0242	0.4349
	Acc_{10}	0.6177	0.5792	0.1122	0.0387	0.4794

Bold values mark the winners of the compared algorithms on each dataset and each metric

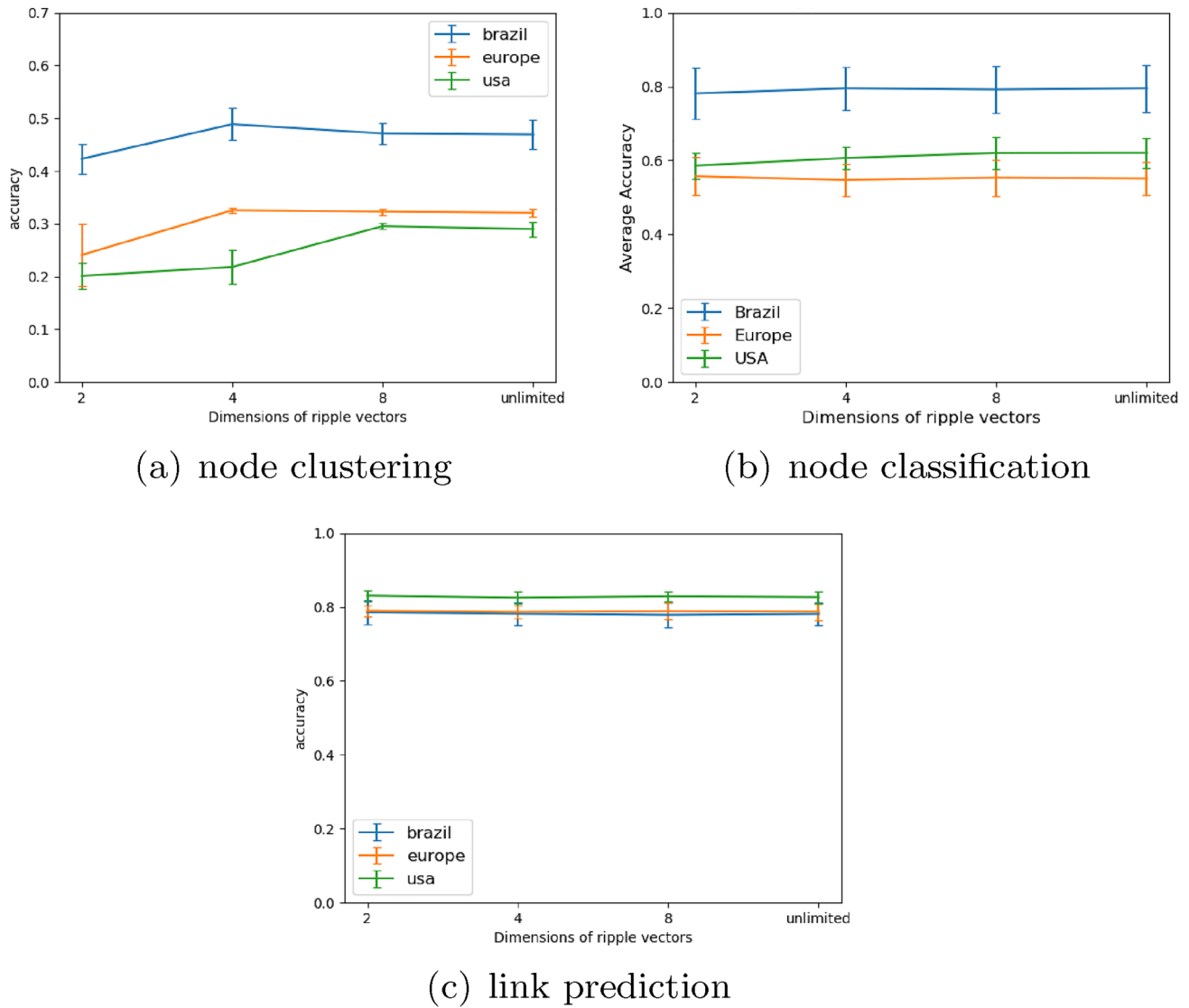


Fig. 7 Impacts of the dimension of the ripple vectors on the performance of ripple2vec

We use 4 real datasets: Allmovie-Imdb [37], Douban Online–Offline [41], ppi and blogCatalog. The Allmovie-Imdb dataset is a movie network dataset from the Rotten Tomatoes website and the Imdb website. The nodes represent movies, and the edges represent two movies with at least one co-actor. Douban Online–Offline dataset is a dataset from Chinese social networks. The nodes represent users, and the edges represent friendships. The ppi network is short for protein–protein interaction network. The nodes represent proteins, and the edges represent the interaction between two proteins. And, the blogCatalog dataset is a social relationship network, with nodes representing users and edges representing whether there is a social relationship (such as a friend relationship). Table 6 summarizes the statistics of these datasets.

On all datasets, we generate network alignment datasets for testing by randomly removing 20% of the edges and randomly modify the attributes of 15% of the nodes on a single real network to generate different subnetworks.

We compare RAlign with 4 algorithms, *i.e.*, GAlign [37], IsoRank [38], FINAL [39], REGAL [40]. Among them, the IsoRank and FINAL are supervised algorithms. For these two algorithms, we use 20% of the ground truth as the training data. The performance is evaluated as Acc_1 , Acc_5 , Acc_{10} and MAP [42], where Acc_q indicates whether the first q candidates have a correct matching and $MAP = mean(\frac{1}{ra})$, where ra is the rank of true anchor target in the sorted list of anchor candidates. The experimental results are reported in Table 7.

Table 7 tells us that (1) on the artificial synthetic dataset, the various metrics of network alignment using RAlign are much higher than other algorithms; (2) on two real datasets,

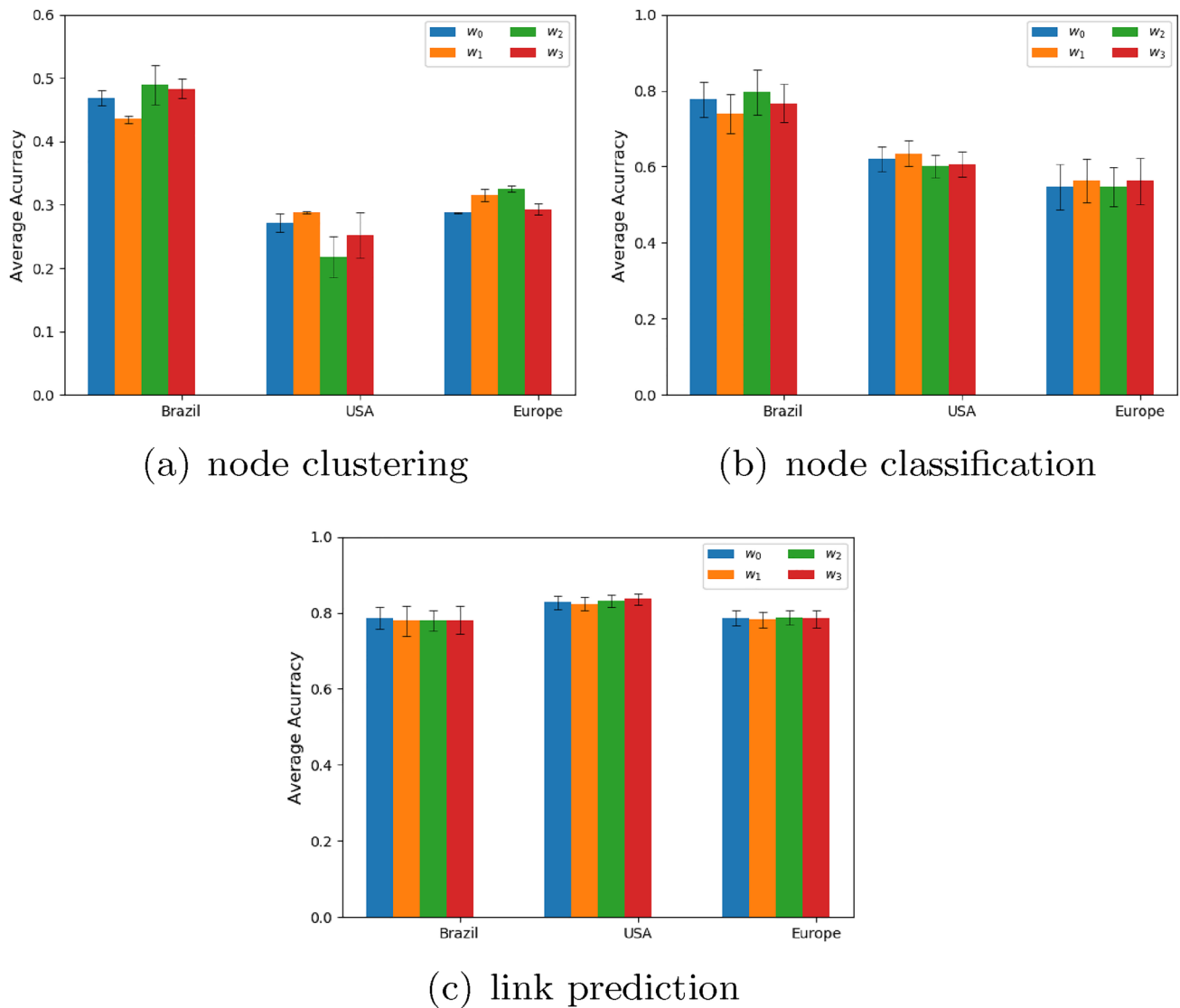


Fig. 8 Impacts of the weights of the ripple vectors on the performance of ripple2vec

the obtained test results also meet our expectations, RAlign only performs worse than other algorithms in three tests, and RAlign ranks second in these three tests. This proves that ripple vector can capture the local structure of the network well, and this ability is also effective in the field of network alignment.

5.4 Effects of the Dimension and the Weights

Next, we will show the effects of the dimension K and the weights to help you understand how to better adjust the parameters.

Effects of K on the performance of ripple2vec. On all datasets in Subsection 5.2 with the same experimental settings,

we change the dimension K of the ripple vectors and observe its impacts on the performance of ripple2vec in these analysis tasks.

