A Survey of the Link Prediction on Static and Temporal Knowledge Graph

Thanh Le[1,2], Hoang Nguyen[1,2], Bac Le[1,2]

[1] Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam

Correspondence: Thanh Le, lnthanh@fit.hcmus.edu.vn
Communication: received 17 June, revised 11 August, accepted 25 August
Digital Object Identifier: 10.32913/mic-ict-research.v2021.n2.972

***Abstract*: Link prediction in knowledge graphs gradually plays an essential role in the field of research and application. Through detecting latent connections, we can refine the knowledge in the graph, discover interesting relationships, answer user questions or make item suggestions. In this paper, we conduct a survey of the methods that are currently achieving good results in link prediction. Specially, we perform surveys on both static and temporal graphs. First, we divide the algorithms into groups based on the characteristic representation of entities and relations. After that, we describe the original idea and analyze the key improvements. In each group, comparisons and investigation on the pros and cons of each method as well as their applications are made. Based on that, the correlation of the two graph types in link prediction is drawn. Finally, from the overview of the link prediction problem, we propose some directions to improve the models for future studies.**

Keywords: *Link prediction, static knowledge graph, temporal knowledge graph.*

I. Introduction

In the first half of 2012, Google announced a new data type called the Knowledge Graph (KG). With this platform, Google hopes to make its search engine 1000 times smarter [1]. Although this term was mentioned as early as 1972 [2], KG has received much more attention from the research community and business since the Google announcement. Companies such as Facebook [3], IBM [4], LinkedIn [5], Microsoft [6], Amazon [7] announced the integration of KG into their technology platforms, respectively. In Google Scholar, the number of articles related to KG in the period from 2012 to 2021 accounts for nearly 1.8 million articles, on par with other fields such as computer vision and natural language processing. It proves that the attractiveness of this field in the research community. In addition to KGs developed by commercial organizations and not widely shared, open KGs are being formed and developed rapidly. Some of the representatives in this open KG include DBPedia [8], ConceptNet [9], WordNet [10], Freebase [11], Google's Knowledge Vault [12], WebIsAGraph [13], YAGO [14], Countries [15]. With these KG datasets, researchers can easily experiment and improve their models.

In terms of application, knowledge graphs also have many achievements in today's life. In the Google search engine, KG not only produces results that are relevant to a user's search but also makes broad connections between other potentially relevant data. Hence, the user can expand the topic quickly and increase the ability to receive more relevant information. Facebook Social Search [16] describes the relationship between people, events. From that, Facebook suggests friend connections, fanpages, exciting events for users effectively. Some other applications such as service of fact checking [17], question answering [18], entity linking [19], and many others tasks [20]. The practical applications have spurred research efforts to develop the algorithms on KGs at a large scale and with high performance. In 2021, Shaoxiong Ji et al. [21] synthesized the general picture in the field of knowledge graphs (Fig. 1). The author has divided the research on KG into four main groups: knowledge representation learning, knowledge acquisition, temporal knowledge graph, knowledge aware applications. In each branch, the author shows the subproblems as well as the main solutions.

We have found that for applications running on KG to be highly effective, it is necessary to build good knowledge graph data first. Although depending a lot on the people involved in the data collection, it is challenging to expect perfect data initially. Some problems can occur, such as incomplete data, noises, ambiguity, or missing data. The tools helping detect abnormal as well as discovering the missing knowledge are required. In this paper, we are interested in predicting missing links in KGs. In the process of data formation, KGs often encounter problems such as

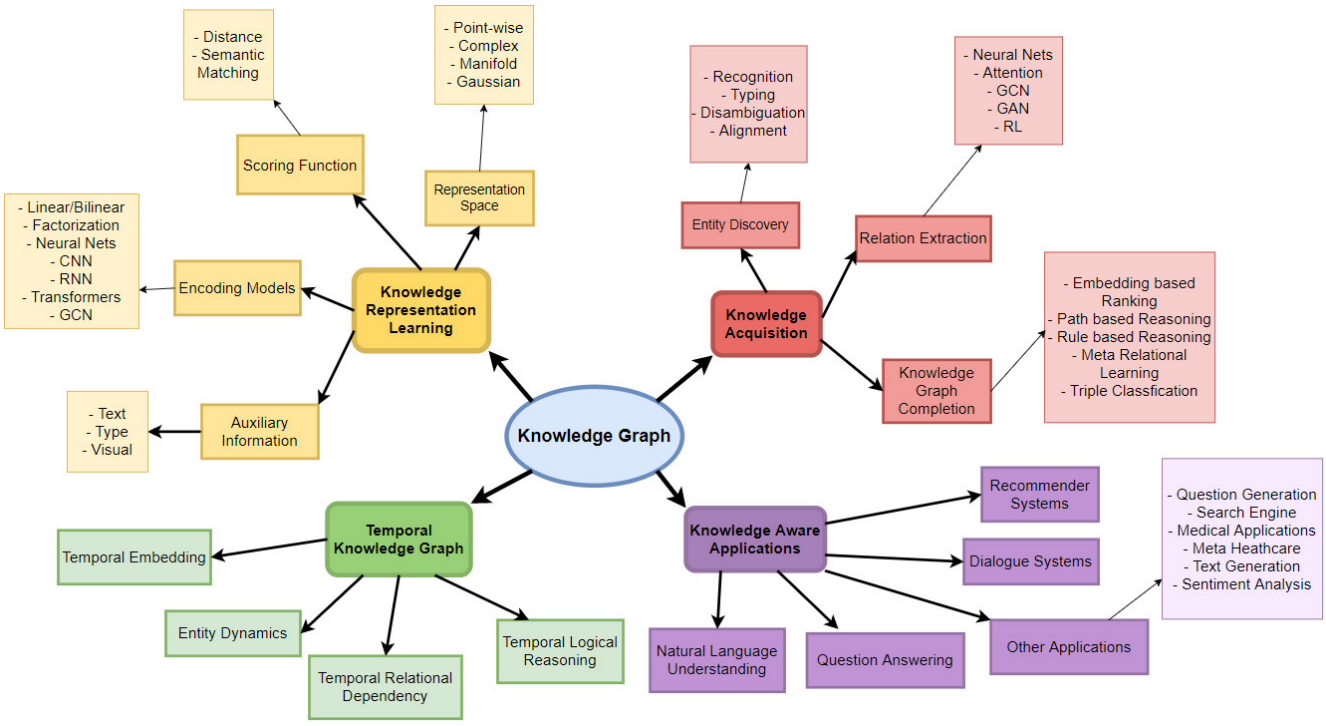


Figure 1. Knowledge graph overview by Shaoxiong Ji [21]

missing or incomplete information. Specifically, there are some relations between entity A and entity B in real life, but in the KG database, they do not exist. Therefore, the task of predicting the link was carried out as an inevitable consequence.

The purpose of link prediction to determine the relations formed between the two entities in a graph that have not previously been linked. From this definition, the researchers divided into two different problems. The first problem is predicting the links to complete the graph (KG completion). Some researchers consider the problem of graph completion as a problem of link prediction on a static graph. They assumed that the world does not change. In other words, closed-world assumption in which all entities and relationships don't change. The task is to discover hidden associations from the current observed structure of the KG. Other terms can be used instead, such as Entity Alignment, Hidden Link Prediction, Missing Link Prediction.

The second problem is to predict the links that form in the future. KG is referred to as a dynamic graph or a historical snapshot. The task is similar to analyzing time series data. This problem is often more challenging than the first problem due to the need to store many snapshots as well as identify significant changes in large-scale graphs over time.

In terms of applications, link prediction has many practical applications. For example, in social networks, we can see the vertices as users and edge the description of their acquaintance. We predict the link on this graph to make friend suggestions [22]. In the Covid epidemic, places and people are the vertices, and the edge can be the time and the way of travel. We use link prediction to trace people at risk of infection [23]. In a criminal investigation, we can predict the cooperation in the actions of criminals [24]. Link prediction can also be used to generate answers in a question answering system [25]. There are still many other applications that can be applied to the link prediction problem, which we discuss in detail in the following sections.

Many surveys are related to link predictions, as summarized in the Related Work section, but most of them focus only on a specific type of KG. This paper takes two kinds together to get a broader view of the KG link prediction domain. Thereby, we can compare significant differences, analyze the suitability of each solution for each group. Furthermore, interesting ideas can be carried from one group to another. To do that, we synthesize previous surveys, add new methods, and analyze the significant improvements in recent times.

Briefly, in this paper, we survey state-of-the-art methods in link prediction problems on both static and temporal knowledge graphs. We begin by reviewing existing surveys in the next section. In it, we list the methods each article describes and analyze the strengths and weaknesses of each. Section III, Section IV, and Section V describe the symbols

TABLE I
SURVEYS RELATED TO STUDIES IN KG LINK PREDICTION FOR THE PERIOD FROM 2011 TO 2021.

Paper	Author	Year	Publisher	Type
Link Prediction in Complex Networks: A Survey	Lu [26]	2011	PHYSA	Journal
A survey of link prediction in complex networks	Martínez [27]	2016	ACM Computing Surveys	Journal
Knowledge Graph Embedding: A Survey of Approaches and Applications	Wang [28]	2017	TKDE	Journal
Knowledge graph refinement: A survey of approaches and evaluation methods	Paulheim [29]	2017	Semantic Web	Journal
Graph Embedding Techniques, Applications, and Performance: A Survey	Goyal [30]	2017	Knowledge-based Systems	Journal
A systemic analysis of link prediction in social network	Haghani [31]	2019	Artificial Intelligence Review	Journal
A Comprehensive Survey of Knowledge Graph Embeddings with Literals: Techniques and Applications	Gesese [32]	2019	ESWC	Conference
Embedding based Link Prediction for Knowledge Graph Completion	Biswas [33]	2020	CIKM	Conference
Knowledge Graph Completion: A Review	Chen [34]	2020	IEEE Access	Journal
Knowledge Graphs	Hogan [35]	2020	arXiv	-
A Survey on Graph Neural Networks for Knowledge Graph Completion	Arora [36]	2020	arXiv	-
A survey on knowledge graph embedding: Approaches, applications and benchmarks	Dai [37]	2020	Data Analysis in Intelligent Communication Systems	Journal
Link prediction techniques, applications, and performance: A survey	Kumar [38]	2020	PHYSA	Journal
Temporal Link Prediction: A Survey	Divakaran [39]	2020	New Generation Computing	Journal
The Knowledge Graph Cookbook: Recipes that Work	Andreas [40]	2020	Monochrom	Book
Knowledge Graphs: Methodology, Tools And Selected Use Cases	Fensel [41]	2020	Springer	Book
A Survey on Knowledge Graph Embeddings for Link Prediction	Wang [42]	2021	Big Data and Artificial Intelligence Conference	Journal
A survey of link prediction in social networks	Al Hasan [43]	2021	Social Network Data Analytics	Book
A Survey on Knowledge Graphs: Representation, Acquisition and Applications	Ji [21]	2021	IEEE Transactions on Neural Networks and Learning Systems	Journal
Knowledge Graph Embedding for Link Prediction: A Comparative Analysis	Rossi [44]	2021	TKDD	Journal
Knowledge Graphs: Fundamentals, Techniques, and Applications	Kejriwal [45]	2021	The MIT Press	Book

and definitions in the link prediction problem, the datasets used to evaluate the models, and the metrics used for comparison. Section VI and Section VII present the main algorithms in static and dynamic graphs. In each group, the algorithms are divided into idea directions from which the algorithms improve gradually. Then, we evaluate models in each graph type in Section VIII and also compare the differences between the two groups to draw essential ideas in Section IX. Section X describes the current applications that have implemented the link prediction problem. Finally, in Section XI, we suggest some ideas that can be used to improve the models.

II. RELATED WORK

In this section, we introduce surveys related to link prediction on knowledge graphs. The aim is to investigate the current state of research in order to provide a better synthesis. In addition, we also add new methods with good results recently. Each survey is summarized, and key features are described. Thereby, we selected and adjusted the organization of solution groups in line with recent improvements. Selected surveys include KG overviews related to link prediction, foundational articles such as graph embedding, graph learning, and direct mentions of link prediction in the two branches of KG completion and temporal link prediction problems. In Table I, we summarize the primary surveys reviewed in our paper.

Because many surveys in Table I and later researches often inherit and overlap with previous articles, we only summarize some outstanding works. First, we describe articles that appeared pretty early, around 2012, when Google published their KG data. These articles show the context and challenges posed in the beginning. Then we select papers about 4 to 5 years after that point. This duration is enough to get outstanding algorithms in the link prediction problem. Finally, we present the surveys after about ten years from 2012, which is the present time. In general, up to now, the link prediction has achieved specific results and promises to be implemented in practice. Besides model accuracy, other related issues are also considered in

these surveys. Especially in 2020 and 2021, books related to KG have been published to become an official and organized reference for the scientific research community and university students. In the next sections, we delve into the analysis and evaluation of the methods compiled from all these surveys and recent works.

As we introduced in Secion I, the link prediction problem is divided into two subproblems: link prediction on static graphs and link prediction on dynamic graphs. Early, the problem on the static graph is considered because the data is simple and easy to collect, and proposed models are being formed. After these models work well on static graphs, the researchers find solutions on dynamic graphs. The problem on dynamic graphs is more complicated. Based on that, we separate the related work into two directions.

1. Surveys on Static Graphs

In the group of early-appearing surveys related to the link prediction on KGs, it can be seen that the author Lu et al. [26]. The author summarizes the recent advances in link prediction algorithms, highlights the views from a physics perspective, and outlines some of the potential future applications of link prediction. The author gives three groups of methods to solve link prediction on networks, including Similarity-Based, Maximum Likelihood, and Probabilistic Models. Although still have many limitations compared to the Markov chain and statistical models, such as slow running time or failure on some networks, the given methods can be used to solve link prediction in complex networks.

In 2016, Martínez et al. [27] presented techniques that can be considered as the backbone of link prediction. A new point compared to previous surveys is that the author has added domain-specific heuristic methods. In practice, link prediction methods are often performed on undirected networks by using topological features. The author categorizes link prediction techniques into four main categories: similarity-based methods, probabilistic and statistical methods, algorithmic methods, and preprocessing methods. There are three main directions in the similarity-based methods: local methods, global methods, and quasi-local methods. In the method based on probabilistic and statistical, there are four main groups: hierarchical structure model, stochastic block model, cycle formation model, local co-occurrence model. In the algorithmic methods, they are divided into three main approaches: classifier-based methods, metaheuristic-based methods, and factorization-based methods. Finally, the preprocessing methods include three main directions: low-rank approximation, unseen bigrams, and filtering (clustering). The author also provides a comparison table of the time complexity for all methods. A year later, Wang et al. [28] provides a systematic review of existing techniques in graph embedding, including state-of-the-arts and recent trends, as well as introduces how KG embedding can be applied, such as KG completion, relation extraction, question answering. The author proposes two techniques for embedding, including KG embedding with facts only and graph embedding with combining additional information. In the methods using only facts, the author divides them into two subgroups, including the translational distance model and the semantic matching model. Meanwhile, combining additional information mainly mentions four types: entity types, relation paths, textual description, and logical rule. Models representing entities and relations as vectors such as TransE [46], TransH [47], DistMult [48], and ComplEx [49] are more efficient because space and time complexity scales linearly with the dimension of entity embedding space.

After ten years, in Knowledge Graphs of Hogan [35], the author has introduced an overview of KG in aspects including common types of graph data models and how to query them and represent schema, identity, and context. In addition, the author describes ways of deductive and inductive reasoning to find new knowledge. Methods to form and enrich a graph, quality assessment, refine a KG. At the same time, Wang et al. [42] offers a comprehensive survey of KG models used for link prediction:

- The author gives some theoretical analysis as well as a comparison of the existing methods of graph embedding.
- The author divides the models into five main directions, including models based on convolutional neural networks, models based on context information, models based on the matrix, models based on graph convolutional networks, variant models of TransE, and analyze some representative models for each group.
- The author conducts experiments to provide information on the pros and cons of these models.

Most recently, Ji et al. [21] compare modern neural architecture for relational learning. The authors present a full view of KG's categories, especially KRL (Knowledge Representation Learning). KRL is categorized into four concepts: representation space, scoring function, encoding models, auxiliary information. In detail, representation space includes point-wise space, complex vector space, Gaussian distribution, manifold, and group. Meanwhile, the scoring function includes distance-based scoring function, semantic matching. And encoding model includes linear/-bilinear models, factorization models, NN, CNN, recurrent NN, transformers, GNNs. Finally, auxiliary information includes textual description, type information, visual information. This survey also presents knowledge acquisition and temporal knowledge graph and KG's application. Directly

related to the link prediction problem, Rossi et al. [44] conducted a study of 16 outstanding works by 2021. Then the author analyzed the algorithms on five standard datasets. The author also divides techniques into groups, including tensor decomposition models, geometric models, and deep learning models. Based on these groups, the author analyzes the efficiency and effectiveness of each algorithm. Finally, the author re-emphasizes the importance of link prediction research in the context of current and emerging problems.

A significant highlight is that there are three books related to KG released by publishers such as Springer, MIT Press, and Monochrom in 2020 and 2021. These are considered the first official books related to KG. Most of the books describe the basic concepts and knowledge for math problems. They present in detail issues, characteristics as well as solutions in each group. The applications of KG in practice are also mentioned. Andreas [40] focuses on system architecture and technologies to put KG into practice in real systems. Meanwhile, Fensel [41] focuses on creating knowledge for the KG as well as organizing the knowledge in the KG in a logical way and easy to use by stakeholders. We pay a lot of attention to the book of author Kejriwal [45] because the author presented quite details about the KG completion problem. The book also fully presents KG fundamental, KG construction as a foundation for all readers to understand this field.

2. Surveys on Temporal Graphs

Link prediction on (dynamic) temporal graphs is still a relatively new problem. The earliest survey in this branch can be considered to belong to Divakaran et al. [39] in 2020. In this survey, the author provides a comprehensive view of studies in temporal link prediction. Specifically, the author develops a new classification for methods based on various techniques used, and each method in each category is discussed in detail. The author also compares the advantages and disadvantages of each method. Challenges and future research directions are presented in detail. This survey can be considered the first to summarize the link prediction studies on dynamic knowledge graphs.

Generally, this problem has only received attention recently because of its challenges. The static graph itself is already large. Storing each snapshot of the graph over time makes the size even larger. Consequently, algorithms running well on static graphs may not be feasible on dynamic graphs. In addition, the link prediction on the static data still requires many breakthrough ideas to get high results. Another challenge is the sparseness and sensitivity to noise that occurs a lot in real-time data. Nonetheless, the applicability of link prediction on dynamic graphs is higher than on static graphs. Predicting the next purchase, suggesting friend connections, or detecting the fraud are exciting applications of this problem.

Currently, there are two main directions for implementing the link prediction problem on dynamic graphs. The first way is to adopt models that perform well on static graphs to run on dynamic data. This is done by adding a time field to the triple to form a quadruple. The strength of this group of methods lies in the fact that it can make good use of later improved models on static graphs. However, the model may take a long time to run, or it may not be easy to track objects to make good predictions. The famous models in this group include HyTE [50], Tero [51], and RTransE [52]. The second way is to apply times series methods from other fields to implementation on graph data. The strength lies in the good retention of temporal information of the objects. From there, make more reasonable predictions. However, the high performance on graph data needs to be further improved in the future. Models such as ITM [53], ARIMA [54], CALA [55] are the classic methods in this group.

In survey [39], the author also describes datasets of dynamic graphs. It can be seen that the number of datasets is quite many, but there is no consensus or standardization of datasets to make the studies comparable. This is also an issue that needs to be improved.

A narrower survey could also help with knowledge graph data that belongs to Dhote et al. [56]. The data the author works on is an online social network. Although this is not necessarily a knowledge graph, the algorithms can be considered and improved to run on dynamic knowledge graphs. In the survey, the author analyzes the development of links over time and introduces concepts of online social networks and link prediction problems. Studies also show that previous methods are impractical in link prediction because nodes can change over time and can be solved using temporal metrics to increase accuracy. Besides, the author also introduces some current methods that effectively exploit temporal information and gives some possible ideas in the future so that researchers can develop more.

In general, among the implementations, the embedding-based methods are the most important. The proof is that many works embed dynamic graphs such as TTransE [57], DE-SimplE [58], ATiSE [59] and show high efficiency compared to other methods. This also indicates a trend that researchers are focusing on doing on dynamic graphs. However, these models still need more research to get significant improvements.

III. Preliminaries and Problem Formalization

In this section, we describe the symbols and definitions used in the KG link prediction problem. Because the term KG has not been uniformly defined, we choose the

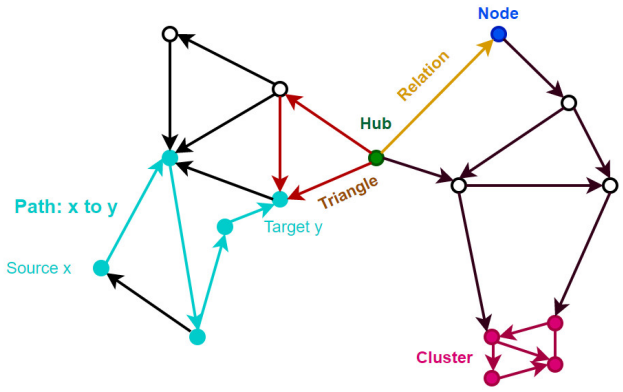


Figure 2. Components in a knowledge graph

TABLE II
GENERAL NOTATIONS AND DEFINITIONS

Notations	Definitions
$\mathcal{G}$	Graph
$\mathcal{G}_{\text{know}}$	Knowledge graph
$\mathcal{F}$	A set of facts
V, E	Vertex set, edge set
e, e_i	Entity, i^{th} entity
r, r_k	Relation, k^{th} relation
$\vec{e}, \vec{r}$	Entity embedding, relation embedding
(h, r, t)	A triple includes head, relation, tail
$(\mathbf{h}, \mathbf{r}, \mathbf{t})$	Embedding of head, relation and tail
T^v, T^e	Vertex type set, edge type set
N_e, N_r	The number of entity sets, the number of relational sets
$\mathbf{E}, \mathbf{R}$	Entity embedding matrix, relational embedding matrix
$f_r(\mathbf{h}, \mathbf{t})$	Scoring function
$\sigma(\cdot), g(\cdot)$	Non-linear activation function
$\mathbf{W}$	Weight matrix
$\mathbf{L}$	Loss function
$\langle \mathbf{h}, \mathbf{t} \rangle$	Hermitian dot product
$\mathbf{t} \otimes \mathbf{r}$	Hamilton product
$\mathbf{h} \circ \mathbf{t}, \mathbf{h} \odot \mathbf{t}$	Hadmard (element-wise) product
$concat(), [\mathbf{h}, \mathbf{r}]$	Vectors/matrices concatenation
ω	Convolutional filters
$*$	Convolutional operator
γ	Decaying factor
μ	Learning rate

descriptive method from the works of Cai [60], Goyal [30], and Wang [42]. We also present components and different ways of looking at relations in a knowledge graph. Fig. 2 illustrates some general features in a graph.

1. Notations

Because the survey involves many algorithms, the number of notations is very large. Therefore, we summarize only the notations that are used frequently in the models. In addition, some notations may vary depending on the model, so we describe them in mentioned algorithms later. Table II shows these notations and their description.

2. Definitions

Definition 1: A *graph* denoted by $G(V, E)$, where $v \in V$ is a vertex and $e \in E$ is an edge. G is formed by

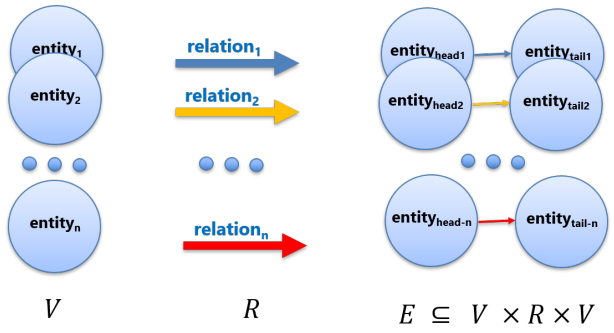


Figure 3. Knowledge graph definition

a vertex type (node type) through the mapping function $f_V : V \rightarrow T^V$ and an edge type through the mapping function $f_E : E \rightarrow T^E$, where T^V and T^E are a set of types of vertices and edges, respectively. Each vertex $v_i \in V$ belongs to a particular type, for example $f_V(v_i) \in T^V$. Similarly, if edge $e_{ij} \in E$, then $f_E(e_{ij}) \in T^E$. The adjacency matrix S of the graph G contains edges with non-negative weights if there are connected edges: $s_{ij} \geq 0$. If vertices v_i and vertices v_j have no joined edges, then $s_{ij} = 0$. For undirected weighted graph, $s_{ij} = s_{ji}$. The weight of the edge s_{ij} is often considered as a measure of the similarity between two vertices v_i and v_j. The higher the weight, the more similar the two vertices are.

Definition 2: A *knowledge graph* $G_{know} = (V, R, E)$ is a directed graph with a vertex set representing entities and an edge set representing relations between the vertices. In other words, edges describe events $E \subseteq V \times R \times V$ in real life. Based on that, they create head-relation-tail triples and are denoted by (h, r, t), where $h, t \in V$ are entities, and $r \in R$ are relations. Fig. 3 shows sets in a knowlege graph.

Definition 3: Given the input of the graph $G = (V, E)$ and the number of dimensions of the embedding d ($d \ll |V|$), *graph embedding* is defined as transforming G into a d-dimensional space, where the information from the graph is kept as much as possible. Graph properties can be quantified using neighborhood measures such as first- and higher-order neighborhoods. Each graph is represented as a d-dimensional vector (for the entire graph) or a set of d-dimensional vectors. In the second method, each vector represents the embedding of a part of the graph (vertices, edges, substructures).

Definition 4: *Knowledge graph embedding* is a technology for mapping the content of entities and relations in a knowledge graph to continuous low-dimensional vector space.

Definition 5: *Static graph* is graph which is created from the original knowledge and does not change over time.

Definition 6: A graph with connections added or removed over time is called a *dynamic graph*, *temporal graph*, *time-varying graph*, or *evolving graph*.

3. Types of Relations

Although relations in the KG are diverse, they can be classified based on some attributes. According to Wang [47], we can classify relations into four types, called mapping properties. They are 1-1 relation, 1-N relation, N-1 relation, and N-N relation. However, Sun [61] also classifies relations into four other types, called connectivity patterns. They are symmetric relation, antisymmetric relation, inverse relation, and compositional relation. Based on the nature of the objects, Gesese [32] divides relations into two simple types, including object relation and data type relation. By organized in sorts, algorithms can have different ways to predict relations more precisely. In the following, we describe the details of these types.

a) Mapping Properties

1-to-1 relation: is a relation in which a head entity has only one connection to the tail entity.

1-to-N relation: is a relation in which a head entity has many connections to the tail entities.

N-to-1 relation: is a relation in which many head entities have connections to the tail entity.

N-to-N relation: is a relation in which many head entities have connections to the tail entities.

b) Connectivity Patterns

Symmetric relation: A relation is symmetric if

$$\forall x, y : r(x, y) \Rightarrow r(y, x)$$

Antisymmetric relation: A relation is antisymmetric if

$$\forall x, y : r(x, y) \Rightarrow \neg r(y, x)$$

or

$$\forall x, y : r(x, y) \wedge r(y, x) \Rightarrow x = y$$

Inverse relation: A relation is inverse if

$$\forall x, y : r_2(x, y) \Rightarrow r_1(y, x)$$

Compositional relation: Relation r_1 is composed of 2 relations r_2 and r_3 if

$$\forall x, y, z : r_2(x, y) \wedge r_3(y, z) \Rightarrow r_1(x, z)$$

c) Object and Data Type Relation

Object relation is a relation that links an entity to another entity. This relation describes the interactions between the components in the system. For example, $(person{-}knows{-}person)$, $(person{-}in{-}position)$ are relations of this type.

Data Type relation is a relationship that links entities to data values that are mainly used to describe objects. This description is only for the object which is being described and not related to other objects. For example, $(person - named - A)$, $(person - characteristic - B)$ relations are data type relations. Although it is not easy to distinguish between object relation and data type relation in some situations, we can use the connection method to choose which type the relation should belong to.

In Fig. 4, we can see these green relations as object relation and black relations as datatype relation.

4. Definition of Link Prediction

Link prediction is the problem of finding possible links between two entities in KG. From there, we have two ways of understanding this problem. First, link prediction aims to identify missing links due to incomplete or lost data collection. In other words, we complete the data for the existing graph. Another form of link prediction is to identify possible links at the next time of the graph. That is, at the moment, the two entities have no interaction (link) with each other, but at the next time, the connection between them appear for some reason. This section formally describes these two types of link prediction problems as a premise for analyzing the models in each type. Fig. 5 shows two types of link Prediction.

a) Knowledge Graph Completion

KG Completion refers to the task of predicting missing interactions (Fig. 5a). It is also divided into two smaller types, which are entity prediction and relation prediction. Entity prediction is when given an entity and a relation, the models predict the other missing entity. More specifically, we predict a tail entity t from given $(h, r, ?)$ triple or predict a head entity h from given $(?, r, t)$ triple. Since it is difficult for models to give only one correct entity to a triple, we instead predict several potential entities and prioritize them. Hence, this problem is also called entity ranking or entity-sorting task. Specifically, the result of the prediction model is a list of entities along with their rank. Based on that, we evaluate the accuracy of the results—the higher rank of the correct entities, the better the model.

Another type of KG Completion problem is relation prediction. Given a triple $(h, ?, t)$, the goal of the model is to identify the missing relation in that triple. This problem is often less interested than the entity prediction

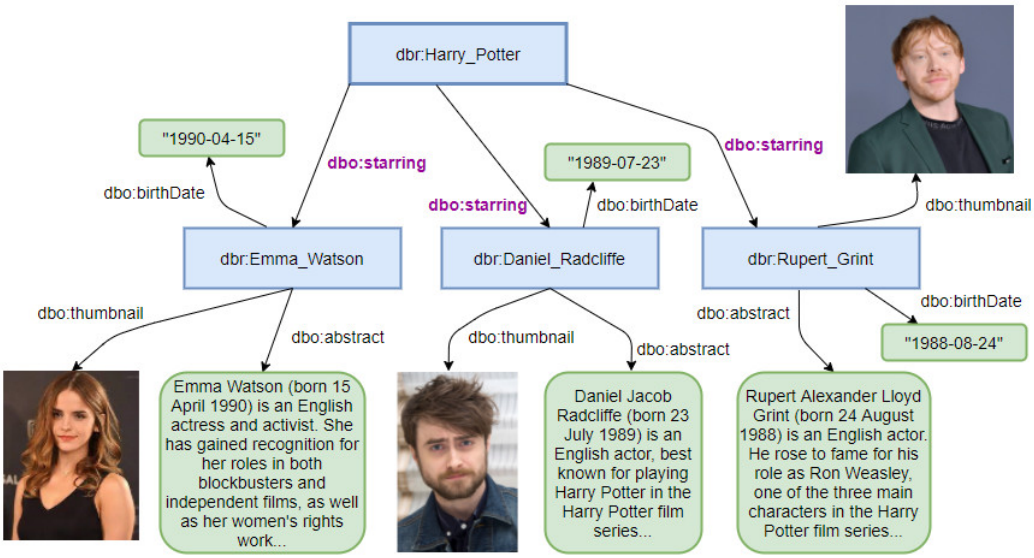


Figure 4. An example of object and data type relation [62]

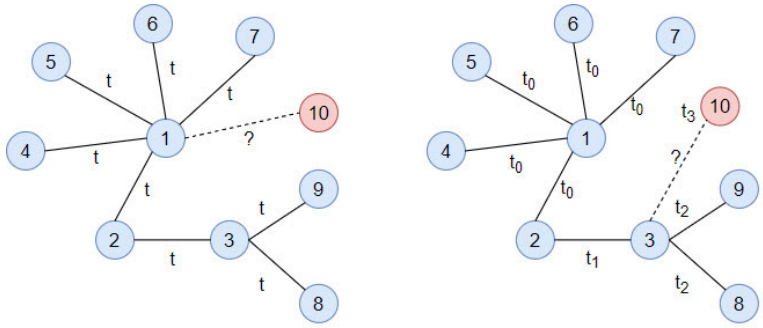


Figure 5. Comparison of two types of link prediction problem

problem because semantically, the two problems are similar. Moreover, relation prediction is much more difficult for the model to learn because it cannot represent the transformation from the source entity to the target entity. Therefore, in this paper, we only focus on investigating algorithms to predict triples $(h, r, ?)$ or $(?, r, t)$.

The inputs to models are positive triplets or facts in the graph. Furthermore, in some approaches, we need also to generate negative triplets to make the model learn to distinguish from existing triplets. Negative triplets are created by replacing h or t with any entity in the graph, such that the newly created triplets do not exist in the graph:

$$\begin{aligned}\Delta'_{(h,r,t)} =& \{(h', r, t) | h' \in E, (h', r, t) \notin \Delta\} \\ & \cup \{(h, r, t') | t' \in E, (h, r, t') \notin \Delta\}\end{aligned} \tag{1}$$

where Δ is the set of positive triplets and Δ' is the set of negative triplets. Each positive triplets (h, r, t) can only replace either the first entity (h', r, t) or the last entity (h, r, t'), not both at the same time.

The link prediction relies on these triplets to learn how to predict the missing entity. Embedding-based methods typically define a scoring function to estimate the plausibility of any fact:

$$\phi(\mathbf{h}, \mathbf{r}, \mathbf{t}) \tag{2}$$

The higher the score of ϕ, the more plausible the fact.

When testing, the head or tail in a triplet is hidden. Then, the model predicts the most suitable entity to place in it. Based on that, we evaluate the accuracy of the trained model.

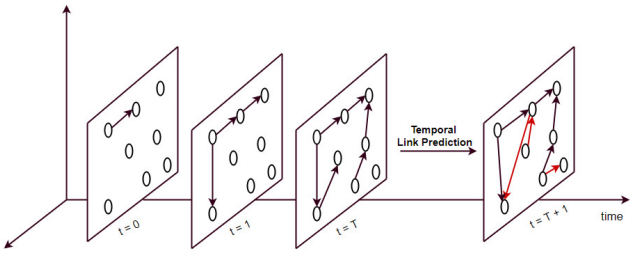


Figure 6. Temporal link prediction with a series of graph slices

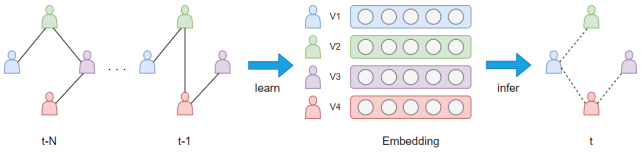


Figure 7. Temporal graph embedding for link prediction [63]

b) Temporal Link Prediction

Temporal link prediction is the problem of predicting the links that appear between entities in the coming time. This problem is also divided into two types depending on the different descriptions of the dynamic graph structure. The first way, the dynamic graph, includes vertices, relations, and timestamped links (Fig. 5b). When predicting the link on this graph, a model performs a time assignment on the newly generated edges. The second way, we consider the graph as a series of slices $(G_1, G_2, ..., G_t)$ corresponding to discrete time intervals $(1, ..., t)$. The goal is to predict a new association for the graph G_{t+1} at time $t + 1$. Fig. 6 illustrates the link prediction process in the second way.

Although the representations are different, they can all be converted to each other. The second representation is easier to embed objects in the embedding space because the interaction is independent of time. Fig. 7 illustrates a simple way to embed graphs for association prediction.

Depending on the approaches, the model's inputs can be triplets or graph slices over time. The model then trains on each of these triplets or slices to generate new triples or changes in the graph at the next time.

5. Positive and Negative Samples

Positive samples are triples appearing in the KG or set of facts. They are often considered as correct triples, or the relations between entities exhibit a clear and consistent connection. In contrast, negative samples are unobserved triples in the knowledge graph. They include implicit and unauthorized relations between entities. Because in the knowledge graph, we only observe correct triples, we need to generate these unobserved triplets for learning by models. Good negative sampling helps the learning models to create better classifiers. In other words, good negative sampling helps the models not to over-generalize the features. As a result, they improve the results of the predictive model.

Currently, there are many ways to generate negative samples such as random sampling, corrupting positive instance [64], typed sampling [65], relational sampling [65], Nearest Neighbor sampling [20]. Each method has strengths and weaknesses and depends on the nature of the data. The simplest technique among negative sampling techniques is random sampling. This method calls the triples (s, r, t) appearing in the graph as positive samples. Randomly generated instances that are not positive are considered negative. However, this method's disadvantage is that it makes many negative samples because the total number of possible triples is $|V|^2$, where V is the set of entities in the graph. Furthermore, we are unsure whether it is the correct negative sample because its randomness can produce accurate facts. And it can create trivial negative samples too.

Another way to create negative samples is to damage the positive instances. This technique is described in [64]. For each relationship (s, r, t) (positive triple) in the graph, we replace either the source or the target entity to produce negative samples. This technique produces negative instances closer to positive instances compare with random sampling. However, if the graph has entities with too many relationships, we may not have enough target (source) entities to damage positive samples. The trivial negative triples are still encountered in this method. According to the analysis, the FreeBase and NELL datasets [66] have a very close relationship between the entities. For example, the relationship *born_in* associates the entity types as *Person* and *City*. Therefore, the target entities used to create negative samples are selected to be compatible with the source entity. These negative triples are more explainable and are closer to the positive ones. If relations in the knowledge graph have characteristics that each entity join only one relationship, creating quality negative samples is quite simple. Just change the relationship between source and target or change one of the source and target entities that come with the new relation.

A different method of generating negative samples is to choose the nearest neighbour. After entities are encoded in an embedding space through a pre-train model, the process finds entities closest to the target entity so that they are not positive samples. Through that, the method generates negative instances that carry more certainty than other ways. Generally, each negative sampling method has its advantages and disadvantages. Therefore, the research to create quality samples continues to requires more effort.

