Journal of Database Management
Volume 34 • Issue 1

A Truss-Based Framework for Graph Similarity Computation

Yanwei Zheng, Shandong University, China
Zichun Zhang, Shandong University, China
Qi Luo, Shandong University, China*
Zhenzhen Xie, Shandong University, China
Dongxiao Yu, Shandong University, China

ABSTRACT

The study of graph kernels has been an important area of graph analysis, which is widely used to solve the similarity problems between graphs. Most of the existing graph kernels consider either local or global properties of the graph, and there are few studies on multiscale graph kernels. In this article, the authors propose a framework for graph kernels based on truss decomposition, which allows multiple graph kernels and even any graph comparison algorithms to compare graphs at different scales. The authors utilize this framework to derive variants of five graph kernels and compare them with the corresponding basic graph kernels on graph classification tasks. Experiments on a large number of benchmark datasets demonstrate the effectiveness and efficiency of the proposed framework.

KEYWORDS

C-SVM, Cohesive Subgraph, Graph Analysis, Graph classification, Graph Kernels, Graph mining, Hierarchical Decomposition, Multiscale Graph Kernels

INTRODUCTION

Graph, a form of structured data, allows for the modeling of complex relationships among objects. In recent years, more and more data are modeled as graphs and such data is ubiquitous in social networks, biology, chemistry, computer vision, and other domains (Cai et al., 2018, 2021; L. Li et al., 2019; Yanardag & Vishwanathan, 2015b). For example, graphs can be derived from interactions between groups of people, such as friendships on a social website, or collaborations in a network of actors or scientists. In biochemistry, molecular compounds can be modeled as graphs with vertices corresponding to atoms and the edges to chemical bonds (Vishwanathan et al., 2010).

The problem of accurately measuring the similarity between graphs is at the core of many applications(Peng et al., 2022). As the demand for structured data analysis grows, the problem of graph

DOI: 10.4018/JDM.322087 *Corresponding Author

classification has been widely studied in these domains. For example, in protein function prediction, it is a common task to determine whether a protein is an enzyme or a non-enzyme (Borgwardt & Kriegel, 2005). A similar task on a collaboration dataset is to predict which genre an ego-network of an actor/actress belongs to (Yanardag & Vishwanathan, 2015b). Since there are many ways to represent images as graphs, many computers vision tasks, such as classifying images(Tian & Han, 2022), can also be considered as graph classification problems.

To date, the kernel methods are one of the most popular methods for comparing similarities between structured objects, especially in the domain of graph classification. Graph kernels have achieved state-of-the-art results on many graph datasets. In general, the kernel methods utilize a positive semidefinite kernel function to measure the similarity between two objects, which can be expressed as an inner product in reproducing kernel Hilbert space $\mathcal{H}$ (Schölkopf & Smola, 2001). Some graph kernels are computed based on local properties, such as the neighborhood features of vertices (Hido & Kashima, 2009), or some small substructures of graphs (i.e., trees, cycles, graphlets) (Horváth et al., 2004; Orsini et al., 2015; Shervashidze et al., 2009, 2011). There are also some graph kernels built from a global perspective and they focus on the global properties of graphs (Kang et al., 2012; Nikolentzos et al., 2017; Wu et al., 2019). Most of the existing graph kernels cannot capture graph similarity at multiple levels of scale. The Multiscale Laplacian Graph Kernel (Kondor & Pan, 2016) is the first graph kernel that can compare graphs at multiple different scales, by building a hierarchy of nested subgraphs. However, it requires Laplacian matrix operations on the graph and thus has a very high runtime.

In the real world, many graphs have different structures at different scales. For example, a molecule has a small structure, such as a specific chemical bond, and an overall structure, such as a chain or a ring. By extracting the structure of the graph at different scales, it is possible to reflect the relationship of local structures relative to the global structure. Therefore, it would be desirable to find an efficient way to reveal the structure of a graph at different scales. Considering that most real-world graphs are usually sparse overall and dense in parts, the dense regions may reflect a higher frequency of interactions between these vertices and greater similarity to each other (Lotito & Montresor, 2020). Hence, the density information of the graph can be used to reveal the graph structure. In the literature, many concepts of cohesive subgraphs have been proposed to explore dense regions. The k -truss is a popular type of cohesive subgraph that has been studied in recent years. It has been applied to community search (Katunka et al., 2017; Yu et al., 2020), complex network visualization (Zhao & Tung, 2012), and other applications. The k -truss of G is the largest subgraph of G in which every edge is contained in at least $k-2$ triangles within the subgraph (J. D. Cohen, 2008). It is defined based on the triangle, which denotes the existence of stable relations among three nodes. k reflects the density information of the graph. The k -truss decomposition procedure is to find the k -trusses for all possible k on the graph, which leads to hierarchical subgraphs that represent the dense regions of a graph at different scales. In this way, it is possible to reveal graph structure at different scales. To the best of the authors' knowledge, graph kernel combined with k -truss has not yet been studied in the literature.

In this article, a new framework is proposed for graph similarity. The authors take advantage of the k -truss decomposition to obtain hierarchical structures of G at different scales, then add up the similarity between the corresponding subgraphs according to the hierarchy. The combined result provides a more accurate representation of graph similarity. More specifically, the contributions in this article are as follows:

- The authors first propose a framework for comparing graphs at multiple different scales utilizing k -truss decomposition of graphs to build a hierarchy of nested subgraphs. The framework can be applied to all methods used for graph comparison.

- The framework generates valid truss variants of several base graph kernels, which enhances the expressivity of base graph kernels for complex substructures. In particular, the framework improves the performance of base graph kernels on graph classification tasks.
- The authors evaluate the framework on a set of benchmark datasets. Moreover, the truss variants perform well and achieve significant improvements over the base kernels in most cases.

The rest of this article is organized as follows. In Section 2, the authors review some related work in the literature. In Section 3, the authors review some preliminary concepts and give details about truss degeneracy and k-truss decomposition. In Section 4, the authors introduce the proposed framework for graph similarity. In section 5, the authors implement and evaluate the proposed framework on several benchmark datasets. Section 6 concludes the article.

BACKGROUND

In this section, the related work is presented in terms of both graph kernel studies and k-truss mining.

Graph Kernels

One of the most popular frameworks to construct kernels is R-convolution (Haussler, 1999) where the key idea is to decompose graphs into substructures and add up the pairwise similarities between them. Specific subgraphs focus on different structural aspects of graphs, such as random walks (Kang et al., 2012; Kashima, 2003), graphlets (Shervashidze et al., 2009), cycles(Horváth et al., 2004), paths (Borgwardt & Kriegel, 2005), and subtrees (B. Li et al., 2012; Morris et al., 2017). Besides the family of R-convolution, assignment kernels have received a lot of attention recently. This framework maximizes the similarity of the two graphs by calculating the matching between their substructures. (Nikolentzos et al., 2017) applies pyramid match kernel to match the node embeddings of graphs. There are also some frameworks that work on top of graph kernels and aim to improve the performance of base kernels. The deep graph kernels framework (Yanardag & Vishwanathan, 2015b) and the structural smoothing framework (Yanardag & Vishwanathan, 2015a) are inspired by natural language processing. R-convolution suffers from diagonal dominance, which means that there are few substructures common to all, causing each graph to be similar only to itself and not to other graphs. These two frameworks were developed to address the diagonal dominance problem. The core framework (Nikolentzos et al., 2018) is another example of improving the performance of graph kernels which takes into account structure at multiple different scales. The k-core of G is the largest subgraph of G in which every vertex has at least k neighbors within the subgraph (Jin et al., 2018; Luo, Yu, Cai, et al., 2021; Luo, Yu, Zheng, et al., 2021). It utilizes the well-known k-core decomposition of graphs to build a hierarchy of nested subgraphs (Hua et al., 2020; Luo et al., 2019; N. Wang et al., 2017; Zhou et al., 2021). It is also applicable to existing graph kernels, as well as any graph comparison algorithm.