Figure 7 reports the average accuracies and the standard deviations. We find that the average accuracies increase a little bit slowly when the dimension changes from 2 to 4 and keep stable when it becomes larger, which is consistent with the fact that the local structures near nodes are more useful than those far from nodes in usual graph analysis tasks.

Effects of weights on the performance of ripple2vec. With the same settings in Subsection 5.2, we use four different groups of weights in the ripple distance and observe their impacts on these classical analysis tasks.

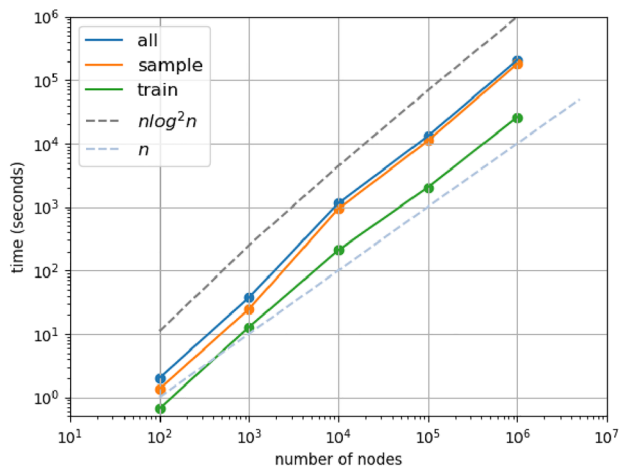


Fig. 9 Performance in node clustering, node classification, link prediction and scalability

- W0: $w_0 = 5, w_1 = 4, w_2 = 3, w_3 = 2, w_4 = 1$;
- W1: $w_0 = 1, w_1 = 2, w_2 = 3, w_3 = 4, w_4 = 5$;
- W2: $w_0 = 20, w_1 = 20, w_2 = 20, w_3 = 20, w_4 = 20$;
- W3: $w_0 = 1, w_1 = 1, w_2 = 1, w_3 = 1, w_4 = 1$;

Figure 8 reports the results. We find that (1) accuracies in former two tasks change more dramatically than in the last one. This is because the sigmoid activation function in the selected algorithm weakens the impacts of distances between embedded vectors. (2) The impacts of the weights are dataset-specific, since any group of weights does not always outperform others both in node clustering and node classification.

5.5 Scalability

To check the scalability of ripple2vec, we fix $K = 4$ and run it on instances of the Erdős–Rényi random graph model. We take the number of nodes as $10^2, 10^3, 10^4, 10^5$ and 10^6 , respectively, and other parameters as in Subsection 5.2. The results are reported in Fig. 9, which indicates that ripple2vec scales super-linearly under curve of $|V| \log^2 |V|$. This is consistent with our analysis.

6 Conclusion

This paper proposes to improve the node embedding by constructing a context graph with a new defined ripple distance over ripple vectors, whose components are the hitting times of the fully condensed neighborhoods and thus characterize the structures of the neighborhoods as pure quantities. The ripple distance satisfies the triangular inequality and is able

to capture the (dis)similarities of the local neighborhood structures. The neighbors of each node in the context graph are optimized to its nearest neighbors under the ripple distance, which guarantees that the short random walks from the node in the context graph only visit its similar nodes in the original graph, which makes ripple2vec map similar nodes to near vectors and dissimilar nodes to far vectors. Algorithms to compute the ripple vectors and the context graph are carefully designed, which makes ripple2vec scale well. As a future work, we will integrate structure similarity captured by ripple2vec with deep neural networks like GCN and graph kernels to improve the performance of downstream graph analysis tasks.

Author contributions Song Xiao completed the design and process of the research. Jizhou Luo provided research ideas and wrote the manuscript. Shouxu Jiang and Hong Gao helped perform the analysis with constructive discussions. Yinuo Xiao implemented the relevant baseline

Funding None.

Data availability All data generated or analyzed during this study are included in this published article.

Code availability The code and the datasets of this research this study have been deposited in: <https://github.com/hitSongXiao/ripple2vec>.

Declarations

Ethics approval and consent to participate Not applicable.

Consent for publication Not applicable.

Competing interests Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Li Enzhi, Le, Zhengyi (2020) Frustrated random walks: a faster algorithm to evaluate node distances on connected and undirected graphs. [arXiv:1908.09644v4](https://arxiv.org/abs/1908.09644)
2. Zhou Xuanhe, Sun Ji, Li Guoliang, Feng Jianhua (2020) Query performance prediction for concurrent queries using graph embedding. *PVLDB* 13(9):1416–1428

3. Cai H, Zheng VW, Chang KC (2017) A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Trans Knowledge Data Eng* 30(9):1616–1637
4. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: Online learning of social representations. In: *Proceedings of the 20th ACM SIGKDD International conference on Knowledge discovery and data mining*. ACM 2014, pp. 701–710
5. Tang J, Qu M, Wang M, et al. (2015) Line: Large-scale information network embedding. In: *Proceedings of the 24th international conference on world wide web*. WWW 2015, pp. 1067–1077
6. Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: *Proceedings of the 22th ACM SIGKDD International conference on Knowledge discovery and data mining*. SIGKDD 2016, pp. 855–864
7. Ribeiro LF, Saverese PH, Figueiredo DR (2017). struc2vec: Learning Node Representations from Structural Identity. In: *Proceedings of the 23th ACM SIGKDD international conference on Knowledge discovery and data mining*. SIGMOD 2017, pp. 385–394
8. Rossi RA, Ahmed NK, Koh E, Kim S, Rao A, Abbasi-Yadkori Y (2020) A structural graph representation learning framework. In: *Proceeding of the 13th ACM international conference on web search and data mining*. WSDM 2020, pp. 483–491
9. Wang D, Cui P, Zhu W (2016) Structural deep network embedding. In: *Proceedings of the 22th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM 2016, pp. 1225–1234
10. 2016Cao S, Lu W, Xu Q (2016) Deep neural networks for learning graph representations. In *Proceedings of the AAAI conference on artificial intelligence*
11. Narayanan A, Chandramohan M, et al. (2017) graph2vec: Learning distributed representations of graphs. [arXiv: Artificial Intelligence](https://arxiv.org/abs/1706.07845)
12. Goyal Palash, Hosseinmardi Homa, Ferrara Emilio, Galstyan Aram (2018) Capturing edge attributes via network embedding. *IEEE Trans Comput Soc Syst* 5(4):907–917
13. Ou M, Cui P, Pei J, Zhang Z, Zhu W (2016). Asymmetric transitivity preserving graph embedding. In: *Proceedings of the 22th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM 2016, pp. 1105–1114
14. Cao S, Lu W, Xu Q (2015). Grarep: Learning graph representations with global structural information. In: *Proceedings of the 24th ACM international conference on Information and Knowledge Management*. CIKM 2015, pp. 891–900
15. Park C, Yang C, Zhu Q, Kim D, Yu H, Han J (2020) Unsupervised differentiable multi-aspect network embedding. In: *Proceeding of the 26th ACM SIGKDD conference on knowledge discovery and data mining*. KDD 2020, pp. 1435–1445
16. Chen H, Perozzi B, Hu Y, Skiena S (2017) Harp: Hierarchical representation learning for networks. [arXiv preprint arXiv:1706.07845](https://arxiv.org/abs/1706.07845)
17. Chamberlain BP, Clough J, Deisenroth MP (2017) Neural embeddings of graphs in hyperbolic space. [arXiv preprint arXiv:1705.10359](https://arxiv.org/abs/1705.10359)
18. Wang Q, Mao Z, Wang B, Guo L (2017) Knowledge graph embedding: a survey of approaches and applications. *IEEE Trans Knowledge Data Eng* 29(12):2724–2743
19. Kriege Nils M, Johansson Fredrik D, Morris Christopher (2020) A survey on graph kernels. *Appl Netw Sci* 5(1):1–42
20. Wang Yanhao, Yuchen Li Ju, Fan Chang Ye, Chai Mingke (2020) A survey of typical attributed graph queries. *World Wide Web* 2020:1–50
21. Wu S, Zhang W, Sun F, Cui B (2020) Graph neural networks in recommender systems: a survey. *information retrieval*. [ArXiv Preprint. ArXiv:2011.02260](https://arxiv.org/abs/2011.02260)
22. Fang Y, Huang X, Qin L, Zhang Y et al (2020) A survey of community search over big graphs. *Vldb J* 29(1):353–92. <https://doi.org/10.1007/s00778-019-00556-x>
23. Yu Y, Lu Z, Liu J, Zhao G, Wen J (2019). RUM: Network representation learning using motifs. In: *Proceeding of IEEE 35th international conference on data engineering*. ICDE 2019, pp. 1382–1393
24. Shen-Orr Shai S, Milo Ron, Mangan Shmoolik, Alon Uri (2002) Network motifs in the transcriptional regulation network of *escherichia coli*. *Nat Genet* 31(1):64–68
25. Shin Kijung Oh, Sejoon Kim Jisu, Bryan Hooi, Christos Faloutsos (2020) Fast, accurate and provable triangle counting in fully dynamic graph streams. *ACM Trans Knowledge Discov Data* 14(2):1–39
26. Grohe M (2020). word2vec, node2vec, graph2vec, X2vec: Towards a theory of vector embeddings of structured data. In: *Proceeding of the 39th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*. PODS 2020, pp. 1–16
27. Linial N, London E, Rabinovich Y (1995) The geometry of graphs and some of its algorithmic applications. *Combinatorica* 15(1995):212–245
28. Mikolov, T, Chen K, Corrado G, Dean J (2013). Efficient estimation of word representations in vector space. In: *Proceeding of the 1st international conference on learning representations*. ICLR 2013. [arXiv:1301.3781v3](https://arxiv.org/abs/1301.3781v3)
29. Ohsaka N (2020) The solution distribution of influence maximization: a high-level experimental study on three algorithmic approaches. In: *Proceeding of the 2020 international conference on management of data*. SIGMOD 2020, pp. 2183–2197
30. Rong Y, Huang W, Xu T, Huang J (2020). DropEdge: Towards deep graph convolutional networks on node classification. In: *Proceeding of the 8th international conference on learning representations*. ICLR 2020
31. Belkin M, Niyogi P (2003) Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Comput* 15(6):1373–1396
32. Tenenbaum J, De Silva V, Langford J (2000) A global geometric framework for nonlinear dimensionality reduction. *Science* 290(2000):2319–2323
33. Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29(1):1–27
34. Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500):2323
35. Fagin R, Lotem A, Naor M (2001) Optimal aggregation algorithm for middleware. In: *Proceeding of the 20th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*. PODS 2001
36. Kipf Thomas N, Welling M: (2017) Semi-supervised classification with graph convolutional networks. In: *Proceeding of the 5th international conference on learning representation*, ICLR 2017
37. Trung HT, Tong VV, Nguyen T, Yin H, Hung N (2020) Adaptive network alignment with unsupervised and multi-order convolutional networks. In: *2020 IEEE 36th international conference on data engineering (icde)*, IEEE, 2020
38. Singh R, Xu J, Berger B. (2008) Global alignment of multiple protein interaction networks with application to functional orthology detection. In: *Proceedings of the National Academy of Sciences of the United States of America*. 105.35(2008):12763–12768
39. Zhang Si, Tong H. (2016) FINAL: Fast attributed network alignment. In: *Acm Sigkdd international conference*, ACM 2016
40. Heimann M et al (2018) REGAL: Representation learning-based graph alignment. *ACM* 2018
41. Zhong E, Fan W, Wang J, Xiao L, Li Y (2012). Comsoc: adaptive transfer of user behaviors over composite social network. *ACM* 2012
42. Man T, Shen H, Liu S et al (2016) Predict anchor links across social networks via an embedding approach. *AAAI Press*, 2016