IV. DATASETS

1. Static Datasets

Static datasets are used mainly in knowledge graph completion problems. Popular static datasets include FreeBase (FB15k, FB15k-237) [46, 68], WordNet (WN18, WN18RR) [46, 69], YAGO3-10 [70]. Besides, datasets such as Countries [15, 71], and DBpedia [72] are also considered in many studies. In the field of biology, OpenBioLink [73] and BioSNAP-DTI [74] are typical datasets. Each dataset has its own characteristics, such as containing symmetric, transitive, and other types of relations. From there, we can evaluate the predictive ability of the models on each type of relationship.

a) Freebase Dataset

FreeBase is a large structured knowledge base gathered from many sources, including individual wiki contributions. It was created from news in the world and used for research purposes. It is considered as a multi-disciplinary dataset that mainly describes some information related to movies, actors, awards, sports, and teams.

FB15k is a subset of the FreeBase dataset and consisting of 14,951 entities and 1,345 relations. Its relation types are symmetric, anti-symmetric, and inversed. The training set consists of 483,142 triplets, the validation set consists of 50,000 triplets, and the test set consists of 59,071 triplets. FB15k has the characteristics of multi-domains, reified, and test leakage. To reduce the influence of test leakage properties on the model results, Toutanova and Chen introduced the dataset FB15k-237.

FB15k-237 includes 14,541 entities and 237 relations . It has symmetric, antisymmetric, and transitive relation types. The training set includes 272,115 triplets, the evaluation set includes 17,535 triplets, and the test set includes 20,466 triplets. The dataset consists of textual descriptions of FreeBase, which are extracted from 200 million sentences from the ClueWeb12 corpus and annotations in FACC1 (FACC1: Freebase annotation of ClueWeb corpora).

b) WordNet Dataset

WordNet is an English vocabulary dataset which groups English words into sets of synonyms called synsets, provides short definitions and examples of usage, and records some relations between sets of synonyms or members. Furthermore, WordNet also contains a system of concepts that have many relationships with each other to form a complex graph.

WN18 is a subset of the WordNet dataset consisting of 40,943 entities and 18 relations. It has symmetric, antisymmetric, and inverse relations. Almost all 151,442 triplets are hypernym or hyponym relations. WN18 dataset adheres to a strict hierarchical structure. The training set consists of 141,442 triplets, the evaluation set consists of 5,000 triplets, and the test set consists of 5,000 triplets. However, the WN18 dataset has a test leakage problem. It means that the edges from the reciprocal relations such as hypernym and hyponym appear in the training set and inverse relations in the evaluation set, so models easily learn those rules to achieve high results.

To reduce leakage, Dettmers introduced the ***WN18RR*** dataset by removing those seven relationships. However, the WN18RR dataset still has four types of symmetric relationships as “also see,” “derivationally related form,” “similar to,” and “verb group.” Simple principles can exploit these relationships.

c) YAGO Dataset

YAGO3-10 is a massive repository of semantic knowledge derived from Wikipedia and GeoNames. YAGO3-10 is extracted from the YAGO3 dataset by selecting entities that participate in at least ten different relationships. The dataset consists of 123,182 entities, 37 relations with 1,079,040 triplets in the training set, 5,000 tuples in each validation and evaluation set.

d) OpenBioLink Dataset

OpenBioLink is a framework for evaluating link prediction models on biomedical data. It also contains datasets collected from multiple public sources related to bioinformatics. OpenBioLink provides four variants in directed and undirected settings, with and without quality cutoff. The triplets range from 8 million to 41 million, with about 185,000 to 486,000 entities and 28 to 32 relations. This dataset is considered helpful in evaluating the effectiveness of models when deployed in the field of biology.

Table III describes the characteristics of datasets commonly used in the link prediction problem.

2. Dynamic Datasets

To evaluate the models for link prediction on dynamic graphs, the researchers used a variety of evolving network datasets including DBLP, Twitter, Reality Mining, Haggle, Infectious, Hypertext 2009 [39]. These datasets are divided into main groups, including co-author and citation networks, communication networks, social networks, and human contact networks. Because there are too many datasets, authors often have to select some of them to run comparative experiments. In contrast, the datasets are almost normalized in the static graph and revolve around 4 to 5 datasets. However, in temporal link prediction, the datasets are still not unified due to the complexity of the relations arising at different times. It isn’t easy to compare the effectiveness of the methods in this link

TABLE III
STATISTICAL INFORMATION OF THE DATASETS USED IN EXPERIMENTS [67].

Datasets	#E	#R	#Train	#Valid	#Test (%)								
					#Total	#Sym	#Inv	#Com	#Others	#1-to-1	#1-to-N	#N-to-1	#N-to-N
FB15k	14,951	1,345	483,142	50,000	59,071	7.34	70.22	22.37	0.06	1.63	9.56	15.8	73.02
WN18	40,943	18	141,442	5,000	5,000	21.74	72.22	3	3.04	0.84	36.94	39.62	22.6
FB15k-237	14,541	237	272,115	17,535	20,466	0	0	90.4	9.6	0.94	6.32	22.03	70.72
WN18RR	40,943	11	86,835	3,034	3,134	34.65	0.29	8.33	56.73	1.34	15.16	47.45	36.06

prediction branch. We found that there are about four most used datasets through the survey, including DBLP, Haggle, Infectious, Reality Mining. Therefore, we focus the description on these datasets.

The ***DBLP*** file contains data related to the computer science bibliography. This dataset includes lists of research articles in the field of computer science. Two authors are linked if they share the same name on at least one paper. The number of vertices in this set includes 1,314,050 vertices and 18,986,618 edges.

Haggle describes the real human contact network where connections are determined based on groups of users carrying small devices. Edges represent the connection between two people who are in contact with each other. The dataset contains five days records of 274 people. The total number of links received is 28,244 connections.

A human proximity network tracked from one day at the Infectious is encapsulated into the ***Infectious*** dataset. Data describing face-to-face behavior of people during the exhibition *INFECTIOUS: STAY AWAY* in 2009 at the Science Gallery in Dublin. It consists of 410 vertices and 17298 edges.

Reality Mining was collected in 2004 by Nathan Eagle and Alex Pentland. It consists of 96 nodes and 1,086,404 edges that describe the physical relationships among 100 MIT students.

Due to a large number of datasets as well as the large size, there has not been a detailed description of the characteristics of the relation types as in the static graph. Therefore, we just summarize the information of each data set that is commonly used in the problem of link prediction on dynamic graphs.

V. METRICS

1. Mean Rank

The mean rank measure is abbreviated as MR. It is the average of the ranks obtained in the rating and is commonly used to evaluate a rating system that lists answers to queries:

$$MR = \frac{1}{|Q|}\sum_{q\in Q} rank(q) \tag{3}$$

where Q is a set of queries, q is the ranking position of a particular query, and $|Q|$ equal to the size of the test set. When predicting, we predict both head and tail for a corresponding sample in the test dataset. For example, we predict $(?, relation, tail)$ and $(head, relation, ?)$ for a test sample. In this case, q represents the entities we predicted with specific ranks, and $rank(q)$ represents the rank of the correct entity in predicted entities. We then calculate the average rank of both the head and tail predictions. Obviously, this measure is in the range of $[1, |E|]$ because there are at most $n-1$ edges from an entity to $n-1$ remaining entities and an edge connecting to its own entity (self-edge). The lower the MR is, the better model we get. However, this metric is susceptible to noise. The reason is that, in some situations, the correct entity is placed at the bottom of the list. To solve this problem, researchers use an additional measure, Mean Reciprocal Rank (MRR).

2. Mean Reciprocal Rank

The mean reciprocal rank measure is abbreviated as MRR. Similar to MR, MRR is a metric to evaluate a rating system for a list of answers to queries. However, MRR is the mean inverse of the ranks obtained in the rating. The higher MRR, the better the model results:

$$MRR = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{q} \tag{4}$$

The value domain of MRR is $0 \leq MRR \leq 1$. The larger MRR, the better the model result. This measure avoids sensitivity of noise because there are a number of triplets ranked terribly, which means the values of ranks near $|E|$.

3. Hit@K

Hit@K (H@K) is a measure of the probability that the rank is less than or equal to the given threshold K:

$$Hit@K = \frac{|q \in Q; q \leqslant K|}{|Q|} \tag{5}$$

where K is the number of triplets with the highest scores is ranked from large to small. Common values for K are $1, 3, 5, 10$. The higher $Hit@K$ is, the better the model

TABLE IV
EXAMPLE PREDICTED ENTITIES WITH THEIR RANKING FOR CALCULATING METRICS.

New created triplet	Score	Rank
t1	0.862	1
t2	0.786	2 (*)
t3	0.589	3
t4	0.436	4
t6	0.421	5 (*)
t9	0.397	6

(*) is the triplet that exists in test set

TABLE V
COMPARISON OF TENSOR DECOMPOSITION MODELS IN TERMS OF SCORING FUNCTIONS AND SPACE COMPLEXITY.

Model	Score function	Space Complexity
RESCAL	$\mathbf{h}^T\mathbf{r}\mathbf{t}$	$O(n_e d_e + n_r d_r^2)$
DistMult	$\mathbf{h} \times \mathbf{r} \times \mathbf{t}$	$O(n_e d_e + n_r d_e)$
ComplEx	$\mathbf{h} \times \mathbf{r} \times \bar{\mathbf{t}}$	$O(n_e d_e + n_r d_e)$
SimplE	$\frac{1}{2}(\mathbf{h}_h \times \mathbf{r} \times \mathbf{t}_t) + \frac{1}{2}(\mathbf{h}_t \times \mathbf{r}_{-1} \times \mathbf{t}_h)$	$O(2n_e d_e + 2n_r d_e)$
TuckER	$\mathbf{W} \times_1 \mathbf{h} \times_2 \mathbf{r} \times_3 \mathbf{t}$	$O(n_e d_e + n_r d_r + d_e d_r d_e)$
NepTuNe	$\mathbf{e}_t^T \cdot f_{act}(\mathbf{W} \times_1 \mathbf{e}_h \times_2 \mathbf{r})$	$O(n_e d_e + n_r d_r)$

results. *Hit@1* is closely related to *MRR* because when K = 1 both the formula of *Hit@K* and *MRR* are the same.

For examples shown in Table IV, there are 5 triples t_2, t_5, t_6, t_7, t_8 in the test set. We have $Hit@1 = 0/5 = 0$, $Hit@3 = 1/5 = 0.2$, $Hit@5 = 2/5 = 0.4$.

4. AUC Curve

AUC-PR, short for Precision-Recall Area Under Curve, is the average of the calculated precision points for each different recall value. The AUC-PR measure is used when the data is remarkably unbalanced.

Precision quantifies the number of correct positive predictions shown as Eq.6.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Recall quantifies the number of correct positive predictions made out of all positive predictions as be shown in Eq.7.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

5. RMSE

RMSE is the square root of the mean of the square of all of the error. In other words, RMSE measures the differences between predicted ranks and actual ranks. The formula of RMSE is showed in Eq. 8. When the value of RMSE error is smaller, it means that the prediction result is close to reality, so the better the model. This measure is more often used in temporal link prediction.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(predict - actual)^2}{N}} \tag{8}$$

VI. LINK PREDICTION IN STATIC GRAPH

In this section, we mention the methods that are highly effective in the link prediction problem. Firstly, the popular approach is based on embedding, which encodes entities and relations using various methods. For instance, we can transform knowledge graphs into vectors or matrices and then decompose them into smaller vectors or matrices. Or, in another way, we can apply neural networks to feed the relations through a series of nonlinear functions. We can also put them to a specific space and use transformations such as translation, rotation to predict connections. The second approach is to extract rules from the triplets and infer relations for entities based on that. The last group uses reinforcement learning methods to find a path as the model decision chain. We describe in detail the characteristics of each group and its prominent algorithms in the following paragraphs.

1. Graph Embedding Based Models

a) Tensor Decomposition

Models of this group consider link prediction as a tensor decomposition task by converting knowledge graph to a 3D adjacency matrix or a 3D tensor. Because of the incomplete graph, the 3D tensor isn't complete as well. The tensor is decomposed to some lower-dimensional vectors, which are used as entities and relation embedding. These models are expected to generalize the graph to detect latent information without overfitting. They also have no or little shared parameters, so they are usually lightweight and have fast training times. Table V gives detailed comparison of tensor decomposition models in terms of scoring functions and space complexity. In the following, we introduce some important models in the method group in turn.

RESCAL [75] is the most classical representative of this group. The model represents each relation as a matrix. In this matrix, the author describes the connection of entities in the graph corresponding to a defined relation. Based on that, each element becomes a feature vector in each relation domain. Fig. 8 shows relation matrices in RESCAL.

With this matrix representation, RESCAL is likely to overfit because there are so many parameters to learn for those matrices. Furthermore, RESCAL is hard to scale on

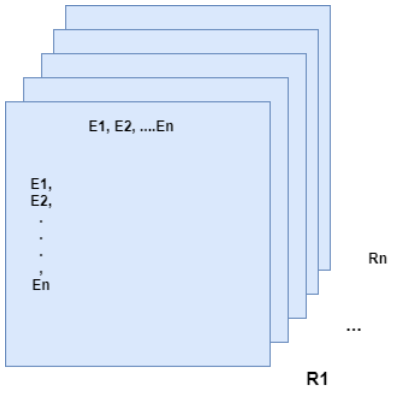


Figure 8. Illustration of RESCAL model

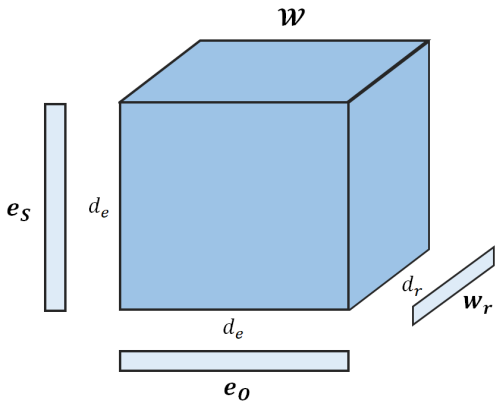


Figure 9. Idea of TuckER model

a large knowledge graph as well. Therefore, there are so many researches over RESCAL to find a solution.

DistMult [48] can be seen as a simpler version of RESCAL. With the same idea of RESCAL using the relation matrix, however, DistMult uses a diagonal matrix instead of a full matrix.

Both RESCAL and DistMult use a 3-dimension tensor for link prediction and use low-rank decomposition to find a representation of the entity or relation of the graph. For a given triplet (h, r, t), the score is calculated by using a multi-dots linear product to decide whether two entities have connected or not. However, the most noticeable drawback of this method is that it can't distinguish which is a head entity or a tail entity, so it can't handle antisymmetric relations. ComplEx [49] is proposed as a extension of DistMult to handle this problem. ComplEx suggests performing tensor decomposition on Complex space, and then the embedding is performed on this Complex space. The model uses the Hermitian product in the scoring function to express much more binary relations than normal space. Furthermore, using conjugate-transpose of the tail entity instead of the original tail helps ComplEx handle antisymmetric well.

Another method, SimplE [76], uses two distinct vectors, $\boldsymbol{h}_e$ and $\boldsymbol{h}_t$, which are head and tail entities, respectively. Furthermore, SimplE also uses two matrices for relations. One is the relation matrix $\mathbf{r}$, and another is the inverse matrix $\boldsymbol{r}^{-1}$ to perform inverse relations. Unfortunately, this approach has too many parameters to learn, which is likely to overfit.

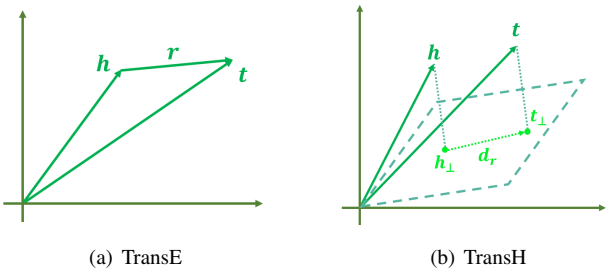
TuckER [77] is a tensor model which is based on Tucker decomposition. The core idea is to transform the tensor into a smaller core tensor - called $\mathbf{W}$ and several other vectors. TuckER used two smaller matrices represents $\mathbf{E} \in \mathbb{R}^{n_\mathbf{e} \times d_\mathbf{e}}$ and $\mathbf{R} \in \mathbb{R}^{n_\mathbf{r} \times d_\mathbf{r}}$, where $n_\mathbf{e}$ and $n_\mathbf{r}$ are the number of entities and relations, $d_\mathbf{e}$ and $d_\mathbf{r}$ are the dimensions of embedding. Different from other models, the dimension of entities and relations are independent. The core tensor is weight matrix $\mathbf{W}$, which can be seen as an interaction of relations and entities. However, the number of parameters is lower than other tensor models. TuckER model is shown in Fig. 9.

Although TuckER got a good performance on the link prediction task, all computations are linear. As we know, linear activation functions do not perform well compared to non-linear activation functions in a neural network. The classical model of a neural network in the link prediction is NTN (Neural Tensor Network) [64]. It has trouble when initiating the weight matrix. From the idea of TuckER and NTN, NepTuNe [78] is born to improve that weakness by combining these two models into a new neural network with parameters generated by TuckER model.

Tensor factorization models have a lower training time compared to others models. But their expressive power may not handle well to relations that are symmetric, antisymmetric. NepTuNe's idea of combining neural networks with tensors is a new approach to future researches. In addition, if we can apply the rule-based information to tensors, it can boost the tensor's model performance a lot.

b) Geometric

Currently, geometric approaches are the main branch of link prediction research. It was considered the earliest of knowledge representation learning. The idea of this group originates from the Word2Vec algorithm [79]. The Word2Vec encodes words or sentences to a matrix or vectors. Inspired by this encoding, TransE [46] was proposed as the first model in representing the knowledge graph to the new space. Specifically, to tackle the link prediction problem, the model transforms the entities into an embedding space and then, through the relation to translates the source entity to an entity that is expected nearest the actual target entity. The model learns the best



(a) TransE (b) TransH

Figure 10. Comparison between TransE and TransH model

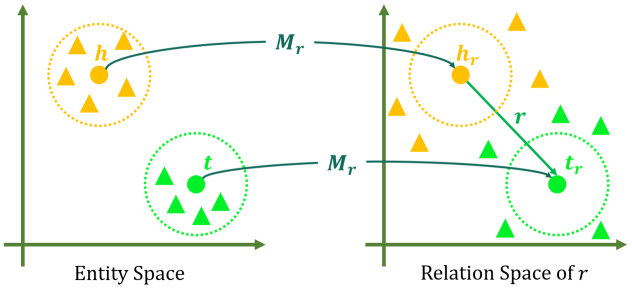


Figure 11. Principle of TransR model

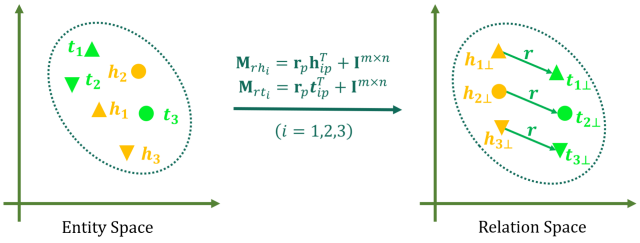


Figure 12. Illustration of TransD model

embedding through minimizes loss function of the total distance. The approaches in this group are divided into two branches: point-wise space and complex space.

The point-wise space projection can also be called TransE and its extension because the advent of TransE has led many researchers to switch to TransE-based improvements and extensions. As a result, there has been an explosion of the number of link prediction models based on this algorithm, which increases the performances significantly compared to previous models. Even in the case that TransE is not used directly for link prediction, it is also used by some algorithms, such as neural network-based models, to initialize input vectors due to its high level of graph generalization. Table VI gives a detailed comparison of a number of geometric models in terms of scoring functions, space complexity, and time complexity.

Point-wise space: TransE assumes that the tail entity is approximate the sum of the two head vectors and the relation by some measure. TransE is only suitable for 1-to-1 relations because they only use one vector to represent each relationship. That leads to TransE not being able to handle other types of relations such as 1-to-many, many-to-1, many-to-many. Fig. 10a presents TransE model.

To get around this, TransH [80] is proposed. It projects both head, and tail entities on relation space then compute the distance (shown in Fig. 10b). However, TransH still assumes that entities and relations represent the same space, which seems unsuitable. Therefore, TransR and CTransR models [81] dealt with that problems by transforming each entity of triplet (h, r, t) into the relational dimension through the transformation $\mathbf{M}_r$. Nevertheless, due to a large number of relations, when moving to a relation space, the representations are often inexact. Consequently, CTransR uses additional algorithms to group entities into clusters corresponding to each relation before translation transformation. An disadvantage of CTransR is that the transformation of entities and relations in large graphs is very computationally expensive. Moreover, by clustering, the model assumes that all entities in the cluster are the same type. That is not correct. For example, in triplet $(Obama, nationality, USA)$, $Obama$ and USA are different types of entities. As a result, TransD [82] is proposed to separate each entity h, r, t into two parts: the new representations on the new plane and their original vectors (Fig. 12). Fig. 11 gives clearly idea of this method.

A common weakness that Trans(E, H, R, D) models still suffer from is that they do not solve the imbalance problem of relations. For relations with too little training data, the performance is not good enough. Thus TransParse [83] was created as an improvement by using sparse matrices over low-dimensional matrices. In this model, the author proposes two versions of TransParse, TransParse (shared) and TransParse (separated). While TransParse (shared) uses only one sparse matrix for the first and last entity, TransParse (separated) applies two different sparse matrices.

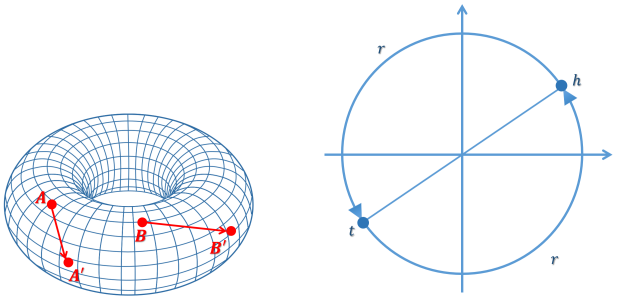
Complex space: Although the point-wise space models have made many achievements, simple spatial dimensions have not helped solve problems such as symmetric, asymmetrical, and transitive relationships. Therefore, we need to study more complex spaces with better representation.

An improvement from TransE to gradually introduce a space different from the point-wise space is TorusE [84]. TorusE performs the translation transformation on Torus Space. Although it is not a complex space yet, it is an initial step for the later algorithms to move to a new space different from the traditional space as in TransE.

TorusE's idea came from a TransE problem when using regularization. It warps the embeddings, so the scoring

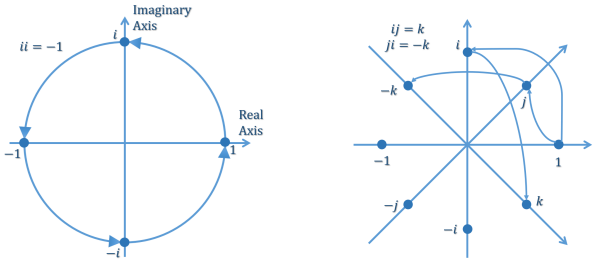
TABLE VI
COMPARISON OF GEOMETRIC MODELS IN TERMS OF SCORING FUNCTIONS, SPACE COMPLEXITY, AND TIME COMPLEXITY.

Model	Scoring function	Space complexity	Time complexity
TransE	$\|\mathbf{h}+\mathbf{r}-\mathbf{t}\|$	$O\left(n_e m + n_r n\right)\ (m=n)$	$O(N)$
TransH	$\left\|\left(\mathbf{h}-\mathbf{w}_r^T\mathbf{h}\mathbf{w}_r\right)+\mathbf{d}_r-\left(\mathbf{t}-\mathbf{w}_r^T\mathbf{t}\mathbf{w}_r\right)\right\|$	$O\left(n_e m + 2n_r n\right)\ (m=n)$	$O(2kN)$
TransR	$\|\mathbf{h}\mathbf{w}_r+\mathbf{r}-\mathbf{t}\mathbf{w}_r\|_2^2$	$O\left(n_e m + n_r\ (m+1)\ n\right)$	$O(2mnN)$
CTransR	$\|\mathbf{h}\mathbf{w}_r+\mathbf{r}-\mathbf{t}\mathbf{w}_r\|_2^2+\alpha\ \|\mathbf{r}_c-\mathbf{r}\|_2^2$	$O\left(n_e m + n_r\ (m+d)\ n\right)$	$O(2mnN)$
TransD	$-\ \|\mathbf{w}_{rh}\mathbf{h}+\mathbf{r}-\mathbf{w}_{rt}t\ \|_2^2$	$O\left(n_e m + n_r n\right)$	$O(2nN)$
TransParse(share)	$\|\mathbf{M}_r(\theta_h)\mathbf{h}+\mathbf{r}-\mathbf{M}_r(\theta_h)\mathbf{t}\|_{l\frac{1}{2}}^2$	$O\left(n_e m + n_r\ (1-\theta)\ (m+1)\ n\right)$	$O\left(2\ (1-\theta)\ mnN\right)$
TransParse(separate)	$\left\|\mathbf{M}_r^h(\theta_h^h)\mathbf{h}+\mathbf{r}-\mathbf{M}_r^t(\theta_h^t)\mathbf{t}\right\|_{l\frac{1}{2}}^2$	$O\left(n_e m + 2n_r\ (1-\theta)\ (m+1)\ n\right)$	$O\left(2\ (1-\theta)\ mnN\right)$
RotatE	$-\|\mathbf{h}\circ\mathbf{r}-\mathbf{t}\|$	$O(n_e m + n_r n)$ (m=n)	$O(N)$
TorusE	$min_{(\mathbf{x},\mathbf{y})\in([\mathbf{h}]+[\mathbf{r}])\times[\mathbf{t}]}\ \|\mathbf{x}-\mathbf{y}\|_i$	$O\left(n\right)$	$O\left(N\right)$
QuatE	$\mathbf{h}\otimes\mathbf{r}^{\diamond}.\mathbf{t}$		$O(N)$



(a) TorusE (b) RotatE

Figure 13. Main idea of TorusE and RotatE models



(a) RotatE (b) QuatE

Figure 14. Compared between RotatE and QuatE

function is affected when operating in the hypersphere. Furthermore, TransE's embedding is not equipped with operations to handle vector *h + R'* or **h + R'** out of space, where *R* is regularization. Hence, TorusE is proposed by using a manifold for embedding space and remove regularization on its scoring function. Fig. 13a illustrates the Torus model.

Although TorusE paves a new way of Translation-based model, the manifold's theories are difficult to understand. A significantly improved model can be seen as RotatE [61] . This model defines a relation as a rotation of head entities to tail entities. This representation is proved that it can handles such relations as symmetric, antisymmetric, composed (Fig. 13b).

Instead of rotation transformation on one plane as RotatE, QuatE [85] represents relations on quaternion embedding. Specifically, relations are transformed in this space through the Hamiltonian cycle and quaternion product (Fig. 14). In this way, QuatE can perform rotation on two planes which they hope have more expressive power.

RotatE believes that combining relations to infer a new relation is commutative. In other words, when changing the position of relation components, the new relation remains the same. This is considered inappropriate because it decreases the expressiveness of the proposition. For example, given a combined clause $r_1(x,y) \wedge r_2(y,z) \Rightarrow r_3(x,z)$, if r_1 is the "is-father-of" clause and r_2 is the "is-mother-of" clause, then r_3 is the "is-grandfather-of" clause. Otherwise, r_3 is the "is-grandmother-of" clause.

Furthermore, RotatE assumes that the axis of rotation must be orthogonal to entities in 2D embedding space. As a result, it limits the ability to represent relations between entities from the original graph. Hence, QuatE has improved this using much more complicated space than RotatE's. However, the way to transform in the dimension is not easy to understand and intuitive. Therefore, DensE [86] was proposed to solve these problems by performing knowledge graphs on 3-dimensional space with two additional components: vector $\vec{v}$ to define the rotation direction and angle ϕ to mean the angle between the axis and the direction of rotation. DensE is shown on Fig. 15.

To sum up, these models represent a graph in a latent space where we can apply geometric knowledge to solve. However, even though they get achievements, they still can not take full advantage of latent information like local structure.

c) Neural Network

In addition to matrix decomposition and geometric transformation, we can use neural networks to solve the link

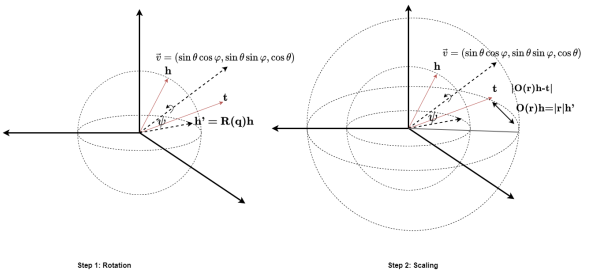


Figure 15. Illustration of DensE model

prediction. Since the input to the neural network is a matrix, the graph needs to be converted to embedding space first. However, depending on the embedding method, information can be lost more or less. Therefore, to keep the most information of the graph or create a form from which latent "knowledge" can be extracted, the neural network-based models use some additional functions such as attention mechanisms. From that, we divide the neural network models into two groups: using additional information and without using additional information. Additional information may be about the path, the attention mechanism, the architecture of the neighboring nodes.

Without addition information: In these models, the architecture is quite simple and is mainly based on the traditional CNN network. Typical models include ConvE [69], ConvR [87] or ConvKB [88].

Generally, the approaches use one or more CNN layers in which each layer performs convolution on the input for feature extraction, then feed into a dense network to predict the final output. However, depending on the model type, there are different ways to transform the input. For example, ConvE reshape head and relation vectors, then combine them into a matrix $[h, r]$. In other ways, ConvKB concatenates all three vectors h, r, t into one matrix with $d \times 3$ dimension. ConvR converts head and relation from a d-dimensional vector to a matrix, but these two matrices are not the same. After that, it performs convolution with a filter which is a relation matrix onto the head matrix. It is clear that the matrix representing the head vectors corresponds to an input image in the CNN network used for image processing and the relation matrix corresponding to the filter and convolution on this network. After successful feature extraction, all three matrices are fed into a dense d-dimensional network to get the final output. However, while ConvE and ConvR calculate the difference between the predicted result and the correct output based on the dot product, ConvKB only indicates whether it is right or not. The overall structure of these models is presented in Fig. 16.

The weakness of models ConvE, ConvR, and ConvKB

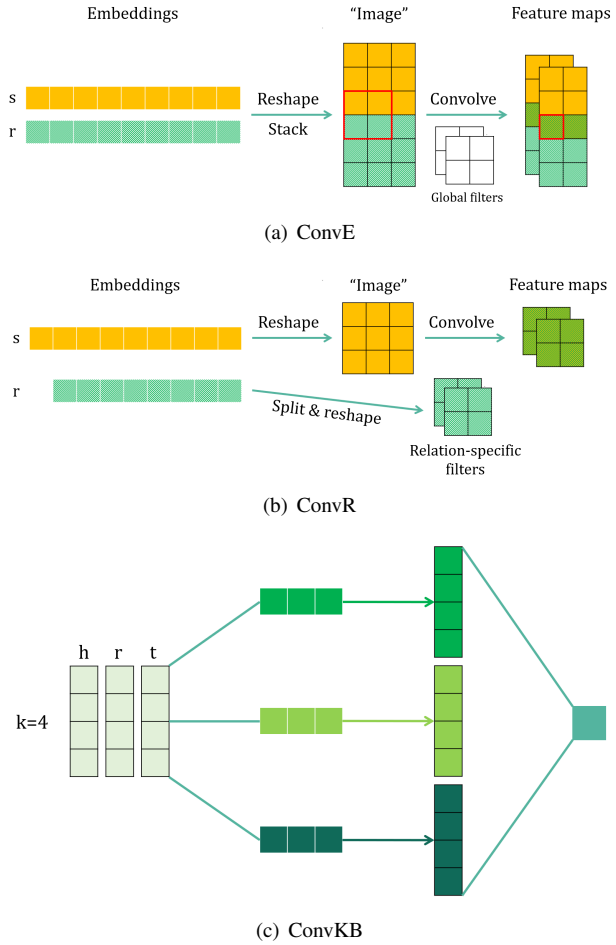


Figure 16. Overall structure of ConvE, ConvR, and ConvKB

is that they all generate parameters randomly. This leads to taking a long time to converge. In addition, they have a fairly similar representation. Firstly, all models learn to embed entities and relations, then combine them into a matrix and perform a series of linear transformations on this new matrix. That reduces the expressiveness of the original graph. Hence George Stoica proposed CoPER [89] - a solution transforming input and generating parameters for the model based on relations, similar to NLP context generation.

In CoPER, instead of using both transformations for both source entities and relations, the authors only apply them to source entities and use relations to generate parameters for the filter, then use those new parameters for training (Fig. 17). This helps the model achieve better performance.

Another solution of the interaction between entities and relations applied to the ConvE model is InteractE model [90]. Instead of using relation to generate filter parameters, the model increases the "interaction" through 3 steps: feature permutation, checkered feature reshaping, and circular convolution. They believe that using permutation of input

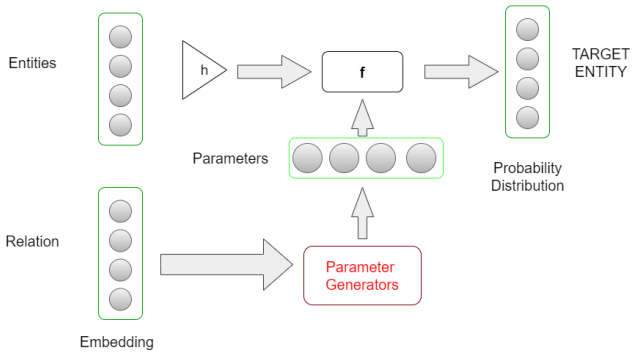


Figure 17. Idea of CoPER model

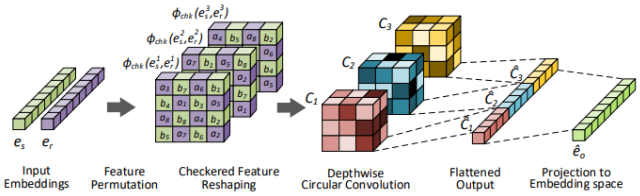


Figure 18. Overview of InteractE [90]

and circular convolution helps learn more latent information between entities. It is shown in Fig. 18.

Using addition information: As mentioned, the neural network requires input as a matrix, so the graph needs to be embedded before feed into the neural network. This can lead to the risk that the knowledge from the original graph is no longer fully preserved before being included in the learning model. Consequently, many other architectures are designed to add additional information to the model, such as attention, BERT, or GAN. KBGAT [91] (attention), LSA-GAT [92] (attention), RAGAT[93] (attention), KG-BERT [94] (BERT), StAR [95] (BERT), KBGAN [96] (GAN) are typical neural network-based models with extra information.

The attention mechanism used in the models is quite similar to that in NLP. For each triple in the graph, after concatenating head, tail, and relation vectors into one, the attention value is calculated and multiplying with the parameter matrix **W** to form a new embedding vector. This vector is passed through the softmax function to find out which vertex has the greatest influence on the head-relation pair under consideration. Fig. 19 is an example of attention mechanism.

In the process of calculating and updating attention, to avoid the situation of isolated vertices (attention values is 0) and exploit more semantic layers that can be inferred from the relations, LSA-GAT considers more types of neighboring structures. With various structures, the author supposes that different layers of hidden semantics are inferred. Here, the author focuses mainly on three primary structural forms:

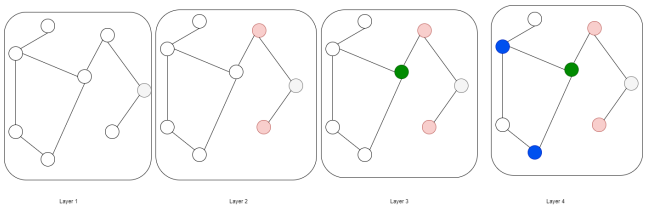


Figure 19. An example of attention mechanism

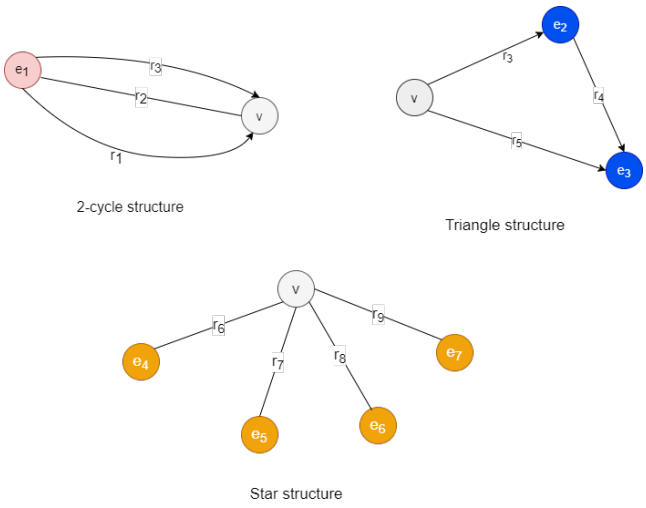


Figure 20. An example of neighboring structures in LSA-GAT

triangle, star, and 2-cycle. Fig. 20 is example of these structures. This approach improves the performance better. The overview architecture of the model is illustrated in Fig. 21.

Along with the idea of the neighboring structures, which LSA-GAT used to learn entity embedding, RAGAT applies the "relation message function" to initialize and support information extraction from neighboring nodes for each different relation. The relation message functions are inspired by previous models. Therefore, the model combines them to create the final embedding for the decoder. Fig. 22 shows

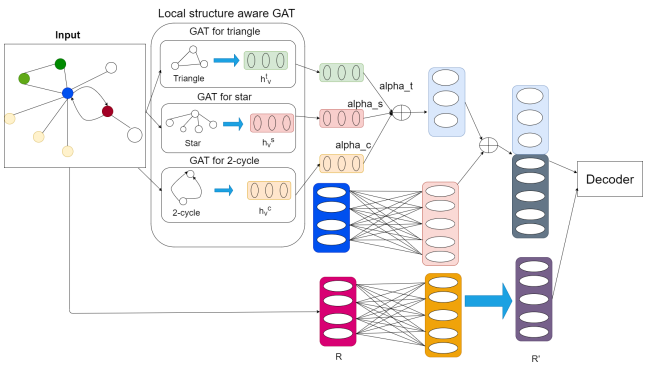


Figure 21. Overall architecture of LSA-GAT model

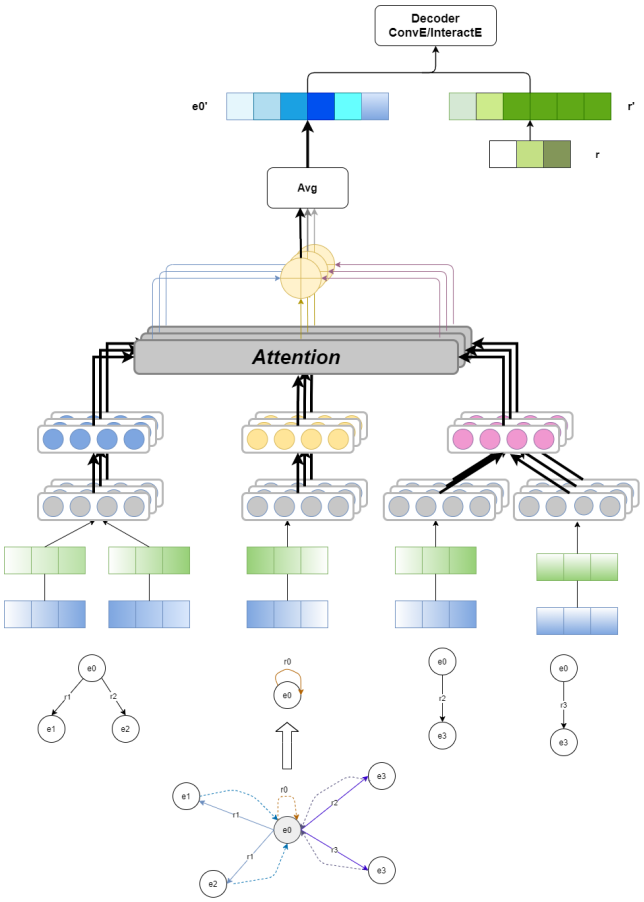


Figure 22. Overall architecture of RAGAT model

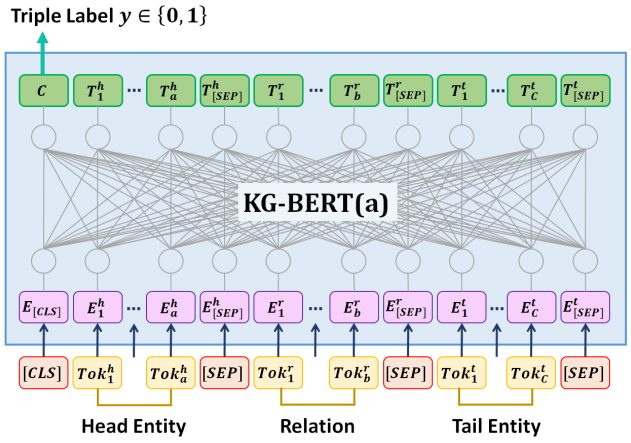


Figure 23. Illustration of KG-BERT model

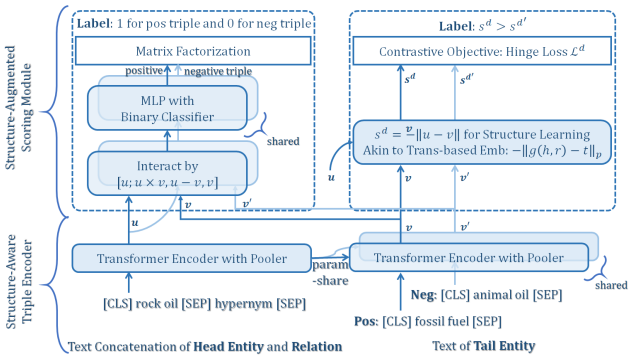


Figure 24. StAR model

this method's idea.

Another information that can use for embedding is semantic information from the BERT model. BERT's core idea is to transform a triple (h, r, t) into a sentence by treating the entity or relation as the subject/predicate of a sentence and use it as input to BERT. In a document, sentences can be simple or complex. Therefore, to better train the model, the authors proposed a method to combine multiple triples to create a form like complex sentences (Fig. 23).

KG-BERT stores triplets as contextual sentences and applies the BERT model to make future predictions. However, this representation still has some flaws, such as the high computational cost for converting triples to sentences as well as inferencing from these sentences. Furthermore, with the lack of background knowledge to serve this contextual encoding, KG-BERT cannot promote the full power of BERT. Therefore, Wang et al. [95] included contextual encoding in graph embedding by separating the triplet into two parts, head - relation and tail, like translation-based models TransE, RotatE, and encoding these two parts according to the Siamese style. This approach uses both graph structure information and context information to help the model effectively exploit facts that are not present in the graph. The overall structure of this model is depicted in Fig. 24.

In general, deep learning models to predict links on knowledge graphs have achieved good results. However, these models depend on embedding to predict, so they also lose information from the original graph. Furthermore, they use a large number of parameters to train. In the future, we hope there is a way to input the graph into a neural network without using embedding.

2. Rule Based Models

Graph embedding's core idea is to convert the graph to vectors and predict based on these vectors. This approach has some drawbacks. Firstly, converting a graph to vectors loses a lot of information of meaning. Later, the cost of converting is extremely high. Finally, vectors can't express the connection between triplets of the graph. To keep information as much as possible, another approach is logical-rule learning, such as the Horn clause.

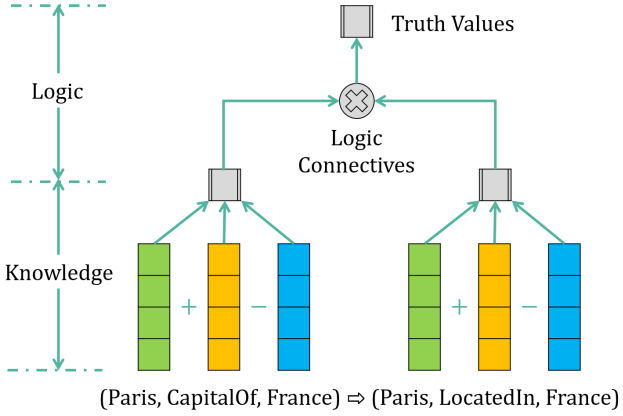


Figure 25. KALE model

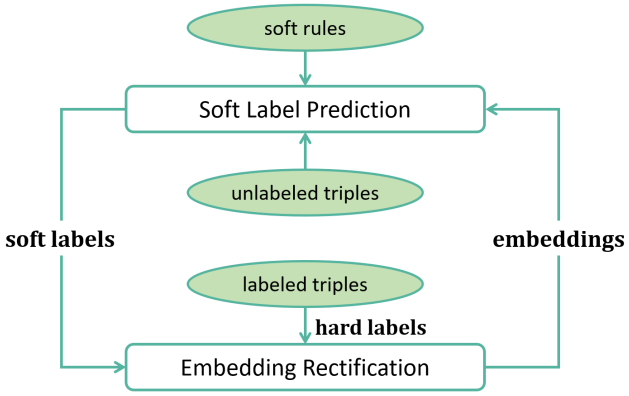


Figure 26. Structure of RUGE model

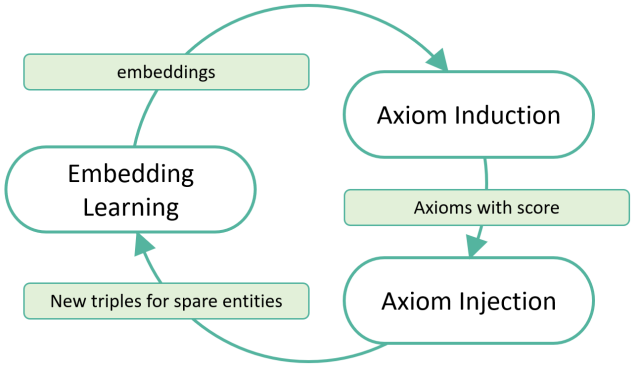


Figure 27. The architecture of IterE model

A horn clause is defined as $head \leftarrow body$. Head is a fact of objects while the body is a fact or multiple facts combined.

AnyBURL [97] is the classical model for this group. The model starts by sampling the data and using a measure to determine whether increase path length or not for generating a new rule. Besides, it also decides whether this rule is good enough to add or not. The algorithm stops when the ratio of generated rules/total number of rules is greater than saturation.

Recently, besides the models that only generate rules and use them for link prediction, many models have started combining knowledge from rules into embedding through joint learning or iterative learning to gather more information of the graph. One example is KALE [98]. It makes the use of rules in conjunction with embedding. The input includes the entities set E, the relations set R, and the rule set L which is pre-trained and auto-updated. The set L is encoded for processing and evaluation. The set E and R relate to embedding and are used to store the hidden semantic information of the entities and relations. After learning, model use truth value to decide whether the rule is good to add to pre-trained rule-based or not. Fig. 25 represents KALE model.

While KALE learns by joint learning, RUGE [99] uses iterative learning to learn information from both rules and embedding. RUGE predicts unlabeled entities based on the current embedding and previously learned ground rules, and then the new labels are fed to the current embedding to update it. RUGE's core idea is shown in Fig. 26.

Rules are precise and can guess implicit links while using embedding helps preserve information and semantics in the graph. Combining them can produce many beneficial results, but they also have their drawbacks. Sparse entities (less mentioned in the graph) become difficult to predict due to lack of information leading to weaker encoding. In addition, the number of rules generated is enormous, so it consumes a lot of search space when making decisions and restructuring rules. For example, if there are 10 relations and 10 entities, the number of rules generated is $(10^2)^3 * 10^3 = 10^9$. The result is tremendous because the number of rules increases exponentially with the number of relations. That isn't easy when adapting to large datasets. Therefore, the IterE model [100] was born to combine the advantages of learning embedding and rules while minimizing the disadvantages of both methods through embedding learning, axiom induction, and axiom injection (Fig. 27).

Another model which combines both rule and embedding is LogicENN model [101]. It is a neural network which is able to encode the rule to the network. It is shown on Fig. 28. This helps detect whether the rule is good or not. Unlike other neural network models which inputs are both entities and relations, LogicENN's inputs are only entities. Furthermore, to encode rules to the neural network, the author demonstrates how to transform common rules into weighted formulas and feed them to the network.

TABLE VII
SUMMARY OF SOME REINFORCEMENT LEARNING MODELS FOR LINK PREDICTION, ACCORDING TO [21].

Method	State	Action	Reward	Policy Network
DeepPath	$(e_t, e_q - e_t)$	r	Global 1 if $e_t = e_q$	Fully connected network
MINERVA	(e_t, e_s, r_q, e_q)	e_t, r, v	$e_t = e_q$	$h_t = LSTM\,(h_{t-1}, [a_{t-1}, o_t])$
Multi-Hop	$(e_t, (e_s, r_q))$	$(r\prime, e\prime) \mid (e_t, r\prime, e\prime) \in \mathbf{G}$	$\gamma + (1-\gamma)\, f_{rq}\,(e_s, e_t)$	$h_t = LSTM\,(h_{t-1}, [a_{t-1}])$

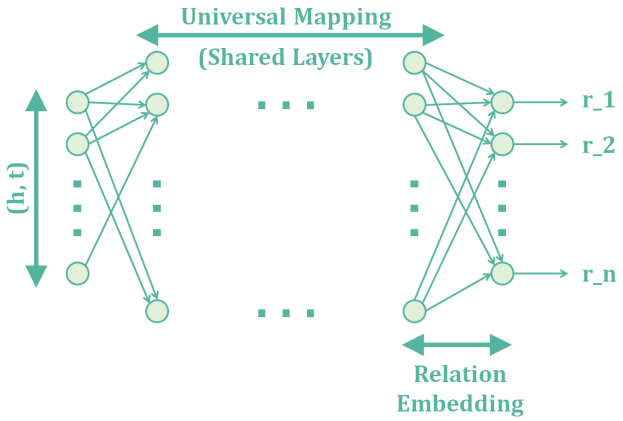


Figure 28. Idea of LogicENN model

Apply rules on link prediction tasks helps models keep information without loss. However, the space for rules is still enormous. This makes prediction time slowly. We hope that there are studies on how to apply data structures to reduce the memory space.

3. RL-Path Based Models

Reinforcement learning (RL) was introduced in multihop reasoning. Reinforcement learning consists of two main parts. The first part is to choose the environment E that can represent the action as a repetitive sequence between the agent and the graph. This environment is called Markov. An action $< S, A, P, RP >$ represents the Markov decision, where S is the chosen space, A is the set of actions that the agent can perform, $P(S_{t+1} = s_0 | S_t = s, A_t = a)$ is the state transition matrix, and $R(s, a)$ is the reward for each action. The second part is training the Policy network to predict the output of actions chosen randomly by the agent. This policy network is updated by network update methods such as gradient descent. Table VII summaries some characteristics of reinforcement learning models in link prediction.

DeepPath [102] is a typical model for applying RL. First, DeepPath embeds the graph by the TransE approach to keep the graph structure and semantic information. After that, the author defines a state space: $\mathbf{s}_t = (\mathbf{e}_t, \mathbf{e}_{target} - \mathbf{e}_t)$, where $\mathbf{e}_t$ is the current state and $\mathbf{e}_{target}$ is the target state. The reward is used through three aspects:

1) Global accuracy: evaluate whether the action was successful or not.

$$r_{GLOBAL} = \begin{cases} +1, \text{ if reach target} \\ -1, \text{ otherwise} \end{cases} \tag{9}$$

2) Path efficiency: the shorter the path is, the better effect it has.

$$r_{EFFICIENCY} = \frac{1}{length(p)} \tag{10}$$

where path p is sequence of relations $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_n$

3) Path diversity: evaluate multiple disjoint paths between any source-destination pair. The agent takes advantage of paths that are learned previously to form a new path. With this approach, the new path maybe has redundant information. This reduces the effectiveness of the model.

$$r_{DIVERSITY} = -\frac{1}{F} \sum_{i=1}^{|F|} cos(\mathbf{p}, \mathbf{p_i}) \tag{11}$$

where $\mathbf{p} = \sum_{i=1}^{n} \mathbf{r_i}$ represents the path embedding for relation sequence $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_n$ and $\mathbf{p}_i$ is path embedding at state i

MINERVA [103] is also a model that applies RL. But this model changes the state space. On the contrary to DeepWalk, MINERVA does not require pre-trained embedding. It represents the space as $E \times E \times R \times E$, which is possible combinations of sets $E(entities)$ and $R(relations)$. In addition, it also introduces an observation: state function to assess for action concerning that agent finds a random edge to add. Updating the state of the new edge selected by the agent is called action transition. Instead of using three aspects like DeepWalk to evaluate the reward, MINERVA uses binary classification to assess whether this path reaches the destination or not. The removal of two others aspects is expected to make full use of the graph's information for other actions.

In conclusion, RL-Path based models use state of the art of reinforcement learning on the knowledge graph. However, they are not effective as other models due to the lack of expressive power and not taking advantage of latent information. These models get better performance if they can do that.

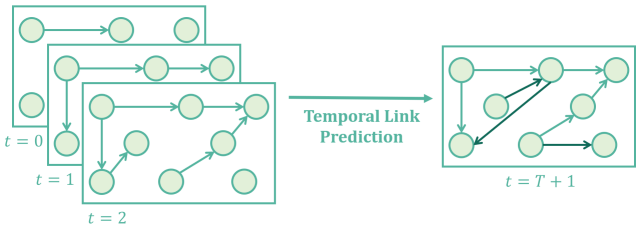


Figure 29. Illustration for link prediction in temporal graph [39]

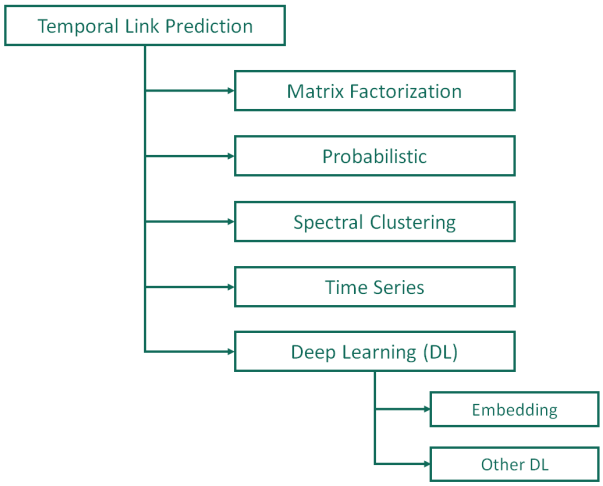


Figure 30. Methods used in temporal graphs for link prediction, according to [39]

VII. Link Prediction on Temporal Graph

The link prediction problem on knowledge graphs mainly focuses on static graphs, which means entities and relations still exist over time. However, dynamic graphs play a significant role in human life, such as social networks, molecular biology networks, internet. Therefore, the link prediction problem becomes more critical. However, the study of dynamic graphs is still not popular compared to static graphs. The cause may originate from its complexity. While a static graph only needs to build a model to help predict at a time, on a temporal graph, in addition to predictability, the model must also be able to predict whether nodes, edges is exist at the next time $t+1$ or not. Fig. 29 illustrates the temporal link prediction definition.

Fig. 30 is an overview of the methods applied in the link prediction for temporal graphs. At a glance, we can see that some methods are similar to those in static graphs, such as matrix factorization, neural networks. In addition, some new methods are available in temporal graphs, such as time series, clustering.

We summarize some characteristics of the groups for temporal link prediction before detailing the algorithms in each group. Firstly, matrix factorization transforms a graph

Figure 31. Principle of temporal matrix factorization model

into a matrix or tensor then applies methods to decompose the matrix into smaller matrices or vectors. Based on that, we can derive properties in the neighborhood of nodes or the entire graph. Although this method can be feasible on static graphs, in dynamic graphs, applying it to each slice graph at time t and combining them take a lot of memory space and time. As a result, matrix factorization may not be applied well on temporal graphs, huge and complex graphs. The next two methods, probabilistic and clustering, can predict how the graph is change, but their costs are enormous, so they can only be applied on small graphs. The time series method can be considered as the best predictor of the time-varying graph because this method is very effective in time-varying datasets such as stock prices, COVID-19 deaths. Still, the time series focuses on graph transformation too much and ignores important things such as the relationship of a node and its neighbors. It isn't easy to generalize about the local topology and total. Deep learning methods are considered better methods than other groups of methods in dynamic graphs because of their ability to extract and generalize graphs over time. However, the computational cost and training time for this model group is extremely large and time-consuming. It is clear that the link prediction on dynamic graphs requires a lot of effort to improve accuracy and reduce complexity so that the model can run on large-scale graphs.

1. Matrix Factorization

The matrix factorization transforms the graph to a matrix or tensor, then decomposes this matrix/tensor into smaller components that are used to extract features. This approach on the temporal graph is the same as on a static graph (Fig. 31). Almost all models of this group have the following typical steps:

1) Convert the graph to matrix form (adjacent matrix).
2) Decompose this matrix using the decomposition algorithm to extract the necessary features.
3) Add weights that represent learning the changes of the graph over time.
4) Use algorithm on new embedding.