K-Truss Mining

The k-truss (J. D. Cohen, 2008) is the largest subgraph of a graph in which every edge is contained in at least $(k-2)$ triangles within the subgraph. The k-truss decomposition (J. Wang & Cheng, 2012) is to find the k-trusses for all possible k values in the graph. In recent works, (J. D. Cohen, 2008) proposed a base algorithm for truss decomposition, which is a typical in-memory truss decomposition algorithm. (J. Cohen, 2009) used MapReduce framework to perform truss decomposition in parallel, but the proposed method is relatively inefficient when dealing with large graphs. (Chen et al., 2014) first improved the existing distributed k-truss decomposition in the MapReduce framework and designed an algorithm based on graph-parallel abstraction. (J. Wang & Cheng, 2012) modified the algorithm proposed by (J. D. Cohen, 2008) and proposed an efficient in-memory algorithm and two

I/O-efficient algorithms for truss decomposition in massive networks. Furthermore, many researchers have studied k-truss on multiple types of graphs, including directed graphs (Liu et al., 2020), probabilistic graphs (Huang et al., 2016), and public-private networks (Ebadian & Huang, 2019). For truss maintenance, (Huang et al., 2014) proposed a simple truss maintenance algorithm on dynamic graphs with single-edge operation. (Akbas & Zhao, 2017) proposed an index structure EquiTruss to accelerate community search which can be efficiently updated dynamically. (Luo et al., 2020) is the first work to study the batch processing of truss maintenance in large graphs and their solution is innovative with multi-edge insertions/deletions.

DETAILS OF THE GRAPH KERNELS AND K-TRUSS

Graph Kernel Theory

In this section, the authors first describe kernel function and kernel matrix, and then an overview of several graph kernels is provided.

Let $k : G \times G \rightarrow \mathcal{R}$ be a kernel function for all $G, G' \in \mathcal{G}$, there exists a map $\phi: \mathcal{G} \rightarrow \mathcal{H}$ into a Hilbert space $\mathcal{H}$ such that

$$\boldsymbol{k}\left(\boldsymbol{G}, \boldsymbol{G'}\right) = \left\langle \phi\left(\boldsymbol{G}\right), \phi\left(\boldsymbol{G'}\right) \right\rangle, \tag{1}$$

where the operation between $\phi\left(G\right)$ and $\phi\left(G'\right)$ is the Euclidean dot product in $\mathcal{H}$. Given a set of graphs $\left\{G_1, \ldots, G_n\right\}$ and a kernel function k, the element K_{ij} of a kernel matrix K is calculated as

$$\boldsymbol{K_{ij}} = \boldsymbol{K_{ji}} = \boldsymbol{k}\left(\boldsymbol{G_i}, \boldsymbol{G_j}\right), \tag{2}$$

for $1 \leq i, j \leq n$. The kernel matrix must be positive semidefinite (i.e. PSD), as it is a necessary condition for solving some convex optimization problems (including SVM) in kernel-based methods. One way to check if a kernel matrix is PSD is to see if the eigenvalues of this matrix are all non-negative. Furthermore, it is noted that kernel functions are additive, that is, a linear sum of two valid kernels is also valid. This can be formalized as:

$$\boldsymbol{k''}\left(\cdot,\cdot\right) = \boldsymbol{ck}\left(\cdot,\cdot\right) + \left(1 - \boldsymbol{c}\right)\boldsymbol{k'}\left(\cdot,\cdot\right), \tag{3}$$

for $0 < c < 1$. A sophisticated kernel can be defined by a linear combination of valid kernels. Good graph kernels are desired to be both effective and efficient. It should be highly expressive of the graph structure and has low computational complexity in both time and space.

The authors have selected several graph kernels for their experiments, which are described below.

- **Graphlet Kernel (GR)** (Shervashidze et al., 2009). A graphlet is a non-isomorph subgraph with k nodes $\left(k \in \left\{3, 4, 5\right\}\right)$. Let $\Gamma = \left\{g_1, g_2, \ldots, g_k\right\}$ be the set of graphlets of size k, $f_G \in R^d$ be a vector whose i-th element represents the number of g_i divided by the total number of graphlets in G. Formally, the graph kernel is defined as:

$$k_{GR}\left(G,G'\right)=\left\langle f_G,f_{G'}\right\rangle, \tag{4}$$

where the operation between f_G and $f_{G'}$ is the Euclidean dot product.

- **Shortest Path Kernel (SP)** (Borgwardt & Kriegel, 2005). The first step of the kernel is to transform the original graphs into shortest paths ones using the shortest path algorithm (i.e. Floyd's algorithm). Given G, the shortest paths graph S has the same vertex set as G. If vertex u and v have a walk in G, there is an edge $\left(u,v\right)$ in S. Then the edge is labeled by the length of the walks. Let $\Gamma=\{p_1,p_2,\ldots,p_k\}$ be the set of all shortest paths in G, with each one denoted as a triplet of the labels of the endpoints and the length of the path. Formally, the shortest path kernel is defined as:

$$k_{SP}\left(G,G'\right)=\left\langle p_G,p_{G'}\right\rangle, \tag{5}$$

where p_G be a vector whose i-th element reflects the occurring frequency of p_i in G.

- **Weisfeiler-Lehman Subtree Kernel (WL)** (Shervashidze et al., 2011). The kernel is based on Weisfeiler-Lehman test of graph isomorphism. Let G, G' be two graphs, $\Sigma\left[i\right]$ be the set of labels that appear in these two graphs at the end of the i-th iteration of the WL algorithm. Especially, $\Sigma\left[0\right]$ is the original label set. In each iteration, the labels of the node and its neighboring nodes merge into a multiset label, which is given a new label. Then the Weisfeiler-Lehman subtree kernel is defined as:

$$k_{WL}\left(G,G'\right)=\left\langle \phi\left(G\right),\phi\left(G'\right)\right\rangle, \tag{6}$$

where $\phi\left(G\right)$ is a vector of $h+1$ blocks, denoted as block 0 to block *h*, and the i-th element in j-th block reflects the occurring frequency of the corresponding label in the j-th iteration.

- **Neighborhood Hash Kernel** (Hido & Kashima, 2009). For labeled graphs, the kernel replaces the discrete node labels with a unique binary representation, then uses logical operations (XOR and ROT) to compute the neighborhood hash of a vertex. Furthermore, the Count-Sensitive neighborhood hash takes into account the number of neighbors with the same label using additional logical operations. Then the kernel is defined as:

$$k_{NH}\left(G,G'\right)=\frac{c}{\left|V\right|+\left|V'\right|-c}, \tag{7}$$

where c is the number of common labels in two graphs.

- **Pyramid Match Graph Kernel (PM)** (Nikolentzos et al., 2017). Firstly, the pyramid match graph kernel embeds the vertices of the input graphs in a vector space and partitions the feature space into regions of increasingly larger size. Given level $l \in \{0,...,L\}$, the d-dimensional unit hypercube has $C = 2^l d$ cells with 2^l cells in one dimension. H_G^l is a histogram of the number of vertices per cell at level l. And then a histogram intersection function I is used to count the matching points in two graphs at level l. ΔI means the new matches found at each level, which are weighted as $w_l = 1/2^{L-l}$. Then the kernel is defined as:

$$k_{PM}(G, G') = I(H_G^L, H_{G'}^L) + \sum_{l=0}^{L-1}(w_l \text{"} I). \tag{8}$$

Truss Degeneracy and Truss Decomposition

In this part, the authors first give an introduction of k-truss and truss degeneracy, then give details about the truss decomposition algorithm. Table 1 summarizes the symbols used in this article.

A k-truss is a connected subgraph of a graph in which every edge is contained in at least $(k-2)$ triangles (J. D. Cohen, 2008). A triangle in G is a cycle of length 3. The triangle with three vertices u, v, w is denoted by $\triangle_{uvw}$. The set of triangles of G is denoted as $\triangle_G$. The support of an edge $e=(u,v)$ in G, denoted by $sup(e,G)$, is defined as $\left|\{\triangle_{uvw} : \triangle_{uvw} \in \triangle_G\}\right|$, as well as $\left|N(u) \cap N(v)\right|$. When the context is clear, $sup(e,G)$ can be replaced by $sup(e)$. The support of e is the number of triangles in which e is involved in G. The k-truss of G, is defined as the largest subgraph H of G such that $\forall e \in E(H), sup(e,H) \geq (k-2)$. G itself is at least a 2-truss. In other words, the condition $k \geq 2$ always holds for any graph. The truss number, or trussness of e in G, is denoted by $t(e)$, as $max\{k : e \in E(T_k)\}$. $t(e) = k$ means e exists in T_k, but not in T_{k+1}. So T_k is the highest-order truss to which e belongs. It is clear that $t(e) = sup(e, T_k) + 2$. Given a graph G, the authors define the maximum trussness $t(e,G)$ of all edges as the truss degeneracy, $\omega^*(G) = max_{e \in E} t(e)$. In Figure 1, it is observed that the truss degeneracy of this graph is $\omega^*(G) = 5$. The 5-truss T_5 contains 5 vertices $\{a,b,c,d,e\}$. The 4-truss T_4 contains 4 vertices $\{g,h,k,l\}$. The edges (c,j) and (k,j) have the trussness of 2. Given a graph G, the problem of truss decomposition is to compute the k-truss of G for all $2 \leq k \leq \omega^*(G)$. For a given value of k, finding k-truss is equivalent to obtaining a subgraph induced by all edges with trussness at least k, or with support at least $k-2$. Truss decomposition aims to obtain the trussness of each edge. Furthermore, assuming that $T = \{T_2, ..., T_{\omega^*(G)}\}$ is the result of the truss decomposition, the T forms a nested chain:

$$T_{\acute{E}^*(G)} \subseteq ... \subseteq T_3 \subseteq T_2 = G. \tag{9}$$

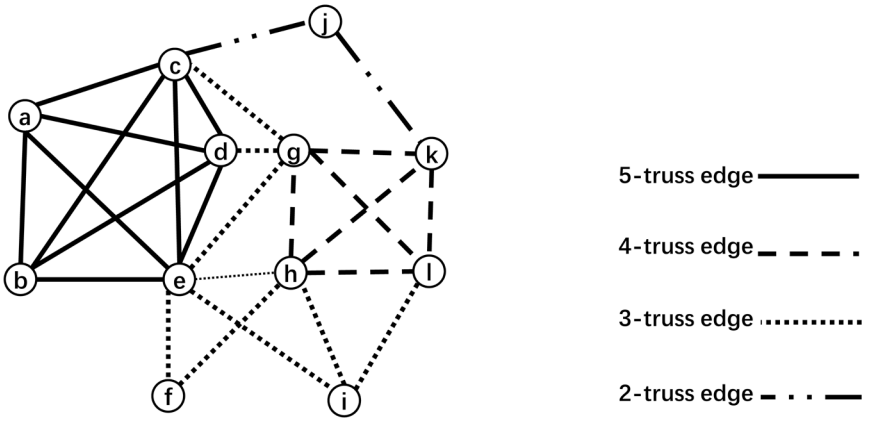
The nested nature above makes truss decomposition a very effective tool for discovering graph hierarchies. An edge belonging to the $(k+1)$-truss obviously belongs to the k-truss. Since higher-order trusses are nested within lower-order trusses, an intuitive idea is that the trusses can be calculated in the reverse order of the nested chain above.

Table 1.
Notations and descriptions

Notations	Descriptions
$\mathcal{G}$	The space of graphs
$\mathcal{H}$	A Hilbert space
$G\left(V,E\right)$	An undirected and unweighted simple graph G
$e=\left(u,v\right)$	An edge e with endpoints u and v
$n=\left\|V\right\|$	The number of vertices
$m=\left\|E\right\|$	The number of edges
H	A subgraph of G
$E\left(H\right)$	The set of edges of H
$\triangle_{uvw}$	A triangle formed by the vertices of u, v, w
$N\left(u,G\right)$	The neighbors of u in G
$deg\left(u,G\right)$	The number of neighbors of u in G
$sup\left(e,G\right)$	The support of e *in* G
$T_k\left(G\right)$	The k-truss of G
$t\left(e,G\right)$	The trussness of e in G
$\omega^*\left(G\right)$	The maximum trussness $t\left(e,G\right)$ of all edges in G
$\omega^*_{min}\left(G\right)$	The minimum $\omega^*\left(G\right)$ in a set of graphs
$\omega^*_{avg}\left(G\right)$	The average $\omega^*\left(G\right)$ in a set of graphs

The traditional batch *k*-truss decomposition algorithm is proposed in (J. D. Cohen, 2008). The algorithm recursively removes all edges with support less than $k-2$ triangles until there are no more edges to remove, and finally obtains the k-truss. Since the time complexity to search for $\triangle_{uvw}$ of this algorithm is high, (J. Wang & Cheng, 2012) proposes an improvement that optimizes the time complexity to $O\left(m^{1.5}\right)$, as shown in Algorithm 1. Both algorithms require $O\left(m+n\right)$ memory space. In Algorithm 1, initially, the algorithm uses the in-memory triangle counting algorithm (Latapy,

Figure 1.
The graph showing edges' trussness



2008) to compute the support of all edges in G. Then the edges in G are sorted in order of their supports using bin sort. For each trussness $k \geq 3$ ($k=2$ is omitted for $T_2 = G$), the edges with support less than $k-2$ are deleted from G, causing the subtraction of 1 from the support of the other two edges of the triangles containing them. The edges whose support values are updated should be reordered. When there are no edges to delete, output G as T_k.

Jdm.322087.g01

For some base graph kernels, the nested nature of the truss can be used to avoid some calculations. For example, as for the Weisfeiler-Lehman subtree kernel, given two trusses of a graph T_i and T_j with $i < j$, all te subtrees found in T_j will also be present in T_i. Some calculations can be performed only once, instead of once at each order of truss.

TRUSS FRAMEWORK

In this section, the authors propose a new framework based on k-truss for graph similarity that works on top of graph kernels, and then show how existing graph kernels can be combined with the framework for performance improvement.

Truss-based Graph Kernels

The new framework utilizes truss decomposition to generate the truss variants of the existing base graph kernels. The framework compares graph structures at different scales represented by the graphs' k-trusses. Theoretically, the internal truss implies a stronger connection between the edges. Hence, it is hoped that by decomposing graphs into progressively denser subgraphs, it is easier to get their potential structure and compare graphs more efficiently. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs and let k be any kernel for graphs. The truss variant kernel k_t of the base kernel k is defined as:

$$\boldsymbol{k_t}\left(\boldsymbol{G}, \boldsymbol{G'}\right) = \boldsymbol{k}\left(\boldsymbol{T}_2, \boldsymbol{T}_2^{'}\right) + \boldsymbol{k}\left(\boldsymbol{T}_3, \boldsymbol{T}_3^{'}\right) + \ldots + \boldsymbol{k}\left(\boldsymbol{T}_{\boldsymbol{E}^*_{min}}, \boldsymbol{T}^{'}_{\boldsymbol{E}^*_{min}}\right), \tag{10}$$

where ω^*_{min} means the minimum of the truss degeneracies of the two graphs, $\left\{T_2, T_3, ..., T_{\omega^*_{min}}\right\}$ and $\left\{T_2^{'}, T_3^{'}, ..., T_{\omega^*_{min}}^{'}\right\}$ mean the 2-truss, 3-truss,..., ω^*_{min}-truss of G and G' respectively. This procedure can be seen as running a specific base kernel function on the corresponding trusses of the two graphs separately and adding up the results of each comparison. Next, the authors prove the positive semidefinite nature of the proposed kernel.

Statement of Validity of the Truss Variants

Let the base kernel k be any positive semidefinite kernel on graphs, then the truss variant kernel k_t of the base kernel k is positive semidefinite. Let ϕ be the feature mapping corresponding to the kernel k:

$$\boldsymbol{k}\left(\boldsymbol{G}, \boldsymbol{G'}\right) = \phi\left(\boldsymbol{G}\right), \phi\left(\boldsymbol{G'}\right). \tag{11}$$

Let $t_i\left(\cdot\right)$ be a function that removes all edges with trussness less than i from ***G***:

$$\boldsymbol{k}\left(\boldsymbol{T_i}, \boldsymbol{T_i'}\right) = \left\langle \phi\left(\boldsymbol{t_i}\left(\boldsymbol{G}\right)\right), \phi\left(\boldsymbol{t_i}\left(\boldsymbol{G'}\right)\right)\right\rangle. \tag{12}$$

If the feature mapping $\phi\left(t_i\left(\cdot\right)\right)$ is defined as $\psi\left(\cdot\right)$, then Equation 12 can be expressed as:

$$\boldsymbol{k}\left(\boldsymbol{T_i}, \boldsymbol{T_i'}\right) = \left\langle \grave{\boldsymbol{E}}\left(\boldsymbol{G}\right), \grave{\boldsymbol{E}}\left(\boldsymbol{G'}\right)\right\rangle. \tag{13}$$

Hence k is a kernel on T_i and $T_i^{'}$. According to Equation 3, as the sum of a set of positive semidefinite kernels, the proposed kernel k_t is also a positive semidefinite kernel.

As stated above, the authors have proposed a framework that can improve the performance of existing graph kernels. It can be combined with any algorithm for graph comparison which is not limited to just the graph kernel. Given two graphs G and G' and a base kernel k, Algorithm 2 gives the procedure for calculating the truss variant kernel k_t.

JDM.322087.g02

Time Complexity Analysis

A good graph kernel not only requires good graph similarity representation capabilities and strong applicability, but it must also be competitive in terms of time complexity. The truss decomposition itself can be done within polynomial time, so the proposed graph kernel is very promising. Given a pair of graphs G and G', and a kernel function k with the time complexity of O_k. The graph kernel function here can be any algorithm used for graph comparison. Let $\omega^*_{min} = \min\{\omega^*\left(G\right), \omega^*\left(G'\right)\}$ be the minimum of the truss degeneracies of G and G'. Then the time complexity of the truss variant k_t is $O_{k_t} = \omega^*_{min} O_k$. In the real world, it always holds that $\omega^*\left(G\right) \ll n$ for most graphs. Therefore, the proposed framework adds a relatively small-time complexity to the base graph kernel.

EXPERIMENTS AND EVALUATION

In this section, the authors first introduce the datasets used in the experiments and then give the details of the experiments. Finally, a performance comparison between the basic kernels and truss variants is given.

Datasets

In order to evaluate the performance of the framework, the authors applied the framework to benchmark datasets which include chemoinformatics datasets (MUTAG, PTC-MR, NCI1, and NCI109), bioinformatics datasets (ENZYMES and PROTEINS), social networks datasets (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, and REDDIT-MULTI-5K), and synthetic datasets (SYNTHETICnew and Synthie). Note that the chemoinformatics and bioinformatics datasets are labeled, while all other graph datasets are unlabeled. Table 2 summarizes the properties of the datasets used in the experiments.

Experimental Setup

To evaluate both the effectiveness and the efficiency of the proposed graph kernels, the authors made use of the GraKeL library (Siglidis et al., 2020) which contains implementations of 15 kernels and 2 kernel frameworks. All kernels are implemented in Python. Several base graph kernels are selected from current main families of graph kernels in the literature, namely graphlet kernel (GR), Weisfeiler-Lehman subtree kernel (WL), shortest-path kernel (SP), pyramid match graph kernel (PM), and neighborhood hash kernel (NH). The authors employed C-Support Vector Machine (SVM) classifier and performed 10-fold cross-validation. The parameter C of the SVM and the hyperparameters of the kernels were optimized on a validation experiment on the training set. All kernel matrices were normalized in the experiments. The value of C was chosen from $\left\{10^{-7},10^{-5},...,10^{5},10^{7}\right\}$. The parameters of the graph kernels were chosen as follows. For the graphlet kernel, the size of graphlets was chosen from $\left\{3,4,5\right\}$, and the number of sampling graphlets was chosen from $\left\{150,200,500\right\}$.

Table 2.
Properties of the datasets used in graph kernel experiments. The "M.C.I" stands for max class imbalance, which refers to the ratio of the size of the smallest class to the largest class in the dataset

Datasets	Size	Classes	M.C.I	Avg. Nodes	Avg. Edges	Node Labels	Avg. Deg	Avg. △
MUTAG	188	2	1:1.98	17.93	19.79	7	2.19	0
PTC-MR	344	2	1:1.26	14.29	14.69	19	1.98	0.04
ENZYMES	600	6	1:1	32.63	62.14	3	3.87	25.51
PROTEINS	1113	2	1:1.47	39.06	72.82	3	3.73	27.4
NCI1	4110	2	1:1.49	29.87	32.3	37	2.16	0.05
NCI109	4127	2	1:1.48	29.68	32.13	38	2.17	0.04
IMDB-BINARY	1000	2	1:1	19.77	96.53	-	8.89	391.99
IMDB-MULTI	1500	3	1:1	13	65.94	-	8.1	305.9
REDDIT-BINARY	2000	2	1:1	429.63	497.75	-	2.34	24.84
REDDIT-MULTI-5K	5000	5	1:1	508.52	594.87	-	2.25	21.77
SYNTHETICnew	300	2	1:1	100	196.25	-	3.93	4.96
Synthie	400	4	1:1.22	95	172.93	-	3.75	18.81

For the Weisfeiler-Lehman subtree kernel, the number of iterations h was chosen from $\{1,2,3,4,5,6\}$. Similarly, for the neighborhood hash kernel, the neighborhood range R was chosen from $\{1,2,3,4,5,6\}$. The shortest-path kernel lacks hyperparameters, while for the pyramid match graph kernel, the embedding dimension d was chosen from $\{4,6,8,10\}$ and level number L was chosen from $\{2,4,6\}$. To summarize, the parameters of the graph kernel are set as follows:

The authors report average accuracies and standard deviations in Table 4. Truss variants with improvements to the basic kernels are shown in bold. Furthermore, to study the runtime, the authors examined the computation time of the kernel matrices and give the comparison of the running time of base kernels vs their truss variants. For each fold of the 10-fold cross-validation, the runtime of the kernel is considered to be the runtime of the kernel matrix that performed best on the validation experiments.

Results

To begin with, base kernels are compared with their truss variants on performing graph classification. Table 4 demonstrates that the proposed framework improves the classification accuracy on almost all datasets. It is noted that the truss variants outperform their base kernels on 43 out of the 60 experiments. On the PTC-MR and REDDIT-MULTI-5K datasets, all the truss variants improve the classification accuracy achieved by the basic kernels. It is clear that on most datasets, the performance of the framework applied to GR and PM is greatly improved. Truss PM improves the accuracy attained by the PM kernel on all datasets. Specifically, truss GR improves the accuracy on 5 datasets by more than 9%, and it improves accuracy on symmetric datasets by more than 12%. Conversely, truss WL improves accuracy by a very small amount on most datasets. One speculation is that WL graph kernel only takes local graph properties into account and aggregates the domain information of the vertices to generate the features of the vertices. It is possible that for most vertices, their local neighborhood in a *k*-truss is roughly the same as its neighbors in the whole graph.

In terms of running time, it is noted that the additional computational cost of computing the truss variants is negligible. For further analysis, the authors calculated the average truss degeneracy $\omega^*_{avg}(G)$ of the graphs for each dataset (shown in Figure 2), and it is visually observed that the running time on a dataset has a close relationship with its $\omega^*_{avg}(G)$ value. In particular, on the IMDB-BINARY and IMDB-MULTI datasets, 6 times more time is needed to calculate the truss variants than the base kernels. One issue of interest is that the WL variant has a relatively larger runtime ratio than the other

Table 3.
Values of the hyperparameters of the graph kernels used in our experiments

Kernels	Hyperparameters
GR	$k\epsilon\{3,4,5\}, n_{sample}\epsilon\{150,200,500\}$
WL	$h\epsilon\{1,2,3,4,5,6\}$
NH	$h\epsilon\{1,2,3,4,5,6\}$
SP	-
PM	$d\epsilon\{4,6,8,10\}, L\epsilon\{2,4,6\}$

Table 4.
Comparison of classification accuracy (± standard deviation) of the graphlet kernel (GK), shortest-path kernel (SP), Weisfeiler-Lehman kernel (WL), pyramid match kernel (PM), neighborhood hash kernel (NH), to their truss variants on 12 graph classification datasets

	MUTAG	PTC-MR	ENZYMES	PROTEINS	NCI1	NCI109
GR	78.51 ±1.86	56.1 ± 1.1	22.09 ± 1.39	71.25 ± 0.45	61.33 ± 0.28	61.32 ± 0.24
Truss GR	**78.78 ±2.07**	**56.32 ± 1.11**	22.06 ± 1.41	**72.99 ± 0.44**	**61.66 ± 0.2**	**61.61 ± 0.34**
SP	80.43 ±2.84	59.53 ± 1.87	40.85 ± 1.52	75.72 ± 0.66	72.36 ± 0.28	72.56 ± 0.32
Truss SP	**80.66 ±2.01**	**59.98 ± 1.63**	**42.9 ± 2.3**	**76.4 ± 0.4**	72.29 ± 0.25	72.46 ± 0.3
WL	82.22 ±2.36	61.39 ± 1.86	50.48 ± 1.33	74.61 ± 0.57	81.68 ± 0.24	81.64 ± 0.24
Truss WL	82.16 ± 2.3	**61.49 ± 2.07**	49.89 ± 1.35	**74.87 ± 0.51**	81.56 ± 0.23	81.55 ± 0.21
PM	83.56 ± 1.98	57.43 ± 1.96	39.89 ± 1.44	73.68 ± 0.64	69.88 ± 0.4	68.97 ± 0.47
Truss PM	**83.74 ± 2.14**	**57.49 ± 1.74**	**43.68 ± 1.33**	**75.02 ± 0.69**	**70.3 ± 0.39**	**69.38 ± 0.4**
NH	85.85 ± 2.08	59.3 ± 1.74	52.77 ± 0.95	74.96 ± 0.6	83.74 ± 0.27	82.97 ± 0.29
Truss NH	85.66 ± 2.06	**60.52 ± 2.03**	51.46 ± 1.0	74.74 ± 0.55	83.52 ± 0.3	**83.13 ± 0.28**
	IMDB-BINARY	**IMDB-MULTI**	**REDDIT-BINARY**	**REDDIT-MULTI-5K**	**SYNTHETIC new**	**Synthie**
GR	61.22 ± 1.01	39.37 ± 0.82	73.85 ± 0.25	33.24 ± 0.19	45.12 ± 2.33	31.12 ± 1.76
Truss GR	**70.9 ± 0.88**	**48.58 ± 0.6**	**75.86 ± 0.39**	**42.3 ± 0.27**	**57.07 ± 2.04**	**48.54 ± 1.77**
SP	72.23 ± 0.77	51.2 ± 0.83	87.21 ± 0.24	51.66 ± 0.13	81.73 ± 0.87	49.12 ± 2.46
Truss SP	**72.31 ± 0.62**	50.53 ± 0.47	**87.41 ± 0.15**	**51.69 ± 0.17**	**83.33 ± 0.82**	**52.62 ± 1.68**
WL	72.46 ± 0.76	50.9 ± 0.46	70.15 ± 0.78	49.46 ± 0.33	97.77 ± 0.36	50.09 ± 1.9
Truss WL	71.69 ± 1.35	**51.03 ± 0.56**	**70.52 ± 0.76**	**49.93 ± 0.27**	96.96 ± 0.6	49.73 ± 2.34
PM	73.04 ± 1.07	50.17 ± 0.57	83.82 ± 0.45	50.36 ± 0.38	65.13 ± 2.16	48.08 ± 2.09
Truss PM	**73.89 ± 0.9**	**50.33 ± 0.59**	**85.0 ± 0.58**	**51.03 ± 0.28**	**71.71 ± 1.66**	**49.37 ± 2.06**
NH	66.24 ± 0.92	47.67 ± 0.41	85.49 ± 0.38	50.64 ± 0.24	96.15 ± 0.44	46.3 ± 2.46
Truss NH	**69.1 ± 0.8**	**48.02 ± 0.51**	84.4 ± 0.38	**51.07 ± 0.31**	92.48 ± 0.49	**49.48 ± 1.66**

three methods. The intuition is that WL kernel method may compute the kernel matrix significantly faster than other methods. The authors counted the longest and shortest time for the five graph kernels to compute the kernel matrix over the entire graph (i.e., 2-truss) on the ENZYMES dataset, with the values of the hyperparameters of the graph kernels set to the maximum and minimum values of the chosen parameter range. Since SP has no hyperparameters, it has only one computation time. Figure 3 shows the result which is in line with the authors' intuition. The calculation time for SP is also relatively short. In general, the additional computational cost is acceptable because the accuracy rate will increase accordingly.

Overall, the truss variants perform better on the social network datasets compared to the bioinformatics and cheminformatics datasets. To investigate the reason, the authors studied the degree distribution of the datasets, as shown in Figure 4. One notable difference between social network and chemoinformatics/bioinformatics datasets is that the former follows the power-law distribution common in nature and social life, while the latter does not. In Figure 4, the social network datasets exhibit significant long-tail characteristics. Since REDDIT-BINARY and REDDIT-MULTI-5K datasets have a degree distribution interval greater than two thousand, the authors use the logarithmic axes to show their degree distribution. For these datasets, it is assumed that the higher-order trusses

Figure 2.
Comparison of running time for kernel matrix computation of base kernels vs their truss variants. The values represent the ratio of running time of truss variants to the basic kernels. Some dataset names have been abbreviated. The cross indicates the $\omega^{*}_{avg}(G)$ of the dataset

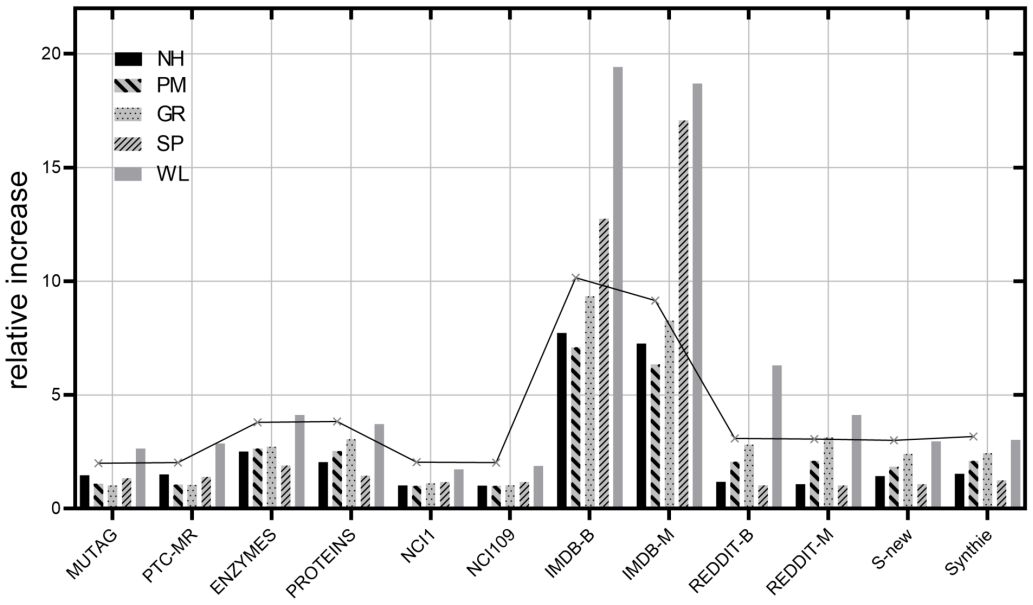
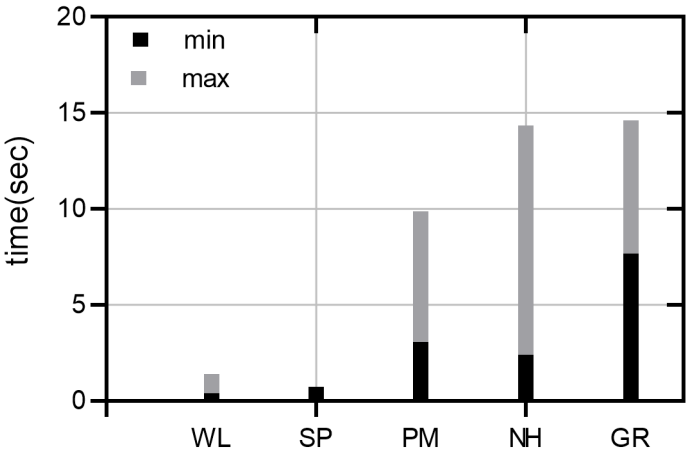
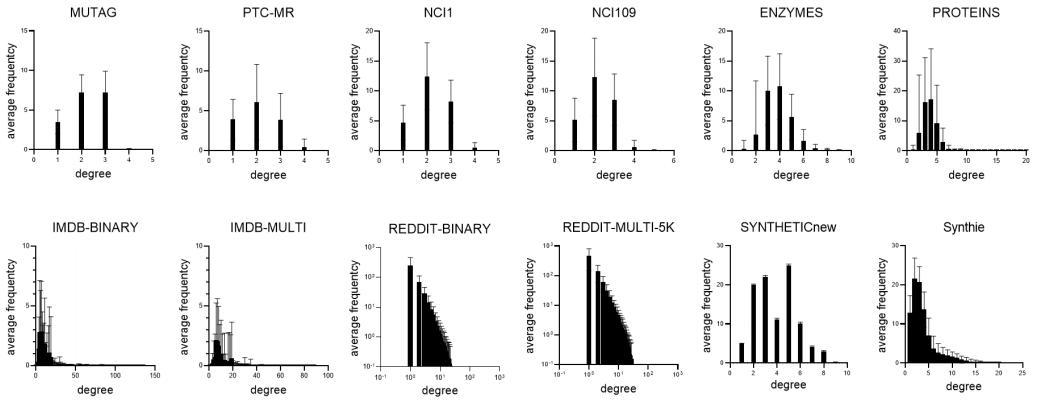


Figure 3.
Running time of the five base kernels on the entire graph (i.e., 2-truss) of the ENZYMES dataset with the values of the hyperparameters of the graph kernels set to the maximum and minimum values of the chosen parameter range



of the graphs capture the most representative areas of the graphs. Conversely, graphs similar to bioinformatics datasets may lead to low truss degeneracies, and many nodes may end up with the same truss value. The basic structural differences between datasets may be a very plausible reason for the greater accuracy improvements in social network datasets compared to chemoinformatics or

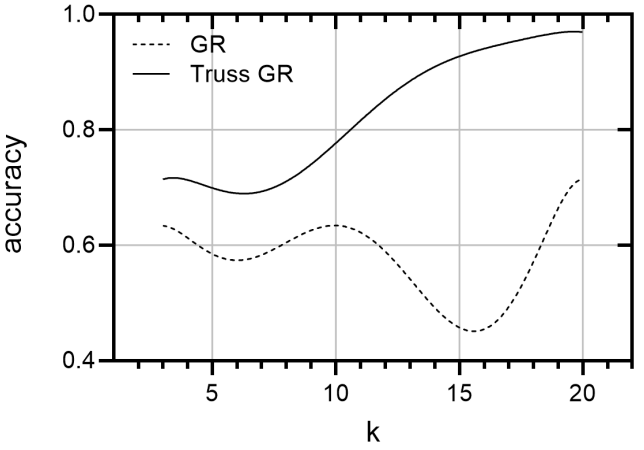
Figure 4.
Degree distribution of the datasets. Both axes of the figures of REDDIT-BINARY and REDDIT-MULTI-5K are logarithmic



bioinformatics datasets. For synthetic datasets, SYNTHETICnew and Synthie have very different degree distributions, with the latter being closer to the social network datasets. Truss GR improves accuracy by over 15% on the Synthie dataset, truss SP, truss PM and truss NH all improve accuracy to a certain extent. The framework is proved to be also effective on symmetric datasets and performs better on the Synthie.

Finally, the authors choose the IMDB-BINARY dataset and compare the execution of GR and its truss variant on a part of the whole range of k-trusses on the IMDB-BINARY dataset. For $k \in \{2,\ldots,20\}$, the GR kernel and its truss variant are computed to perform graph classification. The authors compare the achieved classification accuracies, and the results are in Figure 5. It is noted that the truss variant obtains a higher accuracy compared to GR. In particular, the accuracy of GR drops to a minimum of around k = 15 while the accuracy of truss GR remains high. The accuracy of truss GR gradually increases in the interval of k values, which is in line with our expectations. Consistent conclusions can be drawn on other data sets, which are omitted here.

Figure 5.
Classification accuracy of GR and its truss variant for k -trusses on the IMDB-BINARY dataset for $k \in \{2,\ldots,20\}$



DISCUSSIONS

This paper proposes a framework for comparing graphs at multiple different scales, which applies to any graph similarity algorithm. The framework capitalizes on the well-known k-truss decomposition of graphs to build a hierarchy of nested subgraphs. For existing base graph kernels, the framework generates efficient truss variants, enhances the ability of the base graph kernels to represent complex substructures, and improves the performance of the base graph kernel on graph classification tasks. The authors evaluate the performance of the framework on a set of benchmark datasets for graph classification tasks. In most cases, the truss variants perform well and achieve greater improvements over the base kernel, while their time complexity remains very attractive.

The framework is a powerful tool for improving the performance of graph kernels and can be applied to any graph comparison algorithm. The truss variants of graph kernels generated by the framework perform well on graph classification tasks and can be used in a wide range of applications such as chemical bioinformatics, cyber security, computer vision, and many others. For example, in biochemical information, the properties and functions of a substance can be predicted based on the similarity of the structure to substances with known functions; in cyber security, unknown code samples are compared with known malware samples and clean code to detect malware; in computer vision, images can be classified, especially in biomedical imaging, to distinguish between different brain states and predict whether a person has a certain disease or not.

CONCLUSION

In this article, the authors propose a framework that enhances the performance of graph kernels, which uses truss decomposition to allow existing algorithms to compare graphs at different scales. Extensive experiments demonstrate the improvement of the truss variants over the basic graph kernels in terms of graph classification accuracy with a relatively small increase in time complexity.

ACKNOWLEDGMENTS

Conflict of Interest

The authors of this publication declare there is no conflict of interest.

Funding Agency

This work was supported by the National Key Research and Development Program of China [grant number 2020YFB1005900]; the National Natural Science Foundation of China (NSFC) [grant number 62122042]; and the Shandong University multidisciplinary research and innovation team of young scholars [grant number 2020QNQT017].

REFERENCES

Akbas, E., & Zhao, P. (2017). Truss-based community search: a truss-equivalence based indexing approach. *Proc. VLDB Endow.,* 10. doi:10.14778/3137628.3137640

Borgwardt, K. M., & Kriegel, H. P. (2005). Shortest-path kernels on graphs. *Proceedings - IEEE International Conference on Data Mining, ICDM*, (pp. 74–81). IEEE. doi:10.1109/ICDM.2005.132

Cai, Z., He, Z., Guan, X., & Li, Y. (2018). Collective Data-Sanitization for Preventing Sensitive Information Inference Attacks in Social Networks. *IEEE Transactions on Dependable and Secure Computing*, *15*(4), 577–590. doi:10.1109/TDSC.2016.2613521

Cai, Z., Xiong, Z., Xu, H., Wang, P., Li, W., & Pan, Y. (2021). Generative Adversarial Networks: A Survey Toward Private and Secure Applications. *ACM Computing Surveys*, *54*(6), 1–38. doi:10.1145/3459992

Chen, P., Chou, C., & Chen, M. (2014). *Distributed Algorithms for k -truss Decomposition*.

Cohen, J. (2009). Graph Twiddling in a MapReduce World. *Computing in Science \& Engineering, 11*.

Cohen, J. D. (2008). Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*.

Ebadian, S., & Huang, X. (2019). Fast Algorithm for K-Truss Discovery on Public-Private Graphs. *ArXiv*, 2258–2264.

Haussler, D. (1999). *Convolution kernels on discrete structures*.

Hido, S., & Kashima, H. (2009). A linear-time graph kernel. *Proceedings - IEEE International Conference on Data Mining, ICDM*, (pp. 179–188). IEEE. doi:10.1109/ICDM.2009.30

Horváth, T., Gärtner, T., & Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. doi:10.1145/1014052.1014072

Hua, Q. S., Shi, Y., Yu, D., Jin, H., Yu, J., Cai, Z., Cheng, X., & Chen, H. (2020). Faster Parallel Core Maintenance Algorithms in Dynamic Graphs. *IEEE Transactions on Parallel and Distributed Systems*, *31*(6), 1287–1300. doi:10.1109/TPDS.2019.2960226

Huang, X., Cheng, H., Qin, L., Tian, W., & Yu, J. X. (2014). Querying k-truss community in large and dynamic graphs. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM. doi:10.1145/2588555.2610495

Huang, X., Lu, W., & Lakshmanan, L. V. S. (2016). Truss decomposition of probabilistic graphs: Semantics and algorithms. *Proceedings of the ACM SIGMOD International Conference on Management of Data, 26-June-20*, (pp. 77–90). ACM. doi:10.1145/2882903.2882913

Jin, H., Wang, N., Yu, D., Hua, Q. S., Shi, X., & Xie, X. (2018). Core Maintenance in Dynamic Graphs: A Parallel Approach Based on Matching. *IEEE Transactions on Parallel and Distributed Systems*, *29*(11), 2416–2428. doi:10.1109/TPDS.2018.2835441

Kang, U., Tong, H., & Sun, J. (2012). *Fast Random Walk Graph Kernel*. SDM. doi:10.1137/1.9781611972825.71

Kashima, H. (2003). *Marginalized Kernels Between Labeled Graphs.*, *2002*, 321–328.

Katunka, A. M., Yan, C., Serge, K. B., & Zhang, Z. (2017). K-Truss Based Top-Communities Search in Large Graphs. *Proceedings - 5th International Conference on Advanced Cloud and Big Data, CBD 2017*, (pp. 244–249). IEEE. doi:10.1109/CBD.2017.49

Kondor, R., & Pan, H. (2016). The multiscale laplacian graph kernel. *Advances in Neural Information Processing Systems*, 2990–2998.

Latapy, M. (2008). Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Ence*, *407*(1–3), 458–473. doi:10.1016/j.tcs.2008.07.017

Li, B., Zhu, X., Chi, L., & Zhang, C. (2012). Nested Subtree Hash Kernels for Large-Scale Graph Classification over Streams. *2012 IEEE 12th International Conference on Data Mining*, (pp. 399–408). IEEE.

Li, L., Zhang, F., & Liu, G. (2019). Multi-fuzzy-objective graph pattern matching with big graph data. *Journal of Database Management*, *30*(4), 24–40. doi:10.4018/JDM.2019100102

Liu, Q., Zhao, M., Huang, X., Xu, J., & Gao, Y. (2020). Truss-based Community Search over Large Directed Graphs. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM. doi:10.1145/3318464.3380587

Lotito, Q. F., & Montresor, A. (2020). *Efficient Algorithms to Mine Maximal Span-Trusses From Temporal Graphs*.

Luo, Q., Yu, D., Cai, Z., Lin, X., & Cheng, X. (2021). Hypercore maintenance in dynamic hypergraphs. *Proceedings - International Conference on Data Engineering, 2021-April*, (pp. 2051–2056). IEEE. doi:10.1109/ICDE51399.2021.00199

Luo, Q., Yu, D., Cheng, X., Cai, Z., Yu, J., & Lv, W. (2020). Batch Processing for Truss Maintenance in Large Dynamic Graphs. *IEEE Transactions on Computational Social Systems*, *7*(6), 1435–1446. doi:10.1109/TCSS.2020.3026574

Luo, Q., Yu, D., Li, F., Dou, Z., Cai, Z., Yu, J., & Cheng, X. (2019). Distributed core decomposition in probabilistic graphs. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 11917 LNCS. Springer International Publishing. doi:10.1007/978-3-030-34980-6_2

Luo, Q., Yu, D., Zheng, Y., Sheng, H., & Cheng, X. (2021). Core-GAE: Towards Generation of IoT Networks. *IEEE Internet of Things Journal*, *4662*(c), 1–8. doi:10.1109/JIOT.2021.3085882

Morris, C., Kersting, K., & Mutzel, P. (2017). *Global Weisfeiler-Lehman Graph Kernels*.

Nikolentzos, G., Meladianos, P., Limnios, S., & Vazirgiannis, M. (2018). A Degeneracy Framework for Graph Similarity. *IJCAI (United States)*.

Nikolentzos, G., Meladianos, P., & Vazirgiannis, M. (2017). Matching node embeddings for graph similarity. *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, (pp. 2429–2435). AAAI.

Orsini, F., Frasconi, P., & De Raedt, L. (2015). Graph Invariant Kernels. *IJCAI (United States)*.

Peng, Y., Zhang, J., Zhou, C., & Meng, S. (2022). Knowledge Graph Entity Alignment Using Relation Structural Similarity. *Journal of Database Management*, *33*(1), 1–19. doi:10.4018/JDM.305733

Schölkopf, B., & Smola, A. (2001). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. *Journal of the American Statistical Association*, *98*, 489.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, *12*, 2539–2561.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., & Borgwardt, K. M. (2009). *Efficient graphlet kernels for large graph comparison*. AISTATS.

Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., & Vazirgiannis, M. (2020). GraKeL: A Graph Kernel Library in Python. *J. Mach. Learn. Res., 21*.

Tian, X., & Han, H. (2022). Deep Convolutional Neural Networks With Transfer Learning for Automobile Damage Image Classification. *Journal of Database Management*, *33*(3), 1–16. doi:10.4018/JDM.309738

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., & Borgwardt, K. M. (2010). Graph kernels. *Journal of Machine Learning Research*, *11*, 1201–1242. doi:10.1007/978-1-4899-7687-1_349

Wang, J., & Cheng, J. (2012). Truss decomposition in massive networks. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *5*(9), 812–823. doi:10.14778/2311906.2311909

Wang, N., Yu, D., Jin, H., Qian, C., Xie, X., & Hua, Q. S. (2017). Parallel Algorithm for Core Maintenance in Dynamic Graphs. *Proceedings - International Conference on Distributed Computing Systems, 0*, 2366–2371. doi:10.1109/ICDCS.2017.288

Wu, L., Yen, I. E. H., Zhang, Z., Xu, K., Zhao, L., Peng, X., Xia, Y., & Aggarwal, C. (2019). Scalable global alignment graph kernel using random features: From node embedding to graph embedding. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 1418–1428). IEEE. doi:10.1145/3292500.3330918

Yanardag, P., & Vishwanathan, S. V. N. (2015a). *A Structural Smoothing Framework For Robust Graph Comparison*. NIPS.

Yanardag, P., & Vishwanathan, S. V. N. (2015b). Deep Graph Kernels. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. doi:10.1145/2783258.2783417

Yu, D., Zhang, L., Luo, Q., Cheng, X., Yu, J., & Cai, Z. (2020). Fast skyline community search in multi-valued networks. *Big Data Mining and Analytics*, *3*(3), 171–180. doi:10.26599/BDMA.2020.9020002

Zhao, F., & Tung, A. K. H. (2012). Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *6*(2), 85–96. doi:10.14778/2535568.2448942

Zhou, W., Huang, H., Hua, Q. S., Yu, D., Jin, H., & Fu, X. (2021). Core decomposition and maintenance in weighted graph. *World Wide Web (Bussum)*, *24*(2), 541–561. doi:10.1007/s11280-020-00857-0

Yanwei Zheng received the B.S. degree from the Shandong Jianzhu University in 2006, the M.S. degree in Shandong University in 2006, and the Ph.D degree from Beihang University in 2019. He is currently an associate researcher in the School of Computer Science and Technology, Shandong University. His research interests include computer vision, machine learning, and power automation.

Zichun Zhang received the B.E. degree from the School of Computer Science and Technology, Shandong University, in 2019. She is currently pursuing Master Degree at the School of Computer Science and Technology, Shandong University. Her research interests include graph analysis and data mining.

Qi Luo received the B.S. in computer science from Northeastern University at Qinhuangdao, China, in 2015, and received the M.E. and Ph.D degrees from Shandong University, China, in 2015 and 2018, respectively. His research interests include graph analysis and distributed computing.

Zhenzhen Xie received the M.S. degree in computer science from Jilin University, Changchun, China, in 2014, and the Ph.D. degree, in 2021. Her research areas are reinforcement learning, Internet of Things, and representation learning.

Dongxiao Yu received the B.S. degree in 2006 from the School of Mathematics, Shandong University and the Ph.D degree in 2014 from the Department of Computer Science, The University of Hong Kong. He became an associate professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, in 2016. He is currently a professor in the School of Computer Science and Technology, Shandong University. His research interests include wireless networks, distributed computing and graph algorithms.