Searching to Sparsify Tensor Decomposition for N-ary Relational Data

Shimin DI
The Hong Kong University of Science and Technology
Hong Kong SAR, China
sdiaa@cse.ust.hk

Quanming YAO
4Paradigm Inc.
Tsinghua University
Beijing, China
yaoquanming@4paradigm.com

Lei CHEN
The Hong Kong University of Science and Technology
Hong Kong SAR, China
leichen@cse.ust.hk

ABSTRACT

Tensor, an extension of the vector and matrix to the multi-dimensional case, is a natural way to describe the N-ary relational data. Recently, tensor decomposition methods have been introduced into N-ary relational data and become state-of-the-art on embedding learning. However, the performance of existing tensor decomposition methods is not as good as desired. First, they suffer from the data-sparsity issue since they can only learn from the N-ary relational data with a specific arity, i.e., parts of common N-ary relational data. Besides, they are neither effective nor efficient enough to be trained due to the over-parameterization problem. In this paper, we propose a novel method, i.e., S2S, for effectively and efficiently learning from the N-ary relational data. Specifically, we propose a new tensor decomposition framework, which allows embedding sharing to learn from facts with mixed arity. Since the core tensors may still suffer from the over-parameterization, we propose to reduce parameters by sparsifying the core tensors while retaining their expressive power using neural architecture search (NAS) techniques, which can search for data-dependent architectures. As a result, the proposed S2S not only guarantees to be expressive but also efficiently learns from mixed arity. Finally, empirical results have demonstrated that S2S is efficient to train and achieves state-of-the-art performance.

KEYWORDS

Knowledge Graph, N-ary Relational Data, Tensor Decomposition, Neural Architecture Search

ACM Reference Format:
Shimin DI, Quanming YAO, and Lei CHEN . 2021. Searching to Sparsify Tensor Decomposition for N-ary Relational Data. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449853

1 INTRODUCTION

As an important way to explore and organize human knowledge, web-scale knowledge bases (KBs, i.e., N-ary relational data) [2, 6, 31] has promoted a series of web applications, e.g., semantic search [41], question answering [27], and recommendation system [9, 47]. Generally, the N-ary relational data contains n-ary facts, that is formed by n entities with a relation r such as $(r, e_1, \cdots, e_n)$ (i.e., arity is n). For example, *playedCharacterIn* is one of common 3-ary relations, involved with an actor, a character, and a movie in a 3-ary fact (*playedCharacterIn, LeonardNimoy, Spock, StarTrek 1*). Given a fact, the link prediction task is one of the crucial tasks in the N-ary relational data, which is to verify whether a fact is plausible or not. Previous studies [8, 10, 19, 44, 50] focus on handling the link prediction task on a special case of the N-ary relational data, knowledge graphs (KGs, i.e., binary relational data) [28, 36]. Recently, how to handle the general N-ary relational data has attracted lots of attention [13–15, 30, 39, 48]. Firstly, it is essential to handle hyper-relational facts (i.e., n-ary facts with $n > 2$) because they are very common in KBs. It has been reported in [39] that more than 30% of the entities in Freebase [6] involves in the hyper-relational facts. Moreover, the facts with high-arity may provide benefits in the question answering scenario [12] since it usually contains more complete information compared with binary facts.

Many models have been proposed to tackle the link prediction task on the N-ary relational data. The translational distance models m-TransH [39] and RAE [48] extend a well-known method TransH [38] from binary to the n-ary scenario. But TransH cannot handle certain relations [19, 32]. Thus, it is regarded as inexpressive since a fully expressive model should be able to handle arbitrary relation patterns on the binary case [19]. Consequently, m-TransH and RAE are also not expressive. However, the expressive ability largely determines the performance of embedding models. Thus, the expressiveness of translational distance models worsens their performance in the case of N-ary relational data. Furthermore, the neural network models, NaLP [15], HINGE [30], and NeuInfer [14], achieve good performance by employing complex neural networks to learn embeddings. But they all introduce an enormous amount of parameters, which contradicts the linear time and space requirement in knowledge bases [7].

Tensor decomposition models [3, 25] introduce a natural way to model N-ary relational data with a $(n + 1)$-order tensor and become state-of-the-art because of their expressiveness. TuckER [3] proposes to model the binary relational data with a 3-order tensor and then decomposes it for embedding learning. It is easy to extend TuckER from binary to high-arity relational data by modeling n-ary facts with a high-order tensor, named n-TuckER [3, 25]. However, such a simple extension will lead to the curse of dimensionality due to the large size of the core tensor. Therefore, GETD [25] simplifies the core tensor with Tensor Ring Decomposition [51] to reduce the model complexity. Then, GETD achieves outstanding performance

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449853

Table 1: Summary of existing n-ary works. Whether a scoring function is expressive depends on its capability of handling common relation patterns as in [50]. The Mixed-arity indicates whether a model jointly learn from the N-ary relational data with mixed arity. N is the maximum arity of facts. n_e and n_r are the number of entities and relations, respectively. d_e and d_r denote the dimensionality of embeddings on entity and relation, respectively. And $d_{\max} = \max_i d_i$ with $\prod_{i=1}^{c} d_i = d_e^n d_r$ in GETD [25]. The time is the computational cost of calculating the score of the single n-ary fact based on $d = d_e = d_r$.

Type	Models	Effectiveness		Efficiency	
		Expressive	Mixed-arity	Time	Space
Translational Models	m-TransH [39]	×	✓	$O(d)$	$O(n_e d_e + n_r d_r)$
	RAE [48]	×	✓	$O(d^2)$	$O(n_e d_e + n_r d_r)$
Neural Network Models	NaLP [15]	unknown	✓	$O(d^2)$	$O(n_e d_e + N n_r d_r)$
	HINGE [30]	unknown	✓	$O(d^2)$	$O(n_e d_e + N n_r d_r)$
	NeuInfer [14]	unknown	✓	$O(d^2)$	$O(n_e d_e + N n_r d)$
Tensor Decomposition Models	n-TuckER [25]	✓	×	$O(d^{n+1})$	$O(n_e d_e + n_r d_r + d_e^n d_r)$
	GETD [25]	✓	×	$O(d^3)$	$O(n_e d_e + n_r d_r + c d_{\max}^3)$
	S2S	✓	✓	$O(d)$	$O(n_e d_e + n_r d_r)$

in the N-ary relational data because of less model complexity and expressive guarantee [25].

However, existing tensor decomposition models for N-ary relational data still suffer from two issues: *data sparsity* and *over-parameterization*. First, it is well-known that the N-ary relational data is very sparse, which is difficult for training and learning [29]. But existing tensor decomposition models [3, 25] can only learn embeddings from facts with a specific arity n, while the N-ary relational data usually contains facts with different arities [30, 39]. In other words, tensor decomposition models cannot leverage all known facts of the given N-ary relational data, which causes the data sparsity issue to become even more severe. Second, current tensor decomposition models achieve the expressive capability by maintaining an over-parameterized core tensor, even GTED requires cubic model complexity. Such over-parameterization for expressiveness not only makes the model inefficient but also difficult to train. We summarize the above existing models for N-ary relational data in Table 1. We first compare the two main factors that affect the effectiveness of current models, the expressive capability, and whether the model can learn from facts with mixed arity. Then, to demonstrate whether the model requires a large number of parameters, we compare their efficiency from the infer time and size of parameter space. Obviously, none of the existing works can cover all the aspects.

This paper aims to alleviate the data sparsity and over-parameterization issues of existing tensor decomposition models for n-ary relation data learning. To handle the data sparsity issue, we propose to partially share embeddings across arities and jointly learn embeddings from the N-ary relational data with mixed arity. Then, motivated by the structurally sparse patterns discovered from existing tensor models on binary relational data and the success of neural architecture search (NAS) [18, 45] on designing data-specific deep networks, we search to sparsify the dense core tensors using NAS techniques to avoid over-parameterization. In this way, we address the issues of data sparsity and over-parameterization while retaining the expressiveness of tensor models.

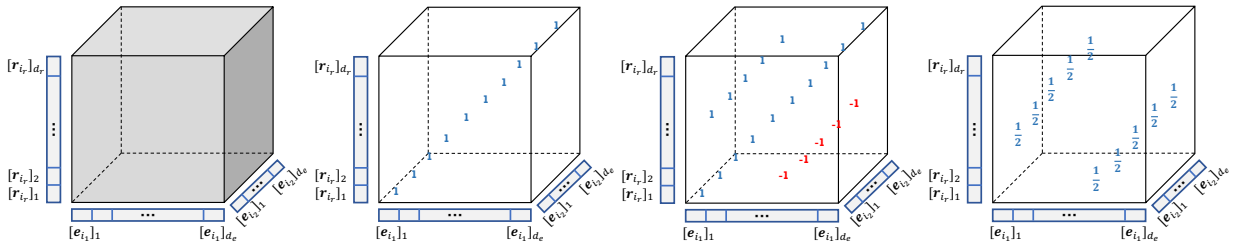
We summarize the important notations in Table 2, and our contributions are listed as follows:

- We propose a new model, i.e., S2S, to learn from N-ary relational data, which simultaneously addresses the data-sparsity and over-parameterization issue faced by existing tensor decomposition models.
- To capture the data-specific knowledge, we propose a novel approach to search for multiple sparse core tensors, which are utilized to jointly learn from any given N-ary relational data with mixed arity.
- We test the proposed model on the link prediction task in both binary and N-ary relational data. Experimental results show that S2S not only achieves outstanding performance in embedding learning but also improves efficiency.

2 RELATED WORKS

Recently, many tensor decomposition approaches have been introduced to describe the N-ary relational data [3, 19, 24, 25, 34, 44]. Specifically, given facts with a specific arity n, a $(n+1)$-order tensor $\mathcal{X} \in \{0,1\}^{n_r \times n_e \times \cdots \times n_e}$ is utilzed to represent a N-ary relational data, where $\mathcal{X}_{i_r, i_1, \ldots, i_n} = 1$ represents an existing fact $(r_{i_r}, e_{i_1}, \cdots, e_{i_n})$ otherwise $\mathcal{X}_{i_r, i_1, \ldots, i_n} = 0$. For instance, binary relational data (i.e., $n = 2$) is represented into 3-order tensor $\mathcal{X} \in \{0,1\}^{n_r \times n_e \times n_e}$. Then, different tensor decomposition models differ in how the tensor $\mathcal{X}$ is decomposed into the entity embedding $\boldsymbol{E} \in \mathbb{R}^{n_e \times d}$, and relation embedding $\boldsymbol{R} \in \mathbb{R}^{n_r \times d}$.

Generally, there are two main tensor decomposition techniques that have been introduced to embed n-ary relational data, i.e., CANDECOMP/PARAFAC (CP) decomposition [17] and Tucker decomposition [35]. CP decomposes $\mathcal{X}$ as $\boldsymbol{R} \circ \boldsymbol{E} \circ \cdots \circ \boldsymbol{E}$, and the scoring function measures the plausibility of a n-ary fact



(a) TuckER. (b) DistMult. (c) ComplEx. (d) SimplE.

Figure 1: (a) Each element in TuckER core tensor interprets the correlation between entities and relations of every embedding dimension; (b), (c) and (d) illustrate DistMult, ComplEx and SimplE under representations of TuckER core tensor, respectively. Note that elements that are set to 0 are represented in white while gray elements are unknown.

$s = (r_{i_r}, e_{i_1}, \ldots, e_{i_n})$ with embedding $\boldsymbol{H} = \{\boldsymbol{E}, \boldsymbol{R}\}$ is

$$f(s, \boldsymbol{H}) = \left\langle \boldsymbol{r}_{i_r}, \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n} \right\rangle. \tag{1}$$

Tucker decomposition factorizes $\mathcal{X}$ as $\mathcal{G} \times_1 \boldsymbol{R} \times_2 \boldsymbol{E} \times_3 \cdots \times_{n+1} \boldsymbol{E}$, where $\mathcal{G} \in \mathbb{R}^{d_r \times d_e \times \cdots \times d_e}$. Then, the corresponding scoring function is

$$f(s, \boldsymbol{H}) = \mathcal{G} \times_1 \boldsymbol{r}_{i_r} \times_2 \boldsymbol{e}_{i_1} \times_3 \cdots \times_{n+1} \boldsymbol{e}_{i_n}. \tag{2}$$

Unlike CP, Tucker's core tensor $\mathcal{G}$ encodes the correlation between entity and relation embeddings. Thus, the core tensor enables different entities and relations to share the same set of knowledge of any given N-ary relational data [3].

2.1 Binary Relational Data Learning

In the past decades, embedding approaches have been developed as a promising method to handle binary relational data, such as translational distance models [8, 38], neural network models [4, 10], and tensor decomposition models [3, 19, 24, 34, 44].

As in Section 1, the expressive capability is important for embedding models to achieve outstanding performance. Among kinds of methods, tensor decomposition models demonstrate their superiority in terms of expressive guarantee [19, 37] and empirical performance [22]. More specifically, the literature [19, 24, 34, 44] have been shown to be different variants based on the CP decomposition [22, 50]. And TuckER [3] first introduces Tucker decomposition [21, 35] into binary relational data learning. Generally, the comprehensive core tensor design in TuckER can interpret CP-based tensor decomposition models (e.g., DistMult [44], ComplEx [34], SimplE [19]) as sparse cases of various core tensors as illustrated in Figure 1. But please note that compared with TuckER, the CP-based tensor decomposition models [19, 34, 44] show competitive performance in binary relational data without introducing the dense core tensor. This motivates us to introduce the structured sparsity into high-order tensor decomposition models for N-ary relational data.

2.2 N-ary Relational Data Learning

As presented in Table 1, many models have been proposed to capture n-ary facts, and tensor decomposition models are state-of-the-arts among them. Specifically, the core tensor of n-TuckER in (2) increases exponentially w.r.t the arity n. To address such an over-parameterization problem, GETD [25] simplifies $\mathcal{G}$ with the help of Tensor Ring Decomposition [51], which can approximate the high-order tensor $\mathcal{G}$ by a set of 3-order latent tensors $\{\mathcal{W}_i\}$. GETD first reshapes $\mathcal{G}$ into c-order tensor $\hat{\mathcal{G}} \in R^{d_1 \times \cdots \times d_c}$ with $\prod_{i=1}^{c} d_i = d_e^n d_r$, then decomposes $\hat{\mathcal{G}}$ into c latent 3-order tensors $\{\mathcal{W}_i | \mathcal{W}_i \in R^{n_i \times d_i \times n_{i+1}}\}_{i=1}^{c}$, where $n_1 = \cdots = n_{c+1}$. As a result, (2) is reformulated as

$$\mathcal{X} \approx \mathrm{TR}(\mathcal{W}_1, \cdots, \mathcal{W}_c) \times_1 \boldsymbol{R}^\top \times_2 \boldsymbol{E}^\top \times_3 \cdots \times_{n+1} \boldsymbol{E}^\top, \tag{3}$$

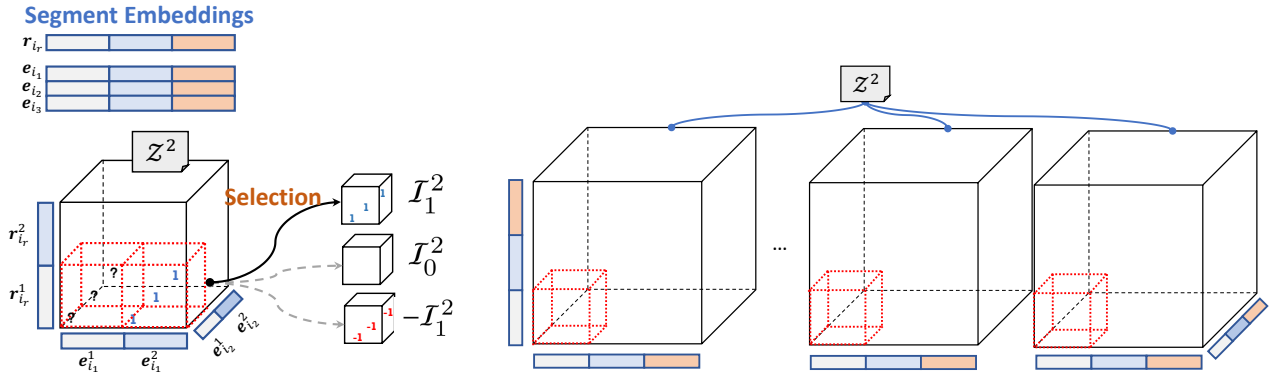
where $\mathrm{TR}(\cdot)$ denotes the Tensor Ring computation [25, 51]. The core tensor in GETD is subsequently reduced to $O(d_{\max}^3)$, where $d_{\max} = \max_i d_i$. However, it still requires cubic complexity, which is hard to train. And note that $\mathcal{X}$ can only represent facts with a specific arity n. Thus, existing tensor decomposition models suffer from the data sparsity issue since they cannot leverage all facts in n-ary relational data.

Table 2: A summary of common notations.

Symbol	Definition
s	The n-ary fact $s = (r_{i_r}, e_{i_1}, \ldots, e_{i_n})$
$\boldsymbol{E}, \boldsymbol{R}$	Embeddings $\boldsymbol{E} \in \mathbb{R}^{n_e \times d}$, $\boldsymbol{R} \in \mathbb{R}^{n_r \times d}$.
$f(s, \boldsymbol{H})$	The scoring function of s with $\boldsymbol{H} = \{\boldsymbol{E}, \boldsymbol{R}\}$
M, N	The number of segments, and maximum arity in given data
$\mathtt{OP}$	Candidate diagonal tensor $\mathtt{OP} = \{-\mathcal{I}_1^n, \mathcal{I}_0^n, \mathcal{I}_1^n\}$
$\mathcal{Z}^n$	The sparse core tensor for facts with arity n
$\boldsymbol{\theta}$	The core tensor weight
$\cdot$	The vector dot product
$\langle \cdot \rangle$	The multi-linear inner product, i.e., $\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \rangle = \sum_{p=1}^{d} [\boldsymbol{a}]_p \cdot [\boldsymbol{b}]_p \cdot [\boldsymbol{c}]_p$
$\circ$	The multi-way outer product, i.e., $(\boldsymbol{R} \circ \boldsymbol{E} \circ \boldsymbol{E})_{ijk} = \langle \boldsymbol{r}_i, \boldsymbol{e}_j, \boldsymbol{e}_k \rangle$
$\times_k$	The k-th mode product of $\mathcal{G} \in \mathbb{R}^{d_1 \times \cdots \times d_n}$ with $\boldsymbol{A} \in \mathbb{R}^{J \times d_k}$, i.e., $(\mathcal{G} \times_k \boldsymbol{A})_{i_1, \ldots, i_{k-1}, j, i_{k+1}, \ldots, i_n} = \sum_{i_k=1}^{d_k} \mathcal{G}_{i_1, \ldots, i_n} \boldsymbol{A}_{j, i_k}$.

3 REFORMULATE TENSOR MODELS

Unfortunately, existing tensor decomposition models for the N-ary relational data still suffer from data-sparsity and over-parameterization (Section 1). First, $\mathcal{X}$ can only represent facts with a specific arity n (Section 2.2), which limits existing models to



(a) Modeling sparse core tensor $\mathcal{Z}_2$. (b) Modeling sparse core tensor $\mathcal{Z}_3$.

Figure 2: Illustration to sparsify core tensor. Set the number of segments $M = 3$. (a) The embedding is segmented into M parts. Then, for the binary fact, we only utilize first 2-th embedding segments for computation and sparsify the core tensor with $\mathcal{Z}^2$, of which component is selected from $\{\mathcal{I}_1^2, \mathcal{I}_0^2, -\mathcal{I}_1^2\}$. (b) For 3-ary fact, we employ all 3 embedding segments for computation. Note that the calculation performed in the red cube is $\mathcal{I}_0^3 \times_1 \boldsymbol{r}_{i_r}^1 \times_2 \boldsymbol{e}_{i_1}^1 \times_3 \boldsymbol{e}_{i_2}^1 \times_4 \boldsymbol{e}_{i_3}^1$.

only learn from facts with the fixed arity. This makes the data-sparsity problