J. COMBIN. MATH. COMBIN. COMPUT. 127b (2025) 7691--7714

Journal of Combinatorial Mathematics
and Combinatorial Computing
www.combinatorialpress.com/jcmcc

Combinatorial Press

Tucker inference learning method for knowledge graph decision making

Jicang Xu[1], Ming Li[1,✉], Cheukhang Leung[2]

[1] *School of Economics and Management, China University of Petroleum, Beijing, 102249, China*

[2] *Hong Kong Institute of AI for Science, City University of Hong Kong, Hong Kong*

ABSTRACT

This study aims to build a framework called Tucker Reasoning Learning Method to train the upper layer knowledge graph (KG) making explainable and reasonable decisions. The numerical experiments show that the accuracy is 84%. The contributions are as follows: (i) It overcomes in-explainable problems of using deep learning method; (ii) It has more feedback rings and reasonable paths than decision tree method; (iii) Compared with RESCAL's application in reasoning domain, it enhances 22 percentage points. It is suitable for application scenarios like financial, justice, and medical decision-making, which require explainable and reasoning paths. This study builds a framework called Tucker Reasoning Learning Method to train the upper layer knowledge graph to make explainable and reasonable decisions. The method has the accuracy of 84%, which enhances 22 percentage points compared to the SOTA methods.

Keywords: Knowledge graph, Tensor analysis, Tucker reasoning, Decision making

1. Introduction

Knowledge graph is a popular research direction in the field of artificial intelligence in recent years, which is a method of representing and organizing knowledge in the form of graphs. By constructing a knowledge graph, information can be integrated from different data sources in multiple domains, and potential connections and laws can be discovered through knowledge reasoning methods [1]. Knowledge reasoning is an important part of knowledge mapping, which refers to the use of knowledge in the knowledge map to generate new knowledge or evaluate existing knowledge. It utilizes the extracted and represented knowledge for logical reasoning and data analysis to discover associations and patterns among the

✉ Corresponding author.
E-mail address: limingzyq@cup.edu.cn (M. Li).
Received 05 March 2024; Revised 25 May 2024; Accepted 05 December 2024; Published Online 16 April 2025.
DOI: 10.61091/jcmcc127b-421

knowledge [2]. A typical reasoning task is to uncover and infer additional information in the graph based on the conditions in the query [3]. Relational reasoning based on knowledge graph can utilize information such as contextual relationships and entity attributes in the knowledge graph to obtain new relationships through transfer and inference between relationships. In knowledge graph, reasoning techniques are usually categorized into logic-based reasoning and statistical reasoning [4]. Knowledge graph technology is widely used in various fields. For example, in the medical field, knowledge graphs are used to store medical knowledge and can be used to communicate with medical research and practice [5]. In finance, knowledge graphs are used to represent assets and risks to help organizations make better financial decisions [6]. In the field of education, knowledge graphs are used to store and analyze learners' learning data in order to provide learners with customized educational programs [7]. With the rise of big data and artificial intelligence, tensor decomposition, as an important data analysis method, has more and more extensive applications[8].Tucker decomposition is one of the classical tensor generalized decomposition methods, which can represent high-dimensional tensor with low-dimensional approximation, thus helping us to better understand and process the data[9], and has been widely used in data dimensionality reduction, pattern recognition, image processing, machine learning, computer vision, data mining and other fields [10-13].

This study aims to build a framework called Tucker Reasoning Learning Method to train the upper layer knowledge graph making explainable and reasonable decisions. Instead of using a KG as feature matrix inputs of machine learning and deep learning to support decision-making, the proposed method conserves the explainability of reasoning paths.

2. Research thought

To explain the application expectations of this method, we propose two possible application scenarios. The first example is related to a behaviour finance situation where gossip about a listed company director is broadcast. As visualized in Figure 1(a), it is possible to happen following things in the company simultaneously that:

1) The director resigns;
2) The board of directors reorganizes;
3) To control the company, the new directors make block trade;
4) The stock price goes up.

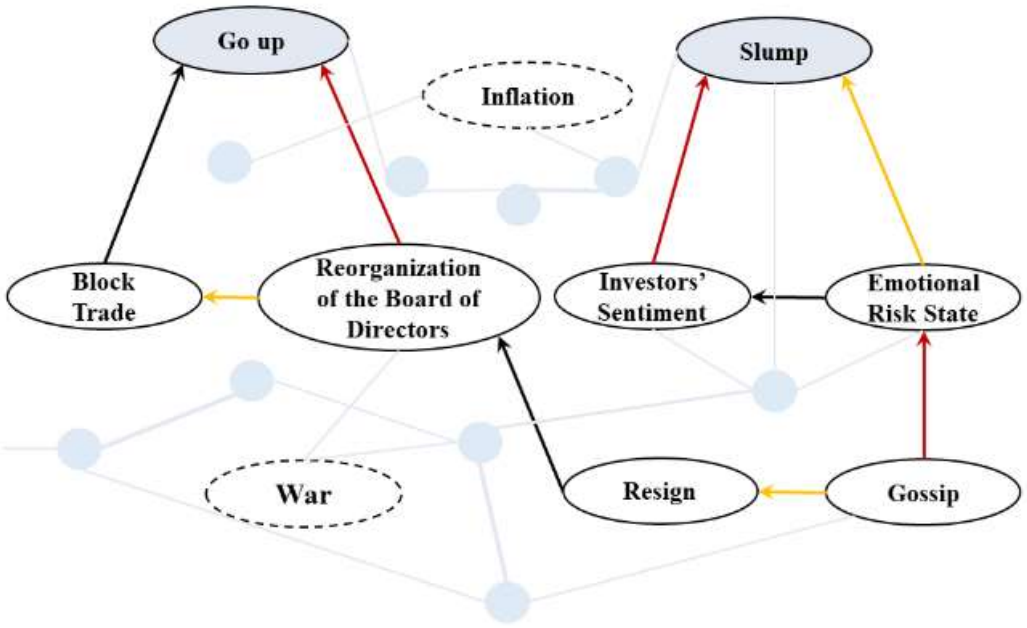
For the folks:

1) Gossip brings about emotional risks among investors with negative sentiments;
2) the stock price slumps.

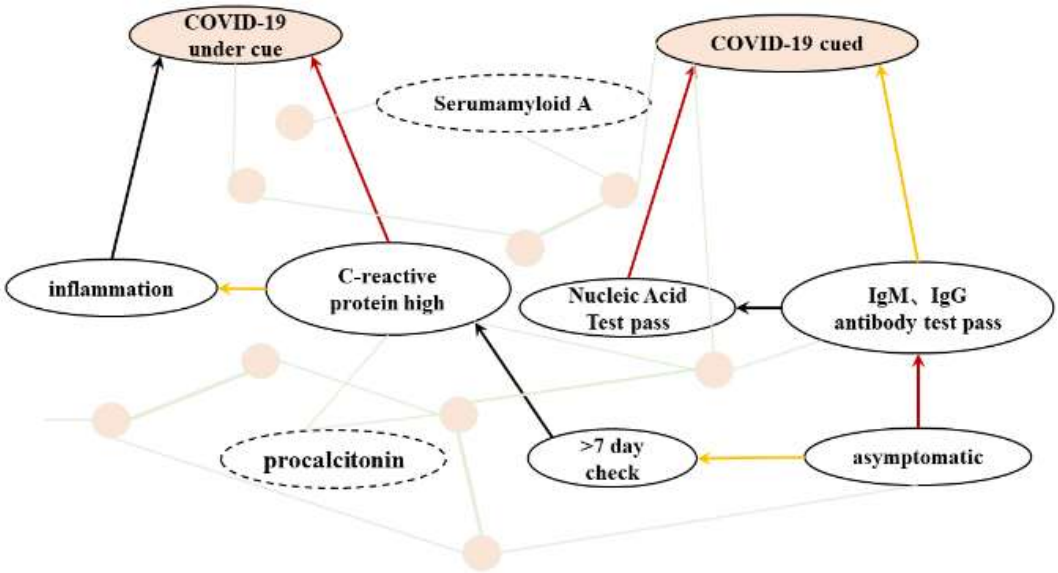
In this scenario, whether the stock price goes up or slumps is hard to predict.

Another example relates to a pandemic. The COVID-19 epidemic used to be rampant in the world. With good immune escape and infection ability, COVID-19 infected patients were hard to be judged whether they were cured. In Figure 1(b), it is possible to happen on patients simultaneously that they were:

1) Asymptomatic;
2) Passing the IgM and IgG antibody test;
3) Passing the nucleic acid test. Still, their medical records were under seven days, and the C-reactive protein was high, showing inflammation in the body.



(a) Scenario of public health



(b) Scenario of behavior finance

Fig. 1. (a) Depicts the decision network for stock price change in behavior finance scenario; (b) Depicts the decision network for quarantine release in public health scenario

Note: The two highlighted nodes on the top of each figure are decision nodes. The dash nodes mean the impact is uncertain and the relationship is not clear; the black line denotes weak connection; the yellow line denotes mid connection; the red line denotes strong connection.

In scenario 2, it is also hard to say whether the patient is cured or not. The proposed model can train related cases accumulated in clinical records and help to make a judgement. The two examples show the necessity of a multiple-step reasoning method of knowledge graph that conserves the explainability of reasoning paths.

Compared to Link Prediction and Subgraph Matching, the proposed Tucker Reasoning Learning Method

has two advantages on making decisions: (1) It is based on human constructed upper layer KG, which is more reliable; (2) The KG's scale and variable scale is small and fixed while making decisions.

The rest of the study is organized as follows. Section 2 introduces related works of other reasoning methods in the KG domain. Section 3 polishes the research objectives in one more step. In Section 4, we propose our method and study it mathematically. The object function and learning algorithms are defined. Section 5 summarizes the comprehensive experiments. In Section 6, based on the findings, the conclusions are drawn, including the research contributions, limitations, and subsequent research directions. The notations in following sections are defined in Table 1.

Table 1. Notations

h, p, t	head node, path, tail node
C	core tensor in Tucker Decomposition belong to $\square^{l\times m\times n}$
$\hat{\mathrm{C}}$	unique best approximation solution on parameter
$\mathrm{S}^{p\times p}$	set of all symmetric matrices of size $p\times p$
K	column-wise orthogonal matrices
Z	knowledge graph embedding tensor
$\otimes$	kronecker product
$\grave{o}(\cdot)$	edge set function
$Tr[\cdot]$	matrix trace

3. Related work

Before moving to next section, we give a short introduction about the KG notations of head, tail, and relation type. A knowledge graph is composed of three components: the head node-set (h), the tail node-set (t), and the relation type-set (r). Generally, the head node-set is the reason set, and the tail node-set is the result set. Consider the knowledge graph in Figure 1(b) as an example. Since there is an arrow pointing from the node "asymptomatic" to the node "> 7 day check", the node "asymptomatic" lies in the head node-set, the node "> 7 day check" lies in the tail node-set, and the yellow arrow representing mid connection lies the relation type-set. Traditionally, two types of reasoning relationships (either exist or not) are considered. However, traditional approach cannot reflect various types of relationships in KG. For instance, the node "Gossip" in Figure 1(a) definitely causes "Emotional risk state" and only possibly causes "Resign". On the other hand, knowledge graphs are not restricted to certain types of relationships. But this flex feature can lead to a sparsity problem in tensor representation. In face of this dilemma, we refer to the earlier version of knowledge graph technology called semantic web, which has limited relationships to cluster various types of relationships. And semantic web relationships can cover all the relationships in knowledge graph technology. Based on literature reviews on semantic web, the knowledge graph (KG) relationship classification map is listed in Table 2. Thus, we only consider three types of reasoning relationships: strong reasoning relationship (S), mid reasoning relationship (M), and weak reasoning relationship (W). This literature review result is applied in the embedding process of KG to a three frontal slice tensor.

Table 2. The relationship types in knowledge graph

Connection types	Beynon-Davies classification	Gao et al. classification	Chowdhary classification
Strong connection	“ISA”, “AKO”	“ISA”, “AKO”	“ISA”, “AKO”
Mid connection	-	“a-member of”	“a-member of”
	-	“subset-of”	“subset-of”
	“association”	-	-
	“part-of”	“a-part-of” “similar-to”	“attributes” “has-parts” “shaped-like”
	-	“composed of”	-
	-	-	“instance-of”
	“has-a”	“have”	-
	-	“possible reason”	-
Weak connection	-	“before-after”	-
	-	“located-at”	-
	-	-	“agent”

4. Objectives

Following the motivations in Section 1, our objectives are as follows:
1) We train the model to predict the decision based on upper layer KG and give the most reliable paths that support the decision. Thus, the inputs are numerous tagged samples of sub-graphs in the upper layer KG and the outputs are decision results, likelihood value, and most reliable reasoning paths. The tagged samples can be easy extracted from unstructured clinical records with KG techniques.
2) When we train the model, we try to improve the accuracy on condition of conserving explain-ability, since many scenarios like finance, justice, and medical treatment have this kind preference. Even if some alternatives provided by machine may be more accurate, the in-evident decisions will not be adopted. Thus, in this research, we only use tensor-based score function and MLE method instead of neural networks.
3) We want to enhance the baseline of famous RESCAL model in the domain of KG reasoning and decision-making. Traditional RESCAL model and its score function is used in KG completion task and link prediction task, which lacks a baseline in KG reasoning and decision-making. Compared to RESCAL, Tucker Reasoning model possesses one more information channel, the edge feature matrices. We want to check whether the model can perform better after adding this feature.

The holistic framework is depicted in Figure 2. We propose representing binary decision choices as two decision nodes in a KG. All the decision-related features are regarded as nodes in KG. They can contain the feedback structure in the KG. We consider three type of relations: strong connection, mid connection and weak connection. Then, we apply the Tucker Decomposition to obtain three feature embedding matrices and a core tensor. The three feature matrices store the cause, the effect, and the strength of causality separately. The core tensor store their tacit relationships. Combined with their reliability parameters generated by random walk, we compute the likelihood function L based on the weighted paths to the decision nodes (“Yes” node and “No” node) by maximizing or minimizing the weighted scores according to the sample tags. We then update the parameters in the feature matrices and the core tensor during training. After training, we can predict the binary decisions when there is a new input.

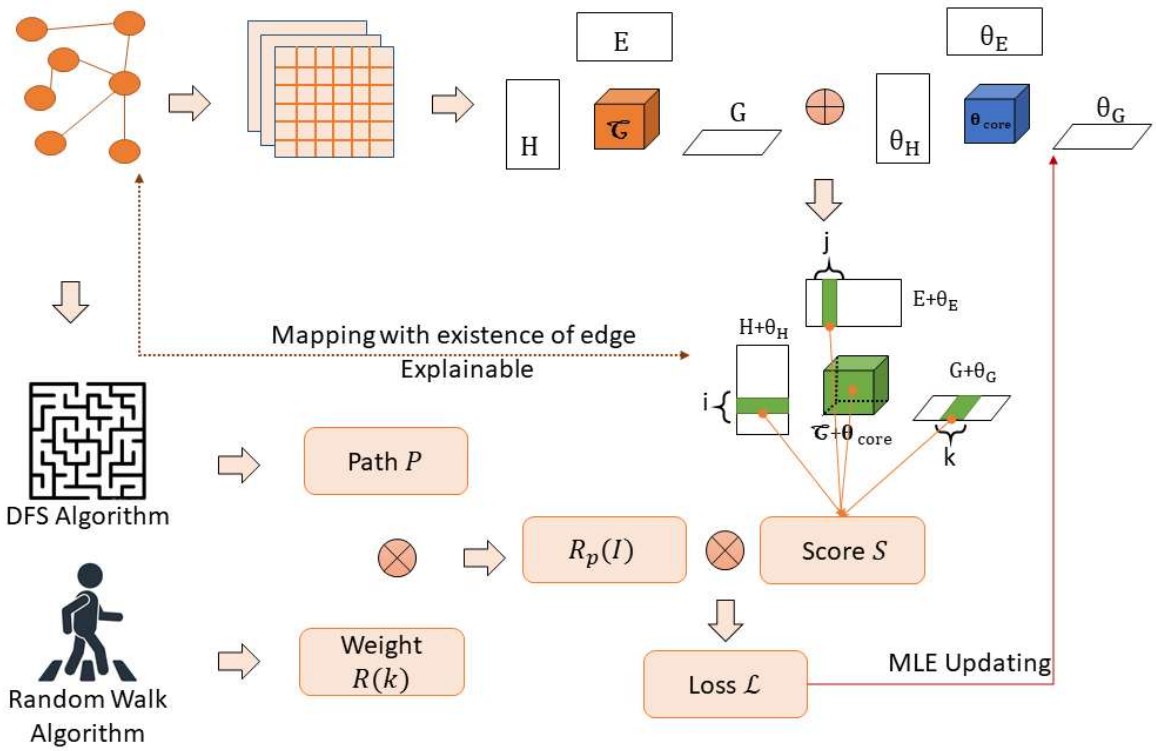


Fig. 2. Tucker-Reasoning learning framework

5. Method

Now we discuss how to learn a model from the observed data, which can help us predict the binary outcomes based on our framework. We term the method as the Tucker Reasoning method. Tucker-Reasoning method aims to establish the score function relying on each edge to ensure the projection is explainable. The likelihood function is generated by composing the score function of each edge on sample paths. In the likelihood function, the decision-targeted correction coefficients are embedded, working as learning parameters. To summarize, the proposed method includes five steps.

1) Decomposing the samples into three features matrices and one core tensor H, E, G, and C.
2) Searching the paths.
3) Calculating the reliability of each path triple.
4) Training the Tucker-Reasoning model.
5) Predicting the binary outcomes according to the trained Tucker-Reasoning model.

We explain each step in details.

1) Sample and decompose the KG:This step aims to obtain initial demonised features, e.g., head node matrix H, tail node matrix E, relation matrix G and core tensor C. In the Tucker-Reasoning method, we use Tucker Decomposition to obtain the feature matrices and a core tensor, which can avoid the pattern noises in learning. In details, we sample batches of tagged samples, some of which support node "Go up" (A) and the others support node "Slump" (B) respectively. Then, we embed all the generated knowledge graphs into tensor form, with three frontal slices representing three types of relationships and each slice encoded as an adjacent matrix. Finally, we calculate the Tucker Decomposition of each sample tensor Z in a batch, obtaining feature matrices H, E, G and core tensor C. We make modifications on

initialization method and core tensor estimation method to obtain better performances (See Algorithm 1).

Algorithm 1 M-TUCKALS3

Input: The tensor data Z

function FEATUREMATRIX(Z)

$H_0, E_0, G_0 \leftarrow$INITIALIZE( )▷uniform initialization.

While U(H,E,G) increase do

$$P_i \leftarrow \mathrm{Z}\left(H_i H_i^T \otimes E_i E_i^T\right)\mathrm{Z}^T$$

$$G_{i+1} \leftarrow P_i G_i \left(G_i^T P_i^2 G_i\right)^{-\frac{1}{2}}$$

$$Q_i \leftarrow \mathrm{Z}\left(E_i E_i^T \otimes G_{i+1} G_{i+1}^T\right)\mathrm{Z}^T$$

$$H_{i+1} \leftarrow Q_i H_i \left(H_i^T Q_i^2 H_i\right)^{-\frac{1}{2}}$$

$$R_i \leftarrow \mathrm{Z}\left(G_{i+1} G_{i+1}^T \otimes H_{i+1} H_{i+1}^T\right)\mathrm{Z}^T$$

$$E_{i+1} \leftarrow R_i E_i \left(E_i^T R_i^2 E_i\right)^{-\frac{1}{2}}$$

$$\mathrm{C} \leftarrow G^T \mathrm{Z}\left(H^T \otimes E^T\right)$$

return H, E, G, C

Since most readers are not familiar with tensor. We provide following introduction to Tucker Decomposition (see Figure 3) and TUCKALS3 algorithm.

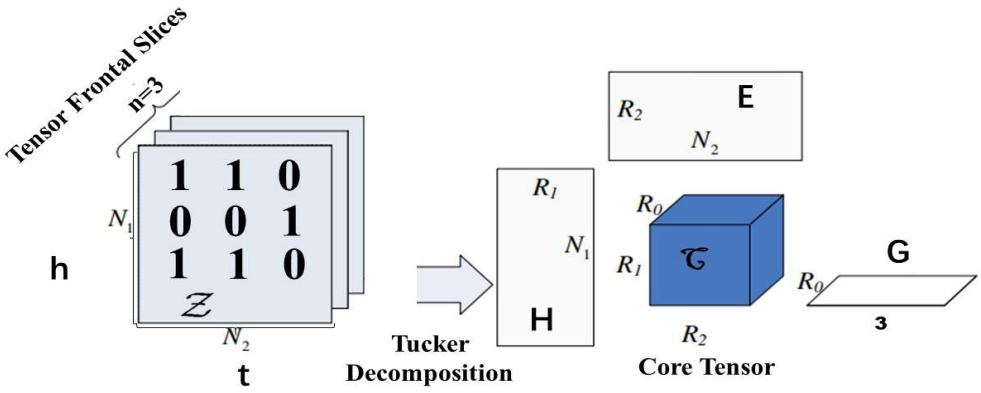


Fig. 3. Tucker Decomposition and its variable notations

In Tucker Decomposition, the whole tensor Z can be decomposed into three feature matrices H, E, G and one core tensor C. Mathematically, the Tucker Decomposition states

$$Z = GC\left(H \otimes E\right) \tag{1}$$

for some H, E, G, and C. Usually, the Tucker Decomposition is not unique. However, under the setting of H, E, G are orthogonal matrices, and the decomposition result is unique.

We have stated that Tucker Decomposition may not be uniquely determined. We, therefore first study the conditions such that the Tucker Decomposition is unique. Let $H \in \mathrm{K}^{m\times p}$, $E \in \mathrm{K}^{n\times q}$, $G \in \mathrm{K}^{l\times o}$, $\mathrm{C} \in \square^{l\times(m\times n)}$, where C is a tensor expanded in matrix form.

To start with, we introduce Lemma 1, which helps study the uniqueness and the best approximation of the Tucker Decomposition.

Lemma 1. $\forall \mathrm{P} \in \mathrm{R}^{l\times(m\times n)}$, we have

$$\begin{aligned}&\left\|G\mathrm{P}\left(H\otimes E\right)-\mathrm{Z}+GG^{T}\mathrm{Z}\left(H^{T}H\otimes E^{T}E\right)\right\|^{2}\\&=\left\|G\mathrm{P}\left(H\otimes E\right)\right\|^{2}+\left\|\mathrm{Z}-GG^{T}\mathrm{Z}\left(H^{T}H\otimes E^{T}E\right)\right\|^{2}\end{aligned} \tag{2}$$

We now move to study the uniqueness and the best approximation of the Tucker Decomposition. Indeed, we have to find H, G, E, and C such that $\left\|G\mathrm{C}\left(H\otimes E\right)-\mathrm{Z}\right\|^{2}$ is minimized. Under the assumptions that $H^{T}H=I$, $E^{T}E=I$, and $G^{T}G=I$, we can show that the best approximation is uniquely given as $\hat{\mathrm{C}}=G^{T}\mathrm{Z}\left(H^{T}\otimes E^{T}\right)$. Lemma 2 summarizes the results.

Lemma 2. Given H, E, and G, $\hat{\mathrm{C}}=G^{T}\mathrm{Z}\left(H^{T}\otimes E^{T}\right)$ is the best approximation solution on parameter C. Besides, it is uniquely determined.

Let $f\left(\mathrm{Z},\hat{Z}\right)=Tr\left[\left(\mathrm{Z}-\hat{Z}\right)^{T}\left(\mathrm{Z}-\hat{Z}\right)\right]$. Substituting $\hat{Z}=G\hat{\mathrm{C}}\left(H\otimes E\right)$ and $\hat{\mathrm{C}}=G^{T}\mathrm{Z}\left(H^{T}\otimes E^{T}\right)$ into $f\left(\mathrm{Z},\hat{Z}\right)$, we can express $f\left(\mathrm{Z},\hat{Z}\right)$ in terms of H, E, G. The corresponding quantity is

$$g\left(H,E,G\right)=Tr\left[\mathrm{Z}^{T}\mathrm{Z}\right]-Tr\left[G^{T}\mathrm{Z}\left(H^{T}H\otimes E^{T}E\right)\mathrm{Z}^{T}G\right] \tag{3}$$

Next, we consider what H, E, G should be such that $g\left(H,E,G\right)$ is minimized. In fact, we have to consider the following optimization problem:

$$\begin{aligned}&\min_{H,E,G} && g\left(H,E,G\right)\\&s.t. && H^{T}H=I,E^{T}E=I,G^{T}G=I\end{aligned} \tag{4}$$

Our objective is finding optimal G, H, and E such that $g\left(H,E,G\right)$ in Eqn. (3) is minimized. Let $U\left(H,E,G\right)=Tr\left[G^{T}\mathrm{Z}\left(H^{T}H\otimes E^{T}E\right)\mathrm{Z}^{T}G\right]$. The optimal solution of Eqn. (3) is the same optimal solution for the following optimization problem:

$$\begin{aligned}&\max_{H,E,G} && U\left(H,E,G\right)\\&s.t. && H^{T}H=I,E^{T}E=I,G^{T}G=I\end{aligned} \tag{4*}$$

The Lagrangian function $\mathrm{V}\left(H,E,G;N,M,L\right)$ corresponding to Eqn. (4*) is

$$\begin{aligned}&U\left(H,E,G\right)-Tr\left[L\left(G^{T}G-I\right)\right]\\&-Tr\left[M\left(H^{T}H-I\right)\right]-Tr\left[N\left(E^{T}E-I\right)\right]\end{aligned} \tag{5}$$

where

$$\begin{aligned} U(H,E,G) &= Tr\left[G^T PG\right] = Tr\left[H^T QH\right] = Tr\left[E^T RE\right] \\ P &= P(H,E) = \mathrm{Z}^T\left(H^T H \otimes E^T E\right)Z \\ Q &= Q(E,G) = \mathrm{Z}^T\left(E^T E \otimes G^T G\right)Z \\ R &= R(H,G) = \mathrm{Z}^T\left(G^T G \otimes H^T H\right)Z \end{aligned} \tag{6}$$

and $L \in \mathrm{S}^{p\times p}$, $M \in \mathrm{S}^{o\times o}$, and $N \in \mathrm{S}^{q\times q}$. According to the Lagrangian function in Eqn.(5), we can obtain the first order conditions of (4*) which are stated as follows:

$$\begin{aligned} Q(E,G)H &= HM \\ R(G,H)E &= EN \\ P(H,E)G &= GL \\ H^T H = I, E^T E = I, G^T G &= I \\ U \le Tr\left[\mathrm{Z}^T \mathrm{Z}\right], M, L, N &\in \mathrm{S}^{p\times p}, \mathrm{S}^{o\times o}, \mathrm{S}^{q\times q} \end{aligned} \tag{7}$$

Our objective is to find $\left(\hat{H},\hat{E},\hat{G},\hat{L},\hat{M},\hat{N}\right)$ which satisfies Eqn. (7). We are going to use the proposed method to obtain the feature tensors. With little modifications, the generated features tensors satisfy Eqn. (7). We summarize the corresponding algorithm in Algorithm 1. In addition, we prove that the modified algorithm can generate feature tensors that satisfy Eqn. (7) in Theorem 1.

Theorem 1. Let $\mathrm{Z} \in \square^{l\times(m\times n)}$, $H \in \mathrm{K}^{m\times p}$, $E \in \mathrm{K}^{n\times q}$, $G \in \mathrm{K}^{l\times o}$, and R, Q, P be defined as in Eqn. (6). Let (H_n, E_n, G_n) be a sequence generated using Algorithm 1. We have:

(1) the generated sequence satisfies Eqn. (7).

(2) the sequence converges to a tuple of limit matrices $S = (H, E, G)$ such that S is a stationary set of $U(H, E, G)$.

2) Search the paths:Generally, we search all the paths of samples and store them (path: one start node and one end node without an extra divergent path), in which each path can support one possible reasoning process. These paths are searched and stored in preparation for reliability calculation, which can finally be accumulated to support decision nodes. In detail, we apply Deep First Search (DFS), a well-defined algorithm, to search paths in samples. This algorithm can search all the paths between arbitrary two nodes. Refer to Richard (1985), we make the traditional DFS be capable of dealing with cyclic graphs by adding line 4 and line 9 (See Algorithm 2).

Algorithm 2 M-DFS

Input: Start node u, end node v, and graph G

```
function M-DFS(u,v,G)
if visited[u] == 1 then                    ▷visited is label vector
    return
Visited [u] ← 1
currentpath.append (u)                     ▷current path is being searching
path, append is function add element to the end of vector
if u == v then
    currentpath1 ← currentpath
    simplepath.append(currentpath1)        ▷output paths
visited [u] ← 0
currentpath.pop()                          ▷pop is function delete and return the last element of vector
return
for next = 0 to range(len(G[u])) do
    if G[u,next] == 1 then
        DFS (next, v, G)
currentpath.pop()
visited [u] ← 0
return simplepath
```

3) Calculate the reliability:Next, we calculate the stable reliability of each reasoning path in samples which requires the reliability of nodes of the graph. If a node is assigned with more in-degrees (i.e., arrows pointing towards the node) and fewer out-degrees (i.e., arrows pointing out to other nodes), then the node is a conclusive node and it should have high reliability. Reasoning paths passing through the conclusive nodes should therefore have higher reliability.

To find the stable reliability of the reasoning paths, we find the stable reliability of the nodes first. Traditional methods require that the graph is irreducible. However, graphs can be reducible in general.

The proposed approach requires the adjacent matrix A of a given graph as an input. The adjacent matrix A in the proposed approach can be obtained as follows. The $(j,i)^{th}$ entry of A is 1 if there is a directed edge pointing from node i to node j, or 0 if no such edges occur. If the node i does not have any out-degrees (i.e., arrows pointing out to other nodes), then the $(j,i)^{th}$ entries for any j are 1. As an illustration, we demonstrate how to obtain A in Example 1.

Example 1. Consider the graph given in Figure 4. Since there are directed edges from node 1 to nodes 2 and 3, and from node 2 to nodes 1 and 3, the entries $(2,1),(3,1),(1,2),(3,2)$ equal 1 and all the other entries are 0. Consequently, the adjacent matrix A^T is $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$.

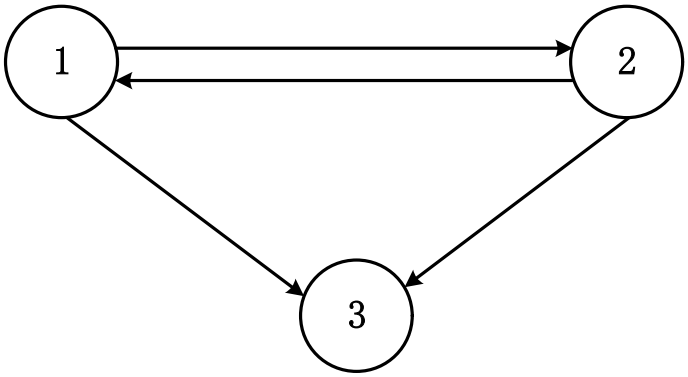


Fig. 4. Calculation of node reliability

After obtaining the adjacent matrix A, we can substitute it into Algorithm 3 to obtain the stable reliability of the nodes.

Algorithm 3 Reliability

Input: A is the adjacent matrix of graph G

Output: The reliability $R(k)$ for each node k in graph G

function RELIABILITY(A)

$D \leftarrow$ COLUMNNORMALIZEMATRIX(A)

$v \leftarrow \frac{1}{n}e$ ▹n is length of column of matrix D

$S \leftarrow \alpha D^T + (1-\alpha)ve^T$ ▹α is damping coefficient that usually is 0.85, e is unit column vector

$k \leftarrow 0$ ▹times of iteration

$e \leftarrow 100{,}000$ ▹Initialize a large e

$S_1 \leftarrow S$

while e < 10e-6 and k < 1000 do

$S_2 \leftarrow S \cdot S_1$

$e \leftarrow \left\| Averg(S_2) - Averg(S_1) \right\|_\infty$

$S_1 \leftarrow S_2$

$k \leftarrow k+1$

return S1,k,e ▹entry in the kth-row of $S1$ is $R(k)$ (The entries are same in same row)

Moreover, $0 < R(k) < 1$. Finally, we can obtain the reliability $R_p(I)$ for each path p whose final node is I which is defined as follows:

$$R_p(I) = \prod_{k \in p} R(k) \tag{8}$$

As an illustration, we demonstrate how to obtain the reliability $R_p(I)$ in Example 2.

Example 2. Consider the graph given in Figure 4 again. Suppose the path p is the path such that $node2 \rightarrow node1 \rightarrow node3$. Then I is $node3$ and $R_p(I) = R(2)R(1)$, where $R(1)$ and $R(2)$ are obtained after substituting A in Example 1 into Algorithm 3

The reliability definition of paths satisfies the property that the longer the reasoning path is, the less reliability of the path is.

Corollary 1. For a directed graph G, if weights of all the vertices $\{v\}$ is summed up to 1, and weights of all the paths $\{p\}$ is defined as product of weights of the v on the p, then the sum weights of all the path is no more than 1.

It is easy to check that the reliability definition of paths (step2-step3) also satisfies Corollary 1.

4) Train the Tucker-Reasoning model:To train the Tucker-Reasoning model, we need to construct the path-based likelihood function. The likelihood function consists of two essential components. The first component is the reliability of paths which has been introduced before. The second component is the edge-based score function.

The score function is used to evaluate the strength of a certain connection. It is defined as

$$S(G_i, \mathrm{C}, H_j, E_k) = G_i \mathrm{C}(H_j \otimes E_k) \tag{9}$$

Here, G_i is the i^{th} -row of G, H_j is the j^{th} -row of H, and E_k is the k^{th} -row of E. We believe the reason node feature matrix should not share the same space with the result node feature matrix, though

they are the same entity. Since certain node is only reason faced with certain node, the compressed storage alternatives is not good in multiple step reasoning tasks.

After defining the score function, we can define the path-based likelihood function. Let $\varphi(x)=\frac{1}{1+e^{-x}}$ be the sigmoid function. Moreover, let $\theta_G=\left[\theta_{G_1}^T,\cdots,\theta_{G_l}^T\right]^T$, $\theta_H=\left[\theta_{H_1}^T,\cdots,\theta_{H_m}^T\right]^T$, and $\theta_E=\left[\theta_{E_1}^T,\cdots,\theta_{E_n}^T\right]^T$ which are the hyper-parameters. The likelihood function L (which depends on $(\theta_H,\theta_E,\theta_G,\theta_{core})$) is given as follows:

$$\mathrm{L}=\sum_{I\in A,B,p\in\Pi_I}1(I)R_p(I)\log\prod_{\substack{(G_i,H_j,E_k)\in\{(X(w),\\ Y(w),T(w))|w\in p\}}}\varphi(\mathrm{B}) \tag{10a}$$

where

$$\mathrm{B}=S\left(G_i+\theta_{G_i},\mathrm{C}+\theta_{core},H_j+\theta_{H_j},E_k+\theta_{E_k}\right)$$
$$1(I)=\begin{cases}1 & I=A\\ -1 & I=B\end{cases} \tag{10b}$$

Here, Π_I is the path set supporting the decision node I. A and B are two decision nodes. w is a certain edge on path p. $\mathrm{X}(w)$, $\mathrm{Y}(w)$ and $\mathrm{T}(w)$ are extraction functions for matrices G, H and E. For example, for a certain edge w, if its position in Z is $[2,4,5]$, $\mathrm{X}(w)$ obtains G_2, $\mathrm{Y}(w)$ obtains H_4 and $\mathrm{T}(w)$ obtains E_5. The parameters are correction coefficients that diminish unrelated patterns in random samples, e.g., θ_G is in the same size as G, and θ_{core} is in the same size as C.

Our objective is to find the optimal hyper-parameters which can distinguish the decisions. To update the parameters, we need to derive the gradients. For tagged A samples, $\mathrm{L}(\theta_H,\theta_E,\theta_G,\theta_{core})$ is maximized over $\theta_H,\theta_E,\theta_G,\theta_{core}$. For tagged B samples, $\mathrm{L}(\theta_H,\theta_E,\theta_G,\theta_{core})$ is minimized over $\theta_H,\theta_E,\theta_G,\theta_{core}$.

Let θ_{lmn} represent the $[l,m,n]^{th}$ element in θ_{core}.

$$\frac{\partial\mathrm{L}}{\partial\theta_{lmn}}=\sum_{I\in A,B,p\in\Pi_I}1(I)R_p(I)\sum_{i,j,k}\frac{\sigma_{ijk}(1-\sigma_{ijk})}{\sigma_{ijk}}\frac{\partial S_{ijk}}{\partial\theta_{lmn}} \tag{11}$$

$$\sigma_{ijk}=\varphi\left(G_i(\mathrm{C}+\theta_{core})(H_j\otimes E_k)\right) \tag{12}$$

$$\frac{\partial S_{ijk}}{\partial\theta_{lmn}}=\sum_{i,j,k}G_{i,l}H_{j,m}E_{k,n} \tag{13}$$

The update method is as follows:

$$\theta_{lmn}\leftarrow\theta_{lmn}\pm\frac{\partial\mathrm{L}(\theta_H,\theta_E,\theta_G,\theta_{core})}{\partial\theta_{lmn}}\eta \tag{14}$$

where η is the step size. Tagged A sample training uses '+,' and tagged B sample training uses '−'.

Obtaining optimal θ_{core}^* in one batch learning, we use C^* to represent $\mathrm{C}+\theta_{core}^*$ and derive $S_{ijk}=\left(G_i+\theta_{G_i}\right)\mathrm{C}^*\left[\left(H_j+\theta_{H_j}\right)\otimes\left(E_k+\theta_{E_k}\right)\right]$,

$$\nabla_{\theta_{G_i}} S_{ijk} = \left[\left(H_j + \theta_{H_j}\right) \otimes \left(E_k + \theta_{E_k}\right)\right]^T C^{*T} \tag{15}$$

$$\nabla_{\theta_{H_j}} S_{ijk} = \begin{bmatrix} \sum_{q=1}^{n}\sum_{s=1}^{l}\left[G_i + \theta_{G_i}\right]_s C^*_{s1q}\left[E_k + \theta_{E_k}\right]_q \\ \vdots \\ \sum_{q=1}^{n}\sum_{s=1}^{l}\left[G_i + \theta_{G_i}\right]_s C^*_{smq}\left[E_k + \theta_{E_k}\right]_q \end{bmatrix} \tag{16}$$

$$\nabla_{\theta_{E_k}} S_{ijk} = \begin{bmatrix} \sum_{p=1}^{m}\sum_{s=1}^{l}\left[G_i + \theta_{G_i}\right]_s C^*_{sp1}\left[H_j + \theta_{H_j}\right]_p \\ \vdots \\ \sum_{p=1}^{m}\sum_{s=1}^{l}\left[G_i + \theta_{G_i}\right]_s C^*_{spn}\left[H_j + \theta_{H_j}\right]_p \end{bmatrix} \tag{17}$$

$$\frac{\partial \mathrm{L}}{\partial \theta_{G_i}} = \sum_{I \in A,B} 1(I) R_p(I) \sum_{i,j,k}\left(1 - \sigma_{ijk}\right) \nabla_{\theta_{G_i}} S_{ijk} \tag{18}$$

$$\frac{\partial \mathrm{L}}{\partial \theta_{H_j}} = \sum_{I \in A,B} 1(I) R_p(I) \sum_{i,j,k}\left(1 - \sigma_{ijk}\right) \nabla_{\theta_{E_k}} S_{ijk} \tag{19}$$

$$\frac{\partial \mathrm{L}}{\partial \theta_{H_j}} = \sum_{I \in A,B} 1(I) R_p(I) \sum_{i,j,k}\left(1 - \sigma_{ijk}\right) \nabla_{\theta_{E_k}} S_{ijk} \tag{20}$$

Then, we use

$$\theta_{G_i} \leftarrow \theta_{G_i} \pm \frac{\partial \mathrm{L}}{\partial \theta_{G_i}} \eta \tag{21}$$

$$\theta_{H_j} \leftarrow \theta_{H_j} \pm \frac{\partial \mathrm{L}}{\partial \theta_{H_j}} \eta \tag{22}$$

$$\theta_{E_k} \leftarrow \theta_{E_k} \pm \frac{\partial \mathrm{L}}{\partial \theta_{H_j}} \eta \tag{23}$$

to update θ_{G_i} , θ_{H_j} , and θ_{E_k} . Again, Tagged A sample training uses ‘ + ,’ and tagged B sample training uses ‘ – ’.

In addition, if RESCAL is applied the gradient is as follows:

$$\frac{\partial S_{ijk}}{\partial \theta_{R(k)}} = A_j^T A_i \tag{24}$$

$$\nabla_{\theta_{A_i}} S_{ijk} = \sum_{j,k} \theta_{A_j} R(k)^T \tag{25}$$

$$\nabla_{\theta_{A_j}^T} S_{ijk} = \sum_{i,k} R(k)^T \theta_{A_i}^T \tag{26}$$

Therefore, we maximize or minimize the function L according to whether it is sample batch tagged node A or B. we update the parameters with Algorithm 4.

Algorithm 4 Train

Input: $Z, \theta_G, \theta_H, \theta_E, \theta_{core}, \zeta$ ▷The parameters scale can be chosen according to demands with restricted to not surpassing the original dimensions in Z

function TRAIN()

M-TUCKALS3(Z)

M-DFS()

RELIABILITY()

if Tagged A sample then

$$\theta_{lmn} \leftarrow \theta_{lmn} + \frac{\partial \mathrm{L}\left(\theta_H, \theta_E, \theta_G, \theta_{core}\right)}{\partial \theta_{lmn}} \eta$$

$$\theta_{H_j} \leftarrow \theta_{H_j} + \frac{\partial \mathrm{L}}{\partial \theta_{H_j}} \eta$$

$$\theta_{E_k} \leftarrow \theta_{E_k} + \frac{\partial \mathrm{L}}{\partial \theta_{E_k}} \eta$$

$$\theta_{G_i} \leftarrow \theta_{G_i} + \frac{\partial \mathrm{L}}{\partial \theta_{G_i}} \eta$$

if Tagged B sample then

$$\theta_{lmn} \leftarrow \theta_{lmn} - \frac{\partial \mathrm{L}\left(\theta_H, \theta_E, \theta_G, \theta_{core}\right)}{\partial \theta_{lmn}} \eta$$

$$\theta_{H_j} \leftarrow \theta_{H_j} - \frac{\partial \mathrm{L}}{\partial \theta_{H_j}} \eta$$

$$\theta_{E_k} \leftarrow \theta_{E_k} - \frac{\partial \mathrm{L}}{\partial \theta_{E_k}} \eta$$

$$\theta_{G_i} \leftarrow \theta_{G_i} - \frac{\partial \mathrm{L}}{\partial \theta_{G_i}} \eta$$

return $\theta_G, \theta_H, \theta_E, \theta_{core}$

5) Predict with the Tucker-Reasoning model:This step aims to give predictions based on comparing the number of weighted scores supporting two decision nodes. In detail, we derive the parameters into function L of the prediction tensor. If the function value exceeds the threshold ζ , make a specific decision, and vice versa (Algorithm 5).

Algorithm 5 Predict

Input: $\mathrm{Z}, \theta_{\mathbf{G}}, \theta_{\mathbf{H}}, \theta_{\mathbf{E}}, \theta_{core}, \zeta$

function PREDICT()

M-TUCKALS3(Z)

M-DFS()

RELIABILITY()

if $\mathrm{L}(G+\theta_G, H+\theta_H, E+\theta_E, C+\theta_{core} \geq \zeta$ then

Predict decision A

if $\mathrm{L}(G+\theta_{\mathbf{G}}, H+\theta_{\mathbf{H}}, E+\theta_{\mathbf{E}}, C+\theta_{core} < \zeta$ then

Predict decision B

9: return Decision

6. Numerical experiments and findings

In this section, we will undergo two experiments: Experiment 1 is under no-conflict settings, and Experiment 2 is under conflict settings. The conflict is assumed as the main variety in the two experiments, and the prediction noise is assumed as the secondary variety. Before describing the numerical experiments, Definition 1 is introduced.

Definition 1. If $\exists e \in \varepsilon(G_1) \cap \varepsilon(G_2)$, then G_1, G_2 are conflict, where ε is edge set function and G is knowledge graph, e is certain edge.

In general, if G_1, G_2 are in conflict, they must share a certain same directed edge.

Example 3. The edge from node 8 to conclusive node B exists in both sub-figures. Thus, the two sub-figures are in conflict.

6.1. Data preparation

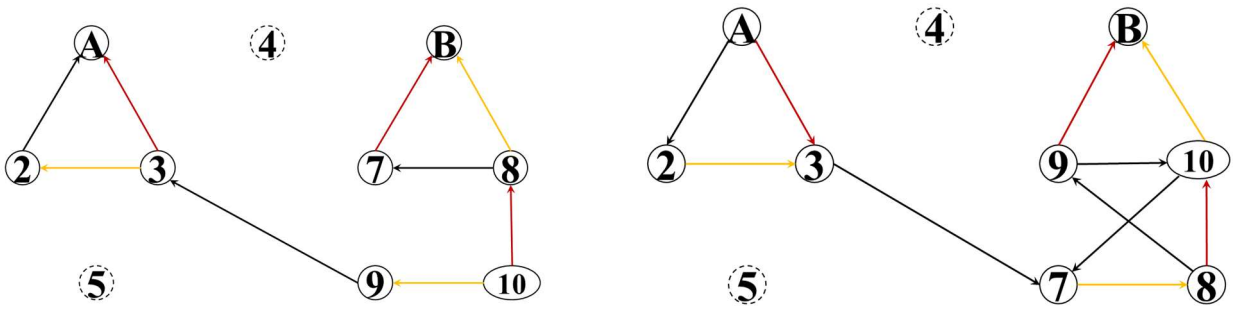
This study uses simulation-generated data to take the place of real datasets. The holistic idea is to add noise connection between nodes on the upper layer KG. For the high noise case, the connection probability is 0.3. For the low noise case, the connection probability is 0.2. We use these data to train the Tucker Reasoning model and to recognize the real upper layer KG pattern that is without noise. In the following experiments, the upper layer KG embedded tensors are in size of $3 \times 10 \times 10$. Each epoch contains 100 noise-added samples as the training set. Simultaneously, the prediction sets are generated in the same way. We choose the best performance epoch of each model as the final results. The upper layer KG is named as the simulation generation base in the following subsection.

6.2. Experiment 1 settings

In Experiment 1, the simulation uses no-conflict graphs in Figure 5. The left one is a simulation generation base tagged supporting node A. The right one is a simulation generation base tagged supporting node B. The training noise is from node 4 and node 5. The other nodes are randomly connected to nodes 4 and 5 on a certain probability to generate different training samples.

The aim of Experiment 1 is to check the prediction effects of the proposed method on the condition of no-conflict. With each batch containing 5 example tensors, Experiment 1 trains 40 batches of samples, 20 tagged supporting node A and 20 tagged supporting node B. The tagged A batch and tagged B batch are trained A-B-A-B alternately. The optimization objective function is the total weighted scores in each batch. Moreover, the gradient is the average of gradients in a batch.

High and low noise are added by using large or small probability to connect arbitrary two nodes while predicting.

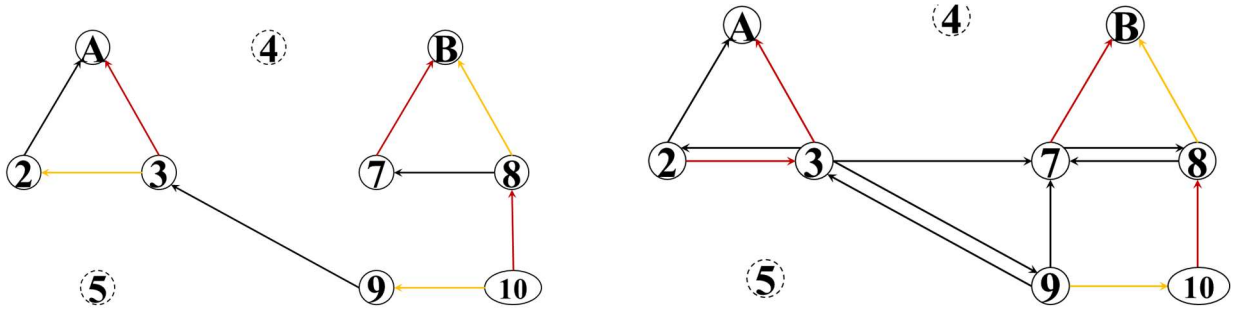


(a) Tagged A sample base (b) Tagged B sample base

Fig. 5. Simulation generation base on no-conflict condition

6.3. *Experiment 2 settings*

In Experiment 2, the simulation uses conflict graphs in Figure 6. The left one is used as a simulation generation base tagged supporting node A. The right one is a simulation generation base tagged supporting node B.



(a) Tagged A sample base (b) Tagged B sample base

Fig. 6. Simulation generation base on conflict condition, tagged A on the left and B on the right

With all the settings are similar to Experiment 1, the aim of Experiment 2 is to check the prediction effects of the proposed method on the condition of conflict. It is found that the prediction performances under low or high noise are worse than those in Experiment 1. This is because, on the condition of conflict graph learning, the tagged A sample shares the same parameters with the tagged B sample, while the former is maximizing the objective function and the latter is minimizing the objective function. The classifier effects can be reduced under this condition.

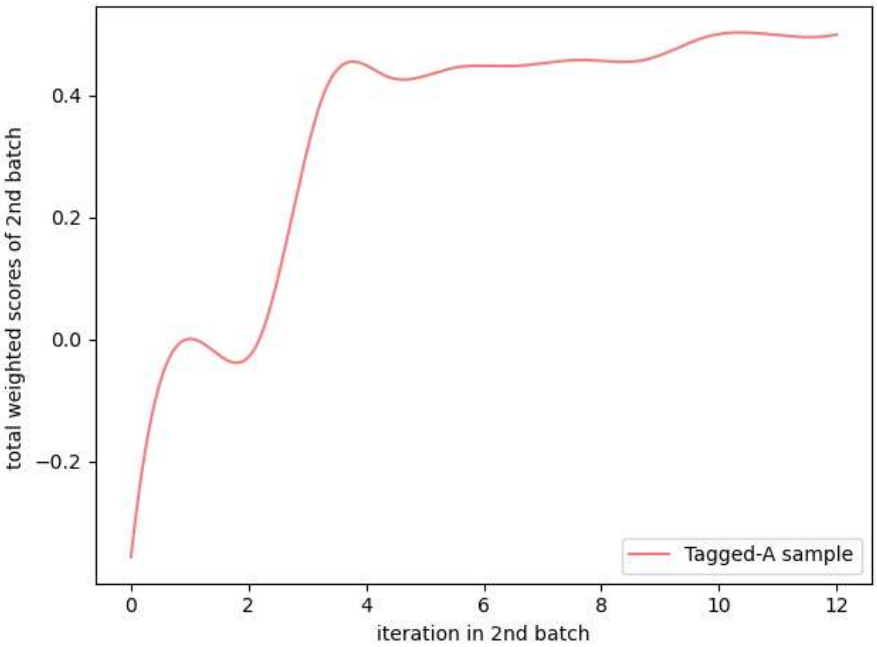
6.4. *Parameter Settings*

During conducting the above experiments, the RESCAL model is also applied to be compared with. The parameter scale of θ and step size η are two hyper-parameters in both RESCAL model and our model. In the decomposition step, Tucker Reasoning's core tensor size is tuned as $3\times6\times6$, thus its parameters θ_{core}, θ_G, θ_H, θ_E have scale of 237. For the RESCAL model, the tuned $R(k)$ is in size of 8×8, thus its parameters θ_R, θ_A have scale of 272. They are almost in the same scale, with RESCAL having a little bit more parameters.

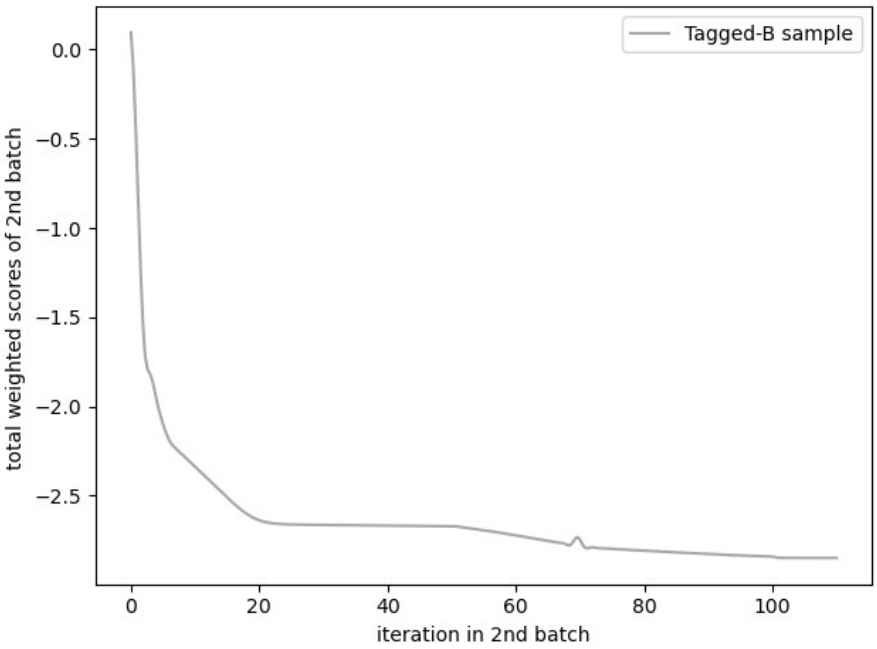
6.5. *Discussions*

The discussions cover the following contents: the convergence of Tucker-Reasoning method, the ROC curve analysis of Tucker-Reasoning method, the P-R curve analysis of Tucker-Reasoning method, Precision & Recall & F1 analysis of Tucker-Reasoning method, and reasoning path score analysis of Tucker-Reasoning method. The last 3 analysis is compared with the baseline of RESCAL model.

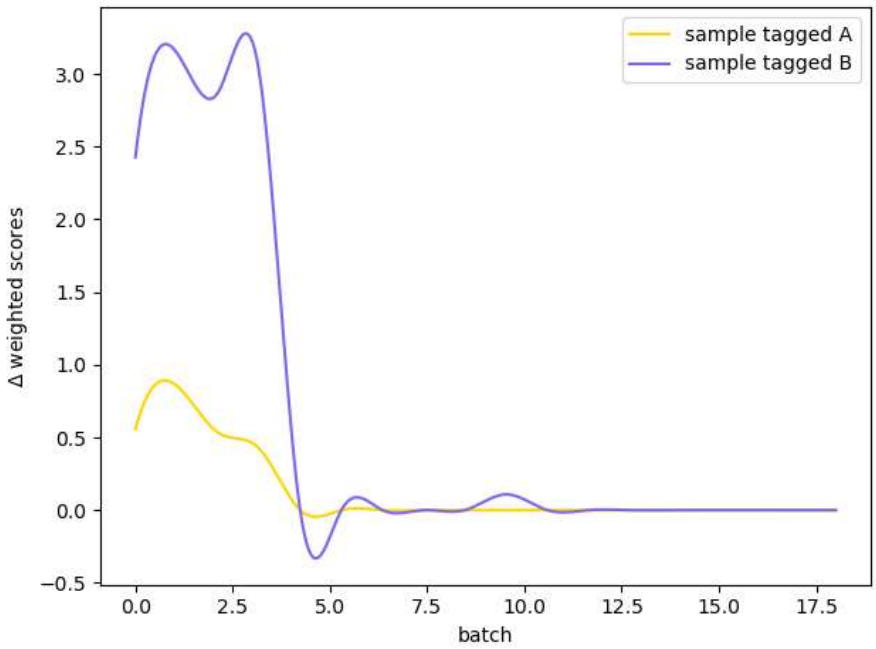
First, our algorithm can converge (See Figure 7). In the subfigure (a), the likelihood value of tagged A samples attain certain upper bound after 12 steps. In the subfigure (b), the likelihood value of tagged B samples become stable after 20 steps. From subfigure (c), Δ weighted scores become 0 in the end. Thus, the proposed method is convergent and stable.



(a) A-sample learning in one batch



(b) B-sample learning in one batch



(c) Δ weighted scores in learning

Fig. 7. Learning process

Second, the ROC curve analysis is applied for comparisons, in which a more covered area indicates a higher AUC value and better model performance. In Figure 8, it is evident that the proposed method obtains acceptable results under no-conflict and low-noise conditions. However, when the settings change into conflict, all the experiment sets obtain bad results, which indicates that the conflict setting is the main experiment variety. The red line coverage area representing RESCAL performance under no-conflict and low-noise condition is 0.12 lower than our proposed model. It is proved that our model is better under same conditions and the objective 3 is achieved.

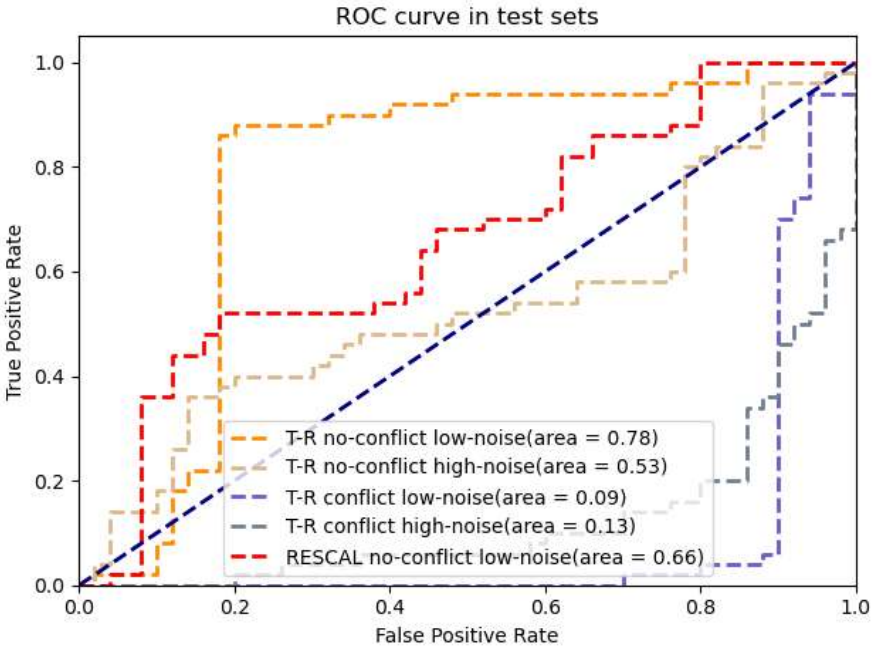


Fig. 8. ROC curve in Experiment 1 and 2

The PR curve analysis is also applied in comparisons, in which the bigger Break-Event Point indicates the higher model performance. In Figure 9, the orange P-R curve possesses the biggest cross point with $y = x$. From the figure, we can see that the precision results of Experiment 1 are better than Experiment 2. Moreover, the results of low prediction noise settings perform better than those of high prediction noise settings. Also, the proposed model outperform 0.07 than the baseline model of RESCAL.

In Table 3, the F1 index attains 0.84 on the condition of no-conflict and low prediction noise, indicating that the Tucker-Reasoning method is generally acceptable. However, in Experiment 2, the F1 index is 0.08 and 0.23, respectively, which indicates that in the condition of conflict, no matter how high the noise is, the final results are usually not acceptable. Otherwise, under same conditions, the F1 index of RESCAL attains 0.58, which highlights our contributions to this area.

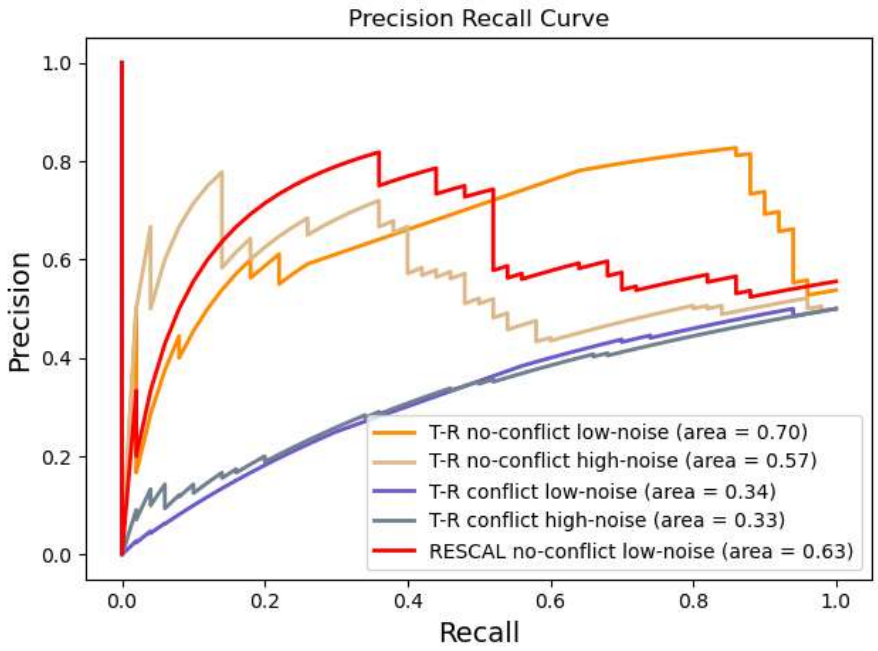


Fig. 9. PR Curve in Experiment 1 and 2

Table 3. Experiments results

Experiment 1	T-R no-conflict		
	P(Macro)	R(Macro)	F1(Macro)
low-noise	0.84	0.84	0.84
high-noise	0.61	0.65	0.58
Experiment 2	T-R conflict		
	P(Macro)	R(Macro)	F1(Macro)
low-noise	0.08	0.08	0.08
high-noise	0.24	0.23	0.23
Baseline	RESCAL no-conflict and low-noise		
	P(Macro)	R(Macro)	F1(Macro)
	0.58	0.58	0.58

Finally, it is easy to see the score of each path in prediction, which can help point out which path is the main reason (See Table 4). Thus, the objective 1 is achieved. Simultaneously, the proposed model avoids to use neural networks in the whole process. Thus, it indicates that our objective 2 is achieved. Tucker reasoning is explainable and reasonable.

Table 4. Example of reasoning paths and related score

Path	Score
2→A	2.68×10^{-8}
3→A	–0.0570
3→2→A	7.10×10^{-9}
9→3→A	–0.0086
9→3→2→A	1.60×10^{-9}
10→9→3→A	–0.0047
10→9→3→2→A	–0.0002
7→B	2.04×10^{-8}
8→B	4.95×10^{-8}
8→7→B	5.40×10^{-9}
10→8→B	-2.75×10^{-8}
10→8→7→B	2.25×10^{-9}

To summarize, the proposed Tucker-Reasoning method is more sensitive to the conflict setting than high noise setting. Compared to RESCAL model, the proposed method can make good predictions and obtain excellent AUC results.

7. Conclusions

To make explainable and reasonable decisions based on upper layer KG, we build a framework called Tucker Reasoning Learning Method. In comparison to traditional decision-making methods in the knowledge graph domain, our method has advantages on accuracy, fixed scale, explainability, and providing reasoning paths.

Our theoretical contribution lies in inventing a new embedding score function of KG edge that is pair-wisely mapped with existence of each edge and used to form decision making objective function. This edge score function contain edge feature matrices information and this information make sense in measuring edge reasoning strength, which affects decision-makings.

In practical applications, our proposed method has the potential to solve real-life decision-making problem in automatic justice based on articles of laws, deal decision based on financial blog texts, and smart diagnosis based on clinical records.

The proposed method shows promising results but can only address decision-making problems under the assumption of no-conflict KGs. As a limitation of this research, we intend to further study conflict conditions and develop approximation methods to address them in future work.

Acknowledgements

This work is outcome of "Research on the Mechanism of Intelligent Decision-making and Standardized Governance for the Scaled Operation of Data Authorization and Operation Platform" (2462024YJRC002); and supported by Science Foundation of China University of Petroleum, Beijing (No. ZX20240073).

References

[1] Shen, T., Zhang, F., & Cheng, J. (2022). A comprehensive overview of knowledge graph completion. Knowledge-Based Systems, 255, 109597.

[2] Tian, L., Zhou, X., Wu, Y. P., Zhou, W. T., Zhang, J. H., & Zhang, T. S. (2022). Knowledge graph and knowledge reasoning: A systematic review. Journal of Electronic Science and Technology, 20(2), 100159.

[3] Liang, K., Meng, L., Liu, M., Liu, Y., Tu, W., Wang, S., ... & He, K. (2024). A survey of knowledge graph reasoning on graph types: Static, dynamic, and multi-modal. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[4] Li, D., & Madden, A. (2019). Cascade embedding model for knowledge graph inference and retrieval. Information Processing & Management, 56(6), 102093.

[5] Rajabi, E., & Kafaie, S. (2022). Knowledge graphs and explainable ai in healthcare. Information, 13(10), 459.

[6] Wang, W., Xu, Y., Du, C., Chen, Y., Wang, Y., & Wen, H. (2021). Data set and evaluation of automated construction of financial knowledge graph. Data Intelligence, 3(3), 418-443.

[7] Chen, P., Lu, Y., Zheng, V. W., Chen, X., & Yang, B. (2018). Knowedu: A system to construct knowledge graph for education. Ieee Access, 6, 31553-31563.

[8] Ge, R., & Ma, T. (2017). On the optimization landscape of tensor decompositions. Advances in Neural Information Processing Systems, 30.

[9] Malik, O. A., & Becker, S. (2018). Low-rank tucker decomposition of large tensors using tensorsketch. Advances in neural information processing systems, 31.

[10] Gao, W., Ma, Z., & Yuan, X. (2022). Dimensionality reduction algorithm of tensor data based on orthogonal tucker decomposition and local discrimination difference. Applied Intelligence, 52(12), 14518-14540.

[11] Wahba, A., Wang, L. C., Zhang, Z., & Sumikawa, N. (2019, April). Wafer pattern recognition using tucker decomposition. In 2019 IEEE 37th VLSI Test Symposium (VTS) (pp. 1-6). IEEE.

[12] Cai, J., Cao, Z., & Zhang, L. (2020). Learning a single tucker decomposition network for lossy image compression with multiple bits-per-pixel rates. IEEE Transactions on Image Processing, 29, 3612-3625.

[13] Gillet, A., Leclercq, É., & Sautot, L. (2023). A Guide to the Tucker Tensor Decomposition for Data Mining: Exploratory Analysis, Clustering and Classification. In Transactions on Large-Scale Data-and Knowledge-Centered Systems LIV: Special Issue on Data Management-Principles, Technologies, and Applications (pp. 56-88). Berlin, Heidelberg: Springer Berlin Heidelberg.