Raymond [104] proposed a method for the link prediction problem, specifically in the node-prediction problem using the semi-supervised learning method. This model aims to

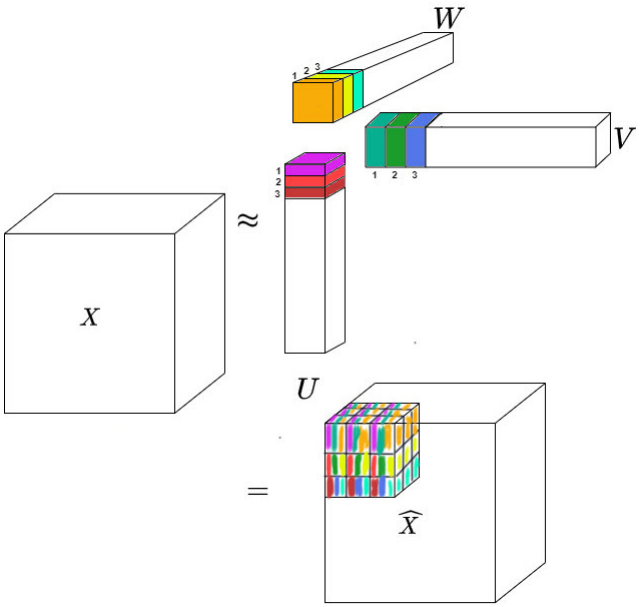


Figure 32. CANDECOMP/PARAFAC (CP) tensor decomposition process

optimize the Chusky and the eigen decomposition applied on a static graph and lays it on temporal graphs. They define the matrix F^* with each element $F(i, j)$ has value 1 if these two vertices have a connection and vice versa. The previous link propagation methods predict unlabeled nodes based on a principle: "The more similar two pairs of nodes are, the higher the relationship forms between them." For temporal graphs, he used an approximation method called incomplete Cholesky decomposition with the aim to reduce the cost of computation.

If Raymond uses vectors to represent change over time, Truncated SVD [105] transforms the entire whole matrix into a 3-dimensional $Z(i, j, t)$ tensor with:

$$Z(i,j,t) = \begin{cases} 1, \text{if node } i \text{ connects to node } j \text{ at time } t \\ 0, \text{otherwise} \end{cases} \quad (12)$$

With this representation, it is not possible to perform the transformation of the nodes over time. Furthermore, the computational cost is enormous. Therefore, the author proposes to use CANDECOMP/PARAFAC (CP) tensor decomposition [106–108]. It decomposes the matrix into a product of lower component vectors. CP is used to decompose the matrix to preserve the most information of the graph while preserving the 3-dimensional architecture as the original matrix Z. Fig. 32 describes this process.

2. Probabilistic

Models of the probabilistic group tend to use maximum likelihood estimation or Bayesian estimation methods to calculate the probability of link formation.

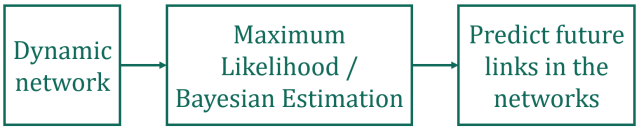


Figure 33. Main steps of probabilistic models

Fig. 33 shows main steps of models in this group:

1) The graph is transformed into snapshots at time t.
2) Apply maximum likelihood or Bayesian estimation to these substates to make predictions about which connection appears in the future.

Extended temporal exponential random graph model (etERGM) [109] uses probability to predict node and link changes on historical data. It uses the Markov assumption: "Each future matrix is independent to its previous state." Instead of using only one joint model, the author designs two different models for training. They both support each other in predicting node and link changes. When predicting, the author uses results from both models to give a final prediction.

Nevertheless, etERGM uses two additional parameters for training, which leads to memory-consuming. Therefore, some approaches without training parameters are published. One of them is Non-parametric Link Prediction in Dynamic Networks [110]. This model helps to exploit more local information from the graph. In addition, a problem of previous models is that new nodes/relations are assumed to form at the same time. However, this is not correct because a new node/link appears or disappears depending on the properties of each element. Hence, the author proposes a method to help overcome these two problems by using the topology structure around the nodes-links instead of directly predicting between the nodes-links.

In addition, to use the graph structure to predict the possibility of a link, the popularity-based structural perturbation method (PBSPM) [111] assumes that there are some other implicit factors like the tendency to create new nodes. However, this approach meets some problems. Firstly, these implicit factors sometimes cannot be found. Secondly, we can't prove whether they are reliable or not. In this model, besides graph structure, the new connection is formed based on the popularity of the node. The popularity of a node is defined as the number of links it has at a specific time. The author defines these influence factors as $x_k = [x_{k,1}, x_{k,2}, ..., x_{k,n}]^T$, where $x_{k,i}$ representing the influence of node i on the composition k.

The author defines the change in popularity of node i over the time period t and $t + 1$ as:

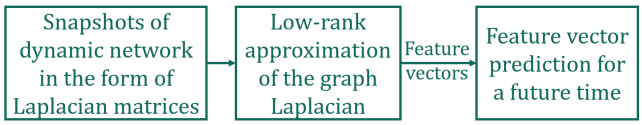


Figure 34. Main steps of spectral clustering model

$$\Delta k_i(t,T) = k_i(t+T) - k_i(t) \tag{13}$$

However, the measure only assesses the influence over time. Therefore, this formula is unsuitable because a node with multiple changes is not likely to be popular or even become isolated the next time. Moreover, this formula does not clearly state the change in popularity between two neighboring nodes in the time interval t. As a result, they define $p_{fresher}$ for the fresh set and p_{older} for the old set. The fresh set is the set of new edges generated during the specific period and vice versa for the old set.

3. Spectral Clustering

Currently, there are not many studies focusing on the direction of spectral graph theory. It aims to study the properties of graphs with characteristic functions, eigenvalues, and eigenfunctions of matrices related to the graph. For example, they are eigenvector, matrix adjacency, and Laplace matrix. However, this direction still has a remarkable achievement that matrix decomposition methods do not do well on large graphs. Fig. 34 shows the basic steps of the model group in this method:

1) Represent the graph at the snapshot time t as a Laplace matrix.
2) Perform low-rank approximation and graph spectrum for feature extraction.
3) Use time series or linear models to draw feature vectors
4) Use these vectors to predict

This method has been applied very effectively in the network analysis problem. The model proposed by Wu [112] applies low-rank eigendecomposition for each snapshot time t of the graph and Finite Impulse Response filter to learn the transformation of the graph.

4. Time Series

These models apply methods used in predicting changes over time, such as predicting stock prices and COVID-19 deaths to predict changes in graph links. Although the models are different from calculation or graph transformation, they go through the following basic steps:

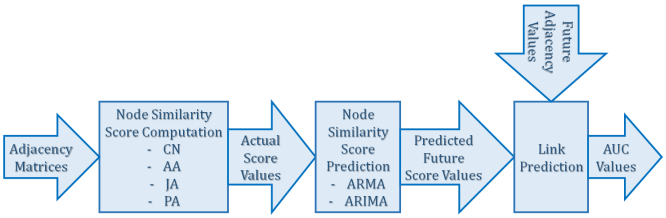


Figure 35. ARIMA's approach for link prediction on temporal graph

1) The graph is saved as a matrix corresponding to the snapshot time t.
2) For each pair of nodes, initialize a time series based on the measure of similarity between this pair of nodes.
3) Apply the time series model to predict the time series score.
4) Use the score to make predictions.

The integrated time series model (ITM) [53] uses additional information to the time series. They are community information and node centrality. Given each snapshot of the graph at time t, it clusters to create communities. Besides, it also uses node centrality to represent the relation of the central node to its neighbors through eigenvalue. When we get both centrality embedding and community embedding, we combine them to get a final prediction.

The classical model of time-series, ARIMA [54], is also applied to link prediction along with node centrality measures. Firstly, it calculates the similarity value between pairs of nodes. After that, the model makes the prediction based on this similarity. We have a deep look of ARIMA through Fig. 35.

Until now, the models use the same coefficient for each time t. In contrast, in CALA [55], every time state and the scores are assigned coefficients to find out which time state and scores have the lowest prediction error.

At the beginning, the original graph is transformed into an $n \times n$ dimension matrix representing the graph at time t and other matrices representing the similarity of nodes using different measures: common neighbor, Jaccard, Katz, Salton [113], Preferential Attachment, Adamic-Adar, and LP. Next, these similarity matrices are considered as feature vectors and are fed into the model.

5. Deep Learning

Deep learning is widely applied and achieved many achievements in other fields such as computer vision, NLP, and temporal graphs. In this group, we divide into two groups: embedding and other models.

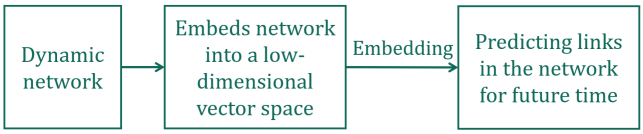


Figure 36. Overall steps for an embedding model

Figure 37. A triad example

a) Embedding model

In a temporal graph, the embedding firstly transforms the graph into a smaller dimension but still retains the most information of the graph. After that, neural networks are used to learn weight sets for link prediction. In general, this approach is the same as deep learning models in the static graph. The following steps summarize the main points in this models (Fig. 36):

1) Transform input as snapshot sequences to keep information "dynamic."
2) Embed these snapshots to a low-dimension latent space. After that, deep learning models are used to predict links in the subsequent periods.
3) Predict on the original graph based on the prediction results on these embeddings.

The previous models assumed that edges do not change significantly over the new dimension when adding regularization. Instead of that, DynamidTriad [114] uses triads to represent the changing state of edges over time. Model is trained by sampling method and updated by SGD.

Given an open set (v_i, v_j, v_k) at time t, where v_i is not connected to v_j but is also connected to v_k. This is called triad (Fig. 37).

Instead of taking snapshots as inputs which can be considered discrete information, CTDNE (Continuous-Time Dynamic Network Embeddings) [115] represents graph changes over time in very small intervals: per second or milliseconds. This is a better approximation for the transformation of the graph. The author defines the graph as $G = (V, E_t, T)$, where V is the set of vertices, E_t is the set of edges of all time, and T is a function that maps vertices to edges in the corresponding time interval. The main idea of the CTDNE is to use a similar version to the random approach on a static graph. The algorithm starts from a randomly selected node by the normal distribution

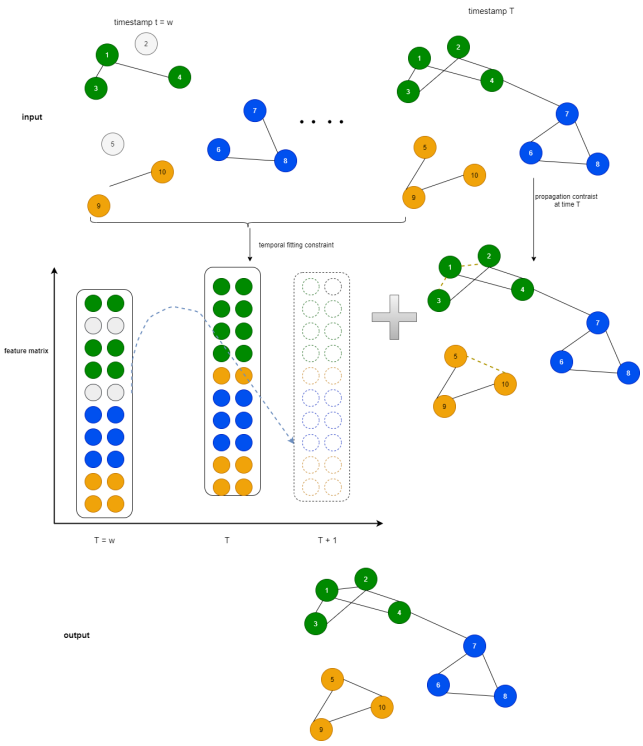


Figure 38. Illustration of LIST model

or previously trained parameters. Then, it tends to move to nodes closest to it at the time t.

When moving over time, the path is likely to pass through invalid vertices at the time t. To handle this issue, the authors suggest adding two parameters: min length ω and max length $\mathbf{L}$ and called context windows. They are used to limit the number of valid steps at that time.

Dynamic2Vec [116] introduces a new way of representing features in temporal graphs. For each node pair (u, v) in the snapshot network, the author defines the embedding as α^{uv}, where u, v are node-pair and α is based on linear transformation. They also introduce the notations compression and reconstruction. If other models tend to transform each pair of vertices with a lower cost, Dynamic2Vec chooses to transform when reconstruction costs the least.

b) Other Models

LIST [117] combines the use of neural networks and matrix decomposition for feature extraction. They use decomposition to learn the graph's changes over time and neural networks to determine whether the link appears or disappears. To combine these two models, the author uses joint optimization to the loss function of these two submodels, which is shown in Fig. 38.

VIII. Evaluation

In this section, we compare models based on the results of running experiments on standard datasets. Based on that, we make analyzes and comments to show the dominant models. However, in general, almost no model runs well on all datasets. The main reason is that the characteristics of each dataset are different, and models are designed to solve problems well on one dataset but are not suitable for solving other datasets.

1. Datasets

The datasets that are often used to evaluate models are introduced in Section IV. In that section, we present the characteristics of each dataset, including properties of the relations and the size of graphs. We found that some datasets are used a lot by the models, but others are still less evaluated because they are newly formed and promote the strengths of the proposed model.

2. Experimental Environments

KG link prediction models are usually trained on machines with CPU 4 cores 2.0 GHz, running Linux distros. An important device is the GPU to speed up the training time. Early, the authors often used the C language for implementation, but later, Python was a better choice due to its easy programming as well as great libraries. Depending on the model, training time ranges from 8 hours to 15 hours for graphs of size 15,000 verities and 1,000 relations on a 12GB VRAM GPU. The classic algorithms are all open source. Several pages on GitHub collect open-source models such as Xinguoxia's[1], Shulin Cao team's[2], and PaperWithCode[3]. This is a good advantage point so that researchers can be rerun and easily compared across the new environment. Models, especially based on deep learning, are implemented mainly on PyTorch, TensorFlow and MXNet frameworks. Some well-known libraries like PyTorch Geometric [118] and DGL [119].

3. Experimental Parameters

a) Common Parameters in Static Graphs

In table VIII, we summarize the parameters extracted from the papers of the important models. These parameters include B is batch size, Ep is training epochs, D_e and D_r are entities embedding and relation embedding dimension, BC is batch count, LR is learning rate, reg is regularization method, λ is lambda for regularization, drop: dropout value (in: in input, out: in output, h_i: in i^{th} hidden layer, feature: in feature), Kernel is kernel size, K is number of filter and N is negative samples per training facts(default: 1).

In general, the values of the parameter do not differ much in each training dataset. However, some well-equipped authors often try some higher value for the best model performance. Another change is in the normalization function to avoid overfitting as well as help the algorithm achieve more optimal results. The extra parameters come from the design of the model. The more parameters there are, the more complicated the value selection is and the harder it gives good explanations.

b) Common Parameters in Temporal Graphs

Because dynamic graph link prediction has received much attention recently, the models have not yet formed clear main groups. Algorithms often also vary widely and run on different data sets. This is the challenge for us to determine the common parameters between the models. Therefore, we only list here the important parameters in the three models that can be clearly demonstrated during their experiments:

- TruncatedSVD: dimension size is in range 10 to 100.
- TCOP: path length is 5 and central neighboring sets is 6
- CALA-LP: learning rate lr is 0.05 and action coefficient parameter is 0.3

4. Evaluating Models on Static Graphs

For the static graph, we collect the results from the original papers of the models as well as from the survey articles that have been re-experimented. Then, we compare the models on four standard datasets, FB15k, WN18, FB15k-237, WN18RR, using the metrics we described in Section V.

We also list all version of each model in both table with:

1) Result of these following models: DistMult, ComplEx, SimplE, TransE, RotatE, ConvE, ConvR, ConvKB are taken from Rossie et al. [44].
2) The results of TuckER on two datasets FB15k and WN18 is taken from from Rossie et al as well.
3) For QuatE model, we include three versions. They are QuatE(1): without type constraints, QuatE(2): N3 regularization and reciprocal learning, and QuatE(3): with type constraints. All these versions are described in [85].
4) For stAR model, there are three versions. They are stAR original, stAR(ensemble) with two parameters $k \leftarrow \infty$, $\alpha \leftarrow 0.5$ and stAR(self-adp) with two parameters $k \leftarrow 1000$, α is learnable, which are shown in [95].

[1]https://github.com/xinguoxia/KGE
[2]https://github.com/thunlp/KRLPapers
[3]https://paperswithcode.com/task/link-prediction

TABLE VIII
MAIN HYPERPARAMETERS USED TO TRAIN THE MODELS

	FB15k	**WN18**	**FB15k-237**	**WN18RR**
DistMult	BC:50; Ep:4000; D:200; Loss:self_adversarial; γ:1; LR:0.0005	BC:50; Ep:4000; D:200; Loss:nll; γ:1; LR:0.0005	BC:50; Ep:4000; D:300; Loss:multiclassnll; LR:0.00005; Reg:L3 (λ:0.0001);	BC:100; Ep:4000; D:350; Loss:multiclassnll; LR:0.0001; Reg:L2 (λ:0.0001);
ComplEx	B:100; Ep:200; D:2000; LR:1e-2; Reg:N3; Opt:Adagrad	B:1000; Ep:20; D:2000; LR:0.1; Reg:N3; Opt:Adagrad	B:100; Ep:100; D:2000; LR:0.1; Reg:N3; Opt:Adagrad	B:100; Ep:100; D:2000; LR: 0.1; Reg:N3; Opt:Adagrad
SimplE	B:4832; Ep:1000; D:200; LR:0.05; N:10; Reg:L2 (λ:0.1);	B:1415; Ep:1000; D:200; LR:0.1; N:1; Reg:L2 (λ:0.03);	B:4832; Ep:500; D:200; LR:0.1; N:3; Reg:L2 (λ:0.1);	B:1415; Ep:1000; D:150; LR:0.1; N:10; Reg:L2 (λ:0.03);
TuckER	B:128; Ep:500; D:200; LR:0.003; ϵ:0; Decay Rate:0.99; Drop:{in:0.2; h1:0.2; h2:0.3}	B:128; Ep:500; D:200; LR:0.0005; ϵ:0.1; Decay Rate:0.995; Drop:{in:0.3; h1:0.4; h2:0.5}	B:128; Ep:500; De:200; Dr:30; LR:0.005; ϵ:0.1; Decay Rate:1.0; Drop:{in:0.2; h1:0.1; h2:0.2}	B:128; Ep:500; De:200; Dr:30; LR:0.01; ϵ:0.1; Decay Rate: 1.0; Drop:{in:0.2; h1:0.2; h2:0.3}
TransE	BC:100; Ep:4000; D:150; Loss:multiclassnll; Norm:L1; LR:5e-05; Reg:L3 (λ: 0.0001);	BC:100; Ep:4000; D:150; Loss:multiclassnll; Norm: L1; LR:5e-05; Reg:L3 (λ: 0.0001);	BC:64; Ep:4000; D:400; Loss:multiclassnll; Norm:L1; LR:0.0001; Reg:L2 (λ: 0.0001);	BC:150; Ep:4000; D:350; Loss:multiclassnll; Norm:L1; LR: 0.0001; Reg:L2 (λ: 0.0001);
RotatE	B:1024; Steps:150000; D:1000; LR:0.0001; N:256; γ:24; AdvTemp:1.0;	B:512; Steps:80000; D:500; LR:0.0001; N:1024; γ:12; AdvTemp:0.5;	B:1024; Steps:100000; D:1000; LR:0.00005; N:256; γ:9; AdvTemp:1.0;	B:512; Steps:80000; D:500; LR:0.00005; N:1024; γ:6; AdvTemp:0.5;
QuatE	D:200; reg:(λ1:0.05; λ2:0.05); N:10	D:300; reg:(λ1:0.05; λ2:0.05); N:10	D:100; reg:(λ1:0.3; λ2:0.3); N:10	D: 100; reg:(λ1:0.1; λ2:0.1); N:1
ConvE	B:128; Ep:1000; D:200; LR:0.003; Kernel:3x3; K:256; Decay Rate:0.995; ϵ:0.1; Drop:{in:0.2; h:0.3; feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ϵ:0.1; Kernel:3x3; K:256; Drop:{in:0.2; h:0.3; feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ϵ:0.1; Kernel:3x3; K:256; Drop:{in:0.2; h:0.3; feat:0.2}	B:128; Ep:1000; D:200; LR:0.003; Decay Rate:0.995; ϵ:0.1; Kernel:3x3; K:256; Drop:{in:0.2; h:0.3; feat:0.2}
ConvR	B:128; Ep:1000; De:200; LR:0.001; ϵ:0.1; Drop:{in:0.1; h:0.4; feat:0.2}; Kernel:3x3; K:100;	B:128; Ep:1000; De:200; LR:0.001; ϵ:0.1; Drop:{in:0.4; h:0.3; feat:0.3}; Kernel:3x3; K:100;	B:128; Ep:1000; De:100;LR:0.001; ϵ: 0.1; Drop:{in:0.3; h:0.2; feat:0.3}; Kernel:5x5; K:200;	B:128; Ep:1000; De:200; LR:0.001; ϵ:0.1; Drop:{in:0.2;h:0.2;feat:0.5}; Kernel:3x3; K:200
ConvKB	B:128; Ep:200; D:100; LR:0.0005; K:200; Reg:L2 (λ:0.001)	B:256; Ep:200; D:50; LR:5e-05; K:500; Reg:L2 (λ:0.001)	B:256; Ep:200; D:100; LR:0.000005; K:50; Reg:L2 (λ:0.001);	B:256; Ep:200; D:50; LR:0.0001; K:500; Reg:L2 (λ:0.001);
CoPER-ConvE	B:512; Ep:2000000; D_e:200; D_r:32; LR:0.001; Drop{in:0.2; out:0.2; feature:0.3}; ϵ:0.1; N:100	B:512; Ep:2000000; D_e:200; D_r:8; LR:0.001; Drop{in:0.2; out:0.2; feature:0.3}; ϵ:0.1; N:10	B:512; Ep:2000000; D_e:200; D_r:32; LR:0.001; Drop{in:0.2; out:0.2; feature:0.3}; epsilon:0.1; N:100	B:512; Ep:2000000; D_e:200; D_r:8; LR:0.001; Drop{in:0.2; out:0.2; feature:0.3}; ϵ:0.2; N:10

5) MR and MRR are used with filtered rank.

In FB15k-237 and WN18RR datasets, we can see that embedding models get good results. Both tensor and geometric use matrix and vectors to store the embeddings. From these embedding, they generalize to give a prediction. This approach has one problem: this transformation is not good enough to solve complex relations or find latent information. However, in deep learning models, they use the activation function. Some also use additional information as an attention mechanism or a pre-trained model like BERT, which expects to increase the performance. Although good results, these embeddings are likely to lose important information from the graph, which can reduce the performance.

In addition to embedding models, rule-based models also achieve many successes with shorter training times, such as LogicENN. However, the number of rules is quite large, so the ability to scale on large graphs needs to be considered.

Furthermore, reinforcement-learning models also don't perform as well as tensor, geometric or neural network groups because the reward function is not general enough. In addition, finding a path from entities to others costs a lot of time too. As a result, there are many challenges for these models to get a good performance as deep learning models.

In FB15K and WN18 datasets, both have a test leakage problem. The rule-based and tensor method's performance in these two datasets is much better than theirs in FB15k-237 and WN18RR datasets. Recent models are not focused too much on experimenting on these datasets. In general, performance in the WN18 dataset is much better than in the FB15k dataset. Because the WN18 dataset is semantical, they can generalize and predict better with triples which causes test leakage problems. For example, the Hit@10 values in the WN18 dataset are usually larger than 0.8.

Almost none of them can achieve good performance on all datasets because each dataset has different characteristics. For example, WN18 and WN18RR are semantic datasets, so the amount of knowledge in it is less than that of FB15k, FB15k-237, which are about facts, people in the real world. With these differences, the generalization

TABLE IX
COMPARE THE RESULTS OF THE KG COMPLETION MODELS ON TWO DATASETS, FB15K-237 AND WN18RR.

		FB15k-237				WN18RR			
		Hit@1	**Hit@10**	**MR**	**MRR**	**Hit@1**	**Hit@10**	**MR**	**MRR**
Tensor	**DistMult**	0.224	0.490	199	0.313	.397	.502	5913	0.433
	ComplEx	0.257	0.530	202	0.349	0.426	0.521	4907	0.458
	SimplE	0.100	0.344	651	0.179	.383	.427	8764	0.398
	TuckER	0.266	0.544	-	0.358	0.443	0.526	-	0.470
	NepTuNe	0.272	0.547	-	0.366	0.455	0.557	-	0.491
Geometric	**TransE**	0.217	0.497	209	0.31	0.0279	.495	3936	0.206
	RotatE	0.238	0.531	178	0.336	0.426	0.574	3318	0.475
	QuatE(1)	0.221	0.495	176	0.311	0.436	0.564	3472	0.481
	QuatE(2)	0.271	0.556	-	0.366	0.436	0.572	-	0.482
	Quate(3)	0.248	0.550	**87**	0.348	0.438	0.582	2314	0.588
Neural Network	**ConvE**	0.219	0.476	281	0.305	0.390	0.508	4944	0.427
	ConvR	0.256	0.526	251	0.346	0.437	0.527	5646	0.467
	ConvKB	0.140	0.414	309	0.230	0.0563	0.525	3429	0.249
	InteractE	0.263	0.535	172	0.354	0.43	0.528	5202	0.463
	CoPER-ConvE	0.322	0.629	-	0.426	0.441	0.561	-	0.483
	CoPER-MINERVA	0.295	0.504	-	0.365	0.427	0.510	-	0.465
	KG-BERT	-	0.42	153	-	-	0.524	97	-
	stAR	0.205	0.482	117	0.296	0.243	0.709	51	0.401
	stAR(ensemble)	0.264	0.559	109	0.362	0.449	0.675	540	0.524
	stAR(self-adp)	0.266	0.562	117	0.365	**0.459**	**0.732**	**46**	**0.551**
	KBGAT	**0.460**	**0.626**	210	**0.518**	0.360	.581	1940	0.440
	LSA-GAT	0.400	0.6	273	0.47	0.35	0.58	1947	0.440
	RAGAT	0.273	0.547	199	0.365	0.452	0.562	2390	0.489
Rule-based	**AnyBURL 10s**	0.134	0.259	-	-	0.432	0.527	-	-
	AnyBURL 100s	0.196	0.410	-	-	0.445	0.549	-	-
	AnyBURL 1000s	0.230	0.479	-	-	0.446	0.555	-	-
	AnyBURL 10000s	0.233	0.486	-	-	0.441	0.552	-	-
	LogicENN	-	0.473	424	-	-	-	-	-
RF	**Minerva**	0.284	0.301	-	0.305	-	-	-	-
	Multihop(DistMult)	0.381	0.503	-	0.381	-	-	-	-
	Multihop(ConvE)	0.367	0.533	-	0.427	-	-	-	-
	Meta-KGR(DistMult)	0.403	0.580	-	0.458	-	-	-	-
	Meta-KGR(ConvE)	0.412	0.588	-	0.469	-	-	-	-

of these datasets is a challenge.

Both FB15k and WN18 have test leakage, so they may cause inaccurate results compared to the actual capacity of that model. That has been improved with two datasets FB15k-237 and WN18RR. Therefore, later studies only focus on these two datasets more than the FB15k and WN18 datasets. In addition, some models have problems in the score function that make the results inaccurate as well. We hope that future studies can tackle these issues so that the models can achieve better results.

The detailed results are shown in Table IX and Table X.

5. Evaluating Models on Temporal Graphs

Table XI gives detailed information about the scores using the AUC metric of surveyed models on several popular datasets. Overall, each model evaluates its performance using different datasets, where DBLP is used by three models, followed by Hep-th dataset with two models. In detail, TruncatedSVD has the best performance on the DBLP dataset with 0.936, followed by the CP model with 0.928, while TCOP has the lowest score among the three models surveyed on this dataset. When it comes to the Hep-th dataset, there are two models evaluating the predictive score on this one, namely, ARIMA and CALA-LP. The CALA-LP model has better performance than ARIMA with 0.9321 compared to just over 0.9. Turning to the remaining datasets, as seen from the table, each dataset is used by only one model. The Fast and scalable link propagation model evaluates on the Web dataset with 0.963, the CALA-LP model has the score of 0.8823 on the Hep-th dataset. In contrast, the TCOP model evaluates its performance on the HiePH-cite, and HiePH-colab datasets with the AUC score of 0.7392 and 0.9147, respectively.

TABLE X
COMPARE THE RESULTS OF THE KG COMPLETION MODELS ON TWO DATASETS, FB15K AND WN18RR.

		FB15k				WN18			
		Hit@1	**Hit@10**	**MR**	**MRR**	**Hit@1**	**Hit@10**	**MR**	**MRR**
Tensor	**DistMult**	0.736	0.863	173	0.784	0.726	0.946	675	0.824
	ComplEx	**0.816**	**0.905**	34	**0.848**	0.945	0.955	3623	0.949
	SimplE	0.661	0.836	138	0.726	0.932	0.946	759	0.938
	TuckER	0.729	0.889	39	0.788	0.946	0.958	510	0.951
Geometric	**TransE**	0.494	0.847	45	0.628	0.406	0.949	279	0.646
	TransH	-	0.644	87	-	-	0.823	388	-
	TransR	-	0.687	77	-	-	0.920	225	-
	CtransR	-	0.702	75	-	-	0.923	218	-
	TransD	-	0.773	91	-	-	0.922	212	-
	RotatE	0.739	0.881	42	0.791	0.943	0.960	274	0.949
	QuatE(1)	0.700	0.878	41	0.770	0.941	0.960	388	0.949
	QuatE(2)	0.800	0.900	-	0.833	0.944	**0.962**	-	0.950
	Quate(3)	0.711	0.900	**17**	0.782	0.945	0.959	**162**	0.950
Neural Network	**ConvE**	0.595	0.849	51	0.688	0.939	0.957	413	0.945
	ConvR	0.706	0.886	70	0.773	**0.946**	0.959	471	**0.950**
	ConvKB	0.114	0.408	324	0.211	0.529	0.949	202	0.709
Rule-based	**AnyBURL 10s**	0.796	0.838	-	-	0.942	0.949	-	-
	AnyBURL 100s	0.808	0.876	-	-	**0.946**	0,959	-	-
	AnyBURL 1000s	0.804	0.890	-	-	0.939	0.956	-	-
	AnyBURL 10000s	0.796	0.887	-	-	0.935	0.954	-	-
	KALE	0.383	0.762	-	0.523	-	-	-	-
	RUGE	0.703	0.865	-	0.768	-	-	-	-
	LogicENN	-	0.669	175	0.402	-	0.842	368	0.623

TABLE XI
COMPARE THE RESULTS OF THE TEMPORAL LINK PREDICTION MODELS ON SIX DATASETS.

	DBLP	**Web dataset**	**Hep-th dataset**	**Hep-ph dataset**	**HiePH-cite**	**HiePH-colab**
Fast and scalable link propagation	-	0.963	-	-	-	-
TruncatedSVD	0.936	-	-	-	-	-
CP	0.928	-	-	-	-	-
TCOP	0.859	-	-	-	0.7392	0.9147
ARIMA	-	-	>0.9?	-	-	-
CALA-LP	-	-	0.9321	0.8823	-	-

IX. COMPARE AND CONTRAST

As discussed in the previous sections, both on static and temporal graphs, the approaches have many similarities. For example, using embeddings is suitable for both static and temporal graphs. However, some embedding techniques are not suitable to apply to temporal graphs, such as geometric embeddings. The geometric embedding technique uses each space to represent the graph at a specific time. After that, they combine the information from these spaces to predict a temporal graph. However, that is not effective. Similarly, applying rule-based methods to temporal graphs is challenging due to the enormous rules generated from training. Reinforcement learning methods do not achieve good performance in a static graph, so they do not perform well at the temporal graph.

The methods on temporal graphs tend to involve more knowledge about time series or probability to make inferences instead of projecting onto embedded space. The reason is that dynamic graphs constantly change over time, so it is more advantageous to apply Markov chains for inference. Meanwhile, in static graphs, there is no need to handle this.

In addition, because the input of dynamic graph methods is based on the previous n-1 graph slices to predict at time t, the computational cost is enormous compared to the models on the static graph. Applying state-of-the-art algorithms from static graphs to dynamic graphs is much more difficult, especially in reducing the computational cost. For example, using an attention engine or BERT on a static graph costs a lot of computation. It is much more time-consuming when transferring to a dynamic graph.

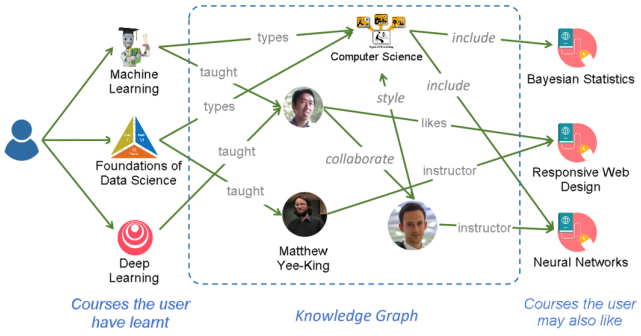


Figure 39. The course recommendation system applies link prediction

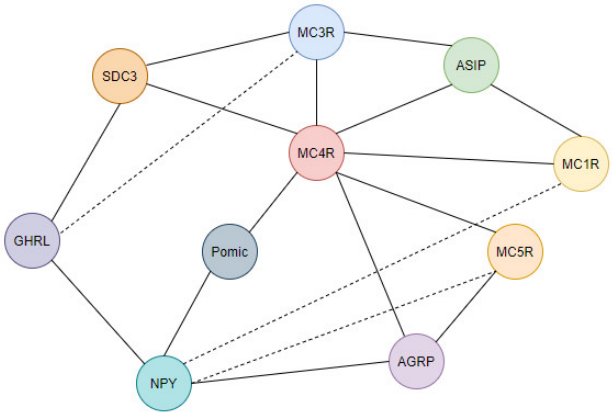


Figure 40. Apply link prediction to protein-protein interactions

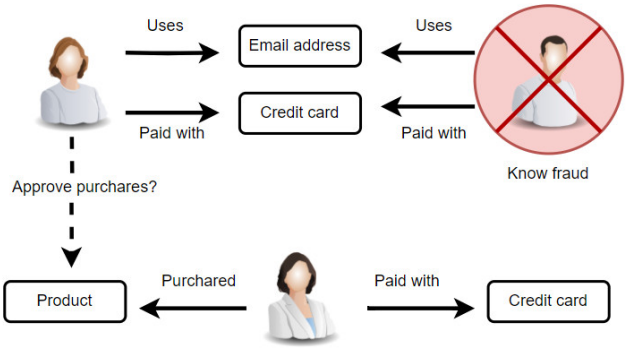


Figure 41. A fraud detection system based on link prediction

X. Applications

Link prediction on the knowledge graph has many applications up to now. In this section, we describe some typical applications that have been employed in many practical systems.

1. Recommender System

A recommender system is a system that provides relevant, valuable information to personalize the user. Usually, the recommendation system has applied collaborative filtering algorithms to produce reasonable recommendations. For sparse data, however, these collaborative filtering algorithms are quite limited. Therefore, applying link prediction to the recommendation system is considered an effective method [120]. Because of the data-filling nature of the link prediction, it generates more useful relevant information and makes the recommendation richer but equally effective. Some of the recommended systems that have been implemented include movie recommendations like NetFlix, product recommendations like Amazon, music recommendations like Spotify. Fig. 39 illustrates a recommender system to suggest the courses.

2. Protein-Protein Interaction

Protein-protein interactions (PPIs) play an important role in essential biological processes of organisms, such as metabolism, DNA replication, molecular transport. PPTs help better understand how diseases work, leading to effective research for cures or drug and vaccine development. However, the data on interactions between proteins is limited, so it is necessary to use link prediction methods for these interactions. Link prediction helps uncover interactions that have not yet been found, adding essential information for future biological studies [121]. Fig. 40 gives an example in PPIs.

3. Privacy Control in Social Networks

Today, registering a social network account is relatively easy because the user's information when registering is not necessarily accurate. Some guys maybe create fake accounts with many purposes, bad intentions, affecting other users. Therefore, link prediction is applied to detect abnormal and counterfeit accounts to control security better and protect users in social networks [122]. Fig. 41 depicts a fraud detection system based on link prediction.

4. Question Answering

Question answering is the execution of a data query to get the correct data to produce outputs that match the inputs. Therefore, achieving high performance requires a large enough amount of data and closely linked together. Applying link prediction to the question-answering system has solved these problems [25]. Link prediction makes data denser and more seamlessly related, thereby helping to extract fuller and more efficient outputs.

5. Disease Prediction

Link prediction in the medical and health fields is also gaining popularity, and one of them is predicting disease for patients [123]. It works by predicting the association between diseases. For example, for a given pair of diseases (A, B), if a patient has disease A, the probability of disease B is likely to occur and vice versa. In other words, in the disease prediction problem, consider the vertices as the diseases and the edges as the association relationship between the diseases. The higher this association, the more likely the prediction is to be accurate.

6. Enzymatic Link Prediction

As we all know, enzymes are involved in most of the essential metabolic processes in the body, such as detoxification, transport, energy supply. Hence, the study of enzymes is significant. However, the discovery of the association and modification of enzymes with molecules in biology is still incomplete. Therefore, enzymatic link prediction is applied to predict the probability of enzyme-particle reactions to provide more information about the variation between them [124]. From there, they create a foundation for research development in this field.

XI. Future Direction

The knowledge graph represents information from the real world, so it contains a lot of data. The better holding this knowledge, the better performance for link prediction. Some possible directions for this problem:

- Add implicit information to the model, especially neural network models, with as little computational cost as possible.
- Combine models in a group or between two groups to create a new model with higher performance. For example, the combination of a matrix factorization and a neural network-based models because the matrix costs less to compute, but the neural network has activation functions that help learn more hidden semantic layers; or combine information from the rule-based model.
- Applying achievements from static graph to dynamic graph. Although the dynamic graph changes over time, at the specified time t, it is also considered a static graph. Therefore, it is possible to apply the achievements from the static graph along with making the model compact and takes fewer parameters.
- Apply KG's models to many other fields such as image processing, natural language processing because of its ability to handle multi-dimensional data well.
- Deploying KG into practical applications. The increasingly large data and the flexibility and ease of handling of the graph structure make the ability to create many applications valuable in real life.

XII. Conclusion

In this paper, we study the link prediction problem on the knowledge graph. It is one of the exciting problems since Google published knowledge graph data in 2012. Due to the different properties between static and temporal graphs, this problem is divided into two research branches. The first branch is Knowledge Graph Completion which aims to predict the missing link on the static graph. The other branch is temporal link prediction which aims to predict relation over time. In the knowledge completion, we have divided into three main approaches, including matrix decomposition models, geometric-based models, and deep learning-based models. Through the survey, we found that almost no method excels on all standard datasets. In other words, in each group of methods, there are algorithms with high predictive results in some datasets. Next, we divide models in the temporal link prediction branch into five main groups, including matrix factorization, probabilistic, spectral clustering, time series, and deep learning. In each group of methods, we describe the basic models and subsequent significantly improved versions. From there, it is possible to see the formation of ideas to solve the link prediction problem. Datasets and benchmarks are also described for the analysis and compare algorithms. In the end, we present practical applications and suggest some directions for improvement in the future.

References

[1] L. Ulanoff, "Google search just got 1,000 times smarter," May 2012. [Online]. Available: https://mashable.com/2012/05/16/google-knowledge-graph/

[2] E. W. Schneider, "Course modularization applied: The interface system and its implications for sequence control and data analysis." 1973.

[3] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: lessons and challenges," *Communications of the ACM*, vol. 62, no. 8, pp. 36–43, 2019.

[4] D. Devarajan, "Happy birthday watson discovery," 2017. [Online]. Available: https://www.ibm.com/cloud/blog/announcements/happy-birthday-watson-discovery

[5] Q. He, B. Chen, and D. Agarwal, "Building the linkedin knowledge graph," *Engineering. linkedin. com*, 2016.

[6] S. E. Shimony, C. Domshlak, and E. Santos Jr, "Cost-sharing in bayesian knowledge bases," *arXiv preprint arXiv:1302.1567*, 2013.

[7] A. Krishnan, "Making search easier," Aug 2018. [Online]. Available: https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier

[8] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.

[9] R. Speer, J. Chin, and C. Havasi, "Conceptnet 5.5: An open multilingual graph of general knowledge," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[10] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[11] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1247–1250.

[12] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 601–610.

[13] F. Stefano, I. Finocchi, S. P. Ponzetto, and V. Paola, "Webisagraph: A very large hypernymy graph from a web corpus," in *Sixth Italian Conference on Computational Linguistics*, 2019.

[14] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "Yago2: A spatially and temporally enhanced knowledge base from wikipedia," *Artificial Intelligence*, vol. 194, pp. 28–61, 2013.

[15] G. Bouchard, S. Singh, and T. Trouillon, "On approximate reasoning capabilities of low-rank vector spaces." in *AAAI spring symposia*. Citeseer, 2015.

[16] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *arXiv preprint arXiv:1111.4503*, 2011.

[17] B. Shi and T. Weninger, "Fact checking in heterogeneous information networks," in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016, pp. 101–102.

[18] D. Lukovnikov, A. Fischer, J. Lehmann, and S. Auer, "Neural network-based question answering over knowledge graphs on word and character level," in *Proceedings of the 26th international conference on World Wide Web*, 2017, pp. 1211–1220.

[19] B. Hachey, W. Radford, J. Nothman, M. Honnibal, and J. R. Curran, "Evaluating entity linking with wikipedia," *Artificial intelligence*, vol. 194, pp. 130–150, 2013.

[20] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2015.

[21] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Y. Philip, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[22] S. Samanta and M. Pal, "Link prediction in social networks," in *Graph Theoretic Approaches for Analyzing Large-Scale Social Networks*. IGI Global, 2018, pp. 164–172.

[23] D. Oniani, G. Jiang, H. Liu, and F. Shen, "Constructing co-occurrence network embeddings to assist association extraction for covid-19 and other coronavirus infectious diseases," *Journal of the American Medical Informatics Association*, vol. 27, no. 8, pp. 1259–1267, 2020.

[24] G. Berlusconi, F. Calderoni, N. Parolini, M. Verani, and C. Piccardi, "Link prediction in criminal networks: A tool for criminal intelligence analysis," *PloS one*, vol. 11, no. 4, p. e0154244, 2016.

[25] D. Dalal, M. Arcan, and P. Buitelaar, "Enhancing multiple-choice question answering with causal knowledge," in *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, 2021, pp. 70–80.

[26] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[27] V. Martínez, F. Berzal, and J.-C. Cubero, "A survey of link prediction in complex networks," *ACM computing surveys (CSUR)*, vol. 49, no. 4, pp. 1–33, 2016.

[28] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[29] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2017.

[30] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[31] S. Haghani and M. R. Keyvanpour, "A systemic analysis of link prediction in social network," *Artificial Intelligence Review*, vol. 52, no. 3, pp. 1961–1995, 2019.

[32] G. A. Gesese, R. Biswas, and H. Sack, "A comprehensive survey of knowledge graph embeddings with literals: Techniques and applications." in *DL4KG@ ESWC*, 2019, pp. 31–40.

[33] R. Biswas, "Embedding based link prediction for knowledge graph completion," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3221–3224.

[34] Z. Chen, Y. Wang, B. Zhao, J. Cheng, X. Zhao, and Z. Duan, "Knowledge graph completion: A review," *IEEE Access*, vol. 8, pp. 192 435–192 456, 2020.

[35] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres *et al.*, "Knowledge graphs," *arXiv preprint arXiv:2003.02320*, 2020.

[36] S. Arora, "A survey on graph neural networks for knowledge graph completion," *arXiv preprint arXiv:2007.12374*, 2020.

[37] Y. Dai, S. Wang, N. N. Xiong, and W. Guo, "A survey on knowledge graph embedding: Approaches, applications and benchmarks," *Electronics*, vol. 9, no. 5, p. 750, 2020.

[38] A. Kumar, S. S. Singh, K. Singh, and B. Biswas, "Link prediction techniques, applications, and performance: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 553, p. 124289, 2020.

[39] A. Divakaran and A. Mohan, "Temporal link prediction: a survey," *New Generation Computing*, pp. 1–46, 2019.

[40] B. Andreas and N. Helmut, "The knowledge graph cookbook recipes that work," *Edition mono/monochrom, Vienna*, 2020.

[41] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler, *Knowledge Graphs*. Springer, 2020.

[42] M. Wang, L. Qiu, and X. Wang, "A survey on knowledge graph embeddings for link prediction," *Symmetry*, vol. 13, no. 3, p. 485, 2021.

[43] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*. Springer, 2011, pp. 243–275.

[44] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, and P. Merialdo, "Knowledge graph embedding for link prediction: A comparative analysis," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 2, pp. 1–49, 2021.

[45] M. Kejriwal, C. A. Knoblock, and P. Szekely, *Knowl-*

edge Graphs: Fundamentals, Techniques, and Applications. MIT Press, 2021.

[46] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Neural Information Processing Systems (NIPS)*, 2013, pp. 1–9.

[47] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.

[48] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[49] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *International Conference on Machine Learning*. PMLR, 2016, pp. 2071–2080.

[50] S. S. Dasgupta, S. N. Ray, and P. Talukdar, "Hyte: Hyperplane-based temporally aware knowledge graph embedding," in *Proceedings of the 2018 conference on empirical methods in natural language processing*, 2018, pp. 2001–2011.

[51] C. Xu, M. Nayyeri, F. Alkhoury, H. S. Yazdi, and J. Lehmann, "Tero: A time-aware knowledge graph embedding via temporal rotation," *arXiv preprint arXiv:2010.01029*, 2020.

[52] A. Garcia-Duran, A. Bordes, and N. Usunier, "Composing relationships with translations," Ph.D. dissertation, *CNRS, Heudiasyc*, 2015.

[53] N. M. A. Ibrahim and L. Chen, "Link prediction in dynamic social networks by integrating different types of information," *Applied Intelligence*, vol. 42, no. 4, pp. 738–750, 2015.

[54] İ. Güneş, Ş. Gündüz-Öğüdücü, and Z. Çataltepe, "Link prediction using time series of neighborhood-based node similarity scores," *Data Mining and Knowledge Discovery*, vol. 30, no. 1, pp. 147–180, 2016.

[55] B. Moradabadi and M. R. Meybodi, "Link prediction based on temporal similarity metrics using continuous action set learning automata," *Physica a: statistical mechanics and its applications*, vol. 460, pp. 361–373, 2016.

[56] Y. Dhote, N. Mishra, and S. Sharma, "Survey and analysis of temporal link prediction in online social networks," in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2013, pp. 1178–1183.

[57] T. Jiang, T. Liu, T. Ge, L. Sha, S. Li, B. Chang, and Z. Sui, "Encoding temporal information for time-aware link prediction," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2350–2354.

[58] R. Goel, S. M. Kazemi, M. Brubaker, and P. Poupart, "Diachronic embedding for temporal knowledge graph completion," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3988–3995.

[59] C. Xu, M. Nayyeri, F. Alkhoury, H. S. Yazdi, and J. Lehmann, "Temporal knowledge graph embedding model based on additive time series decomposition," *arXiv preprint arXiv:1911.07893*, 2019.

[60] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[61] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," *arXiv preprint arXiv:1902.10197*, 2019.

[62] G. Asefa Gesese, R. Biswas, M. Alam, and H. Sack, "A survey on knowledge graph embeddings with literals: Which model links better literal-ly?" *arXiv e-prints*, pp. arXiv–1910, 2019.

[63] T. Li, J. Zhang, S. Y. Philip, Y. Zhang, and Y. Yan, "Deep dynamic network embedding for link prediction," *IEEE Access*, vol. 6, pp. 29 219–29 230, 2018.

[64] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in neural information processing systems*. Citeseer, 2013, pp. 926–934.

[65] B. Kotnis and V. Nastase, "Analysis of the impact of negative sampling on link prediction in knowledge graphs," *arXiv preprint arXiv:1708.06816*, 2017.

[66] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka, and T. Mitchell, "Toward an architecture for never-ending language learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010.

[67] Y. Peng and J. Zhang, "Lineare: Simple but powerful knowledge graph embedding for link prediction," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 422–431.

[68] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, 2015, pp. 57–66.

[69] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[70] F. Mahdisoltani, J. Biega, and F. Suchanek, "Yago3: A knowledge base from multilingual wikipedias," in *7th biennial conference on innovative data systems research*. CIDR Conference, 2014.

[71] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[72] B. Shi and T. Weninger, "Open-world knowledge graph completion," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[73] A. Breit, S. Ott, A. Agibetov, and M. Samwald, "Openbiolink: a benchmarking framework for large-scale biomedical link prediction," *Bioinformatics*, vol. 36, no. 13, pp. 4097–4098, 2020.

[74] M. Zitnik, R. Sosic, and J. Leskovec, "Biosnap datasets: Stanford biomedical network dataset collection," *Note: http://snap. stanford. edu/biodata Cited by*, vol. 5, no. 1, 2018.

[75] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Icml*, 2011.

[76] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," *arXiv preprint arXiv:1802.04868*, 2018.

[77] I. Balažević, C. Allen, and T. M. Hospedales, "Tucker: Tensor factorization for knowledge graph completion," *arXiv preprint arXiv:1901.09590*, 2019.

[78] S. Sonkar, A. Katiyar, and R. G. Baraniuk, "Neptune: Neural powered tucker network for knowledge graph completion," *arXiv preprint arXiv:2104.07824*, 2021.

[79] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[80] P. Minervini, C. d'Amato, N. Fanizzi, and F. Esposito, "Efficient learning of entity and predicate embeddings for link prediction in knowledge graphs." in *URSW@ ISWC*, 2015, pp. 26–37.

[81] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity

and relation embeddings for knowledge graph completion," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[82] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, 2015, pp. 687–696.

[83] G. Ji, K. Liu, S. He, and J. Zhao, "Knowledge graph completion with adaptive sparse transfer matrix," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[84] T. Ebisu and R. Ichise, "Toruse: Knowledge graph embedding on a lie group," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[85] S. Zhang, Y. Tay, L. Yao, and Q. Liu, "Quaternion knowledge graph embeddings," *arXiv preprint arXiv:1904.10281*, 2019.

[86] H. Lu and H. Hu, "Dense: An enhanced non-abelian group representation for knowledge graph embedding," *arXiv preprint arXiv:2008.04548*, 2020.

[87] X. Jiang, Q. Wang, and B. Wang, "Adaptive convolution for multi-relational learning," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 978–987.

[88] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Phung, "A novel embedding model for knowledge base completion based on convolutional neural network," *arXiv preprint arXiv:1712.02121*, 2017.

[89] G. Stoica, O. Stretcu, E. A. Platanios, T. Mitchell, and B. Póczos, "Contextual parameter generation for knowledge graph link prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, 2020, pp. 3000–3008.

[90] S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. Talukdar, "Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, 2020, pp. 3009–3016.

[91] D. Nathani, J. Chauhan, C. Sharma, and M. Kaul, "Learning attention-based embeddings for relation prediction in knowledge graphs," *arXiv preprint arXiv:1906.01195*, 2019.

[92] K. Ji, B. Hui, and G. Luo, "Graph attention networks with local structure awareness for knowledge graph completion," *IEEE Access*, 2020.

[93] X. Liu, H. Tan, Q. Chen, and G. Lin, "Ragat: Relation aware graph attention network for knowledge graph completion," *IEEE Access*, vol. 9, pp. 20 840–20 849, 2021.

[94] L. Yao, C. Mao, and Y. Luo, "Kg-bert: Bert for knowledge graph completion," *arXiv preprint arXiv:1909.03193*, 2019.

[95] B. Wang, T. Shen, G. Long, T. Zhou, Y. Wang, and Y. Chang, "Structure-augmented text representation learning for efficient knowledge graph completion," in *Proceedings of the Web Conference 2021*, 2021, pp. 1737–1748.

[96] L. Cai and W. Y. Wang, "Kbgan: Adversarial learning for knowledge graph embeddings," *arXiv preprint arXiv:1711.04071*, 2017.

[97] C. Meilicke, M. W. Chekol, D. Ruffinelli, and H. Stuckenschmidt, "Anytime bottom-up rule learning for knowledge graph completion." in *IJCAI*, 2019, pp. 3137–3143.

[98] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo, "Jointly embedding knowledge graphs and logical rules," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2016, pp. 192–202.

[99] J. Yuan, N. Gao, and J. Xiang, "Transgate: knowledge graph embedding with shared gate structure," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3100–3107.

[100] W. Zhang, B. Paudel, L. Wang, J. Chen, H. Zhu, W. Zhang, A. Bernstein, and H. Chen, "Iteratively learning embeddings and rules for knowledge graph reasoning," in *The World Wide Web Conference*, 2019, pp. 2366–2377.

[101] M. Nayyeri, C. Xu, J. Lehmann, and H. S. Yazdi, "Logicenn: A neural based knowledge graphs embedding model with logical rules," *arXiv preprint arXiv:1908.07141*, 2019.

[102] W. Xiong, T. Hoang, and W. Y. Wang, "Deeppath: A reinforcement learning method for knowledge graph reasoning," *arXiv preprint arXiv:1707.06690*, 2017.

[103] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning," *arXiv preprint arXiv:1711.05851*, 2017.

[104] C. Raymond, R. M. Horton, J. Zscheischler, O. Martius, A. AghaKouchak, J. Balch, S. G. Bowen, S. J. Camargo, J. Hess, K. Kornhuber *et al.*, "Understanding and managing connected extreme events," *Nature climate change*, vol. 10, no. 7, pp. 611–621, 2020.

[105] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, pp. 1–27, 2011.

[106] R. A. Harshman *et al.*, "Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis," 1970.

[107] H. A. Kiers, "Towards a standardized notation and terminology in multiway analysis," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 14, no. 3, pp. 105–122, 2000.

[108] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[109] V. Ouzienko, Y. Guo, and Z. Obradovic, "Prediction of attributes and links in temporal social networks." in *ECAI*, 2010, pp. 1121–1122.

[110] P. Sarkar, D. Chakrabarti, and M. Jordan, "Nonparametric link prediction in dynamic networks," *arXiv preprint arXiv:1206.6394*, 2012.

[111] T. Wang, X.-S. He, M.-Y. Zhou, and Z.-Q. Fu, "Link prediction in evolving networks based on popularity of nodes," *Scientific reports*, vol. 7, no. 1, pp. 1–10, 2017.

[112] T. Wu, C.-S. Chang, and W. Liao, "Tracking network evolution and their applications in structural network analysis," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 562–575, 2018.

[113] G. G. Chowdhury, *Introduction to modern information retrieval*. Facet publishing, 2010.

[114] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[115] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 969–976.

[116] M. Rahman, T. K. Saha, M. A. Hasan, K. S. Xu, and C. K. Reddy, "Dylink2vec: Effective feature representation for link prediction in dynamic networks," *arXiv preprint arXiv:1804.05755*, 2018.

[117] W. Yu, W. Cheng, C. C. Aggarwal, H. Chen, and W. Wang,

"Link prediction with spatial and temporal consistency in dynamic networks." in *IJCAI*, 2017, pp. 3343–3349.

[118] M. Fey, J. E. Lenssen, F. Weichert, and H. Muller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 869–877.

[119] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs." 2019.

[120] N. Talasu, A. Jonnalagadda, S. S. A. Pillai, and J. Rahul, "A link prediction based approach for recommendation systems," in *2017 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2017, pp. 2059–2062.

[121] H. Y. Yuen and J. Jansson, "Better link prediction for protein-protein interaction networks," in *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, 2020, pp. 53–60.

[122] Y. Zheng, B. Wang, W. Lou, and Y. T. Hou, "Privacy-preserving link prediction in decentralized online social networks," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 61–80.

[123] F. Folino and C. Pizzuti, "Link prediction approaches for disease networks," in *International Conference on Information Technology in Bio-and Medical Informatics*. Springer, 2012, pp. 99–108.

[124] J. Jiang, L.-P. Liu, and S. Hassoun, "Learning graph representations of biochemical networks and its application to enzymatic link prediction," *Bioinformatics*, vol. 37, no. 6, pp. 793–799, 2021.

Thanh Le is a PhD student in Computer Science at the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam. He has been lecturing courses related to artificial intelligence and machine learning in the CS program at the university since 2007. His primary areas of research interest are data mining, big data, and the knowledge graph. He is also a member of IEEE Students and Young Professionals.
Email: lnthanh@fit.hcmus.edu.vn

Hoang Nguyen is a final year undergraduate student majoring in Computer Sciences at the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam. He is working on his thesis on the topic related to the knowledge graph. At the same time, he participates in research at the Computer Science lab and is an active member in the discussions. He devotes much of his efforts to data mining, machine learning, and link prediction on the knowledge graph.
Email: nvm569107@gmail.com

Bac Le is currently a Professor and Head of Department of Computer Science, Faculty of Information Technology, University of Science, Vietnam National University, Ho Chi Minh City, Vietnam. His main research includes artificial intelligence, soft computing, machine learning and data mining.
Email: lhbac@fit.hcmus.edu.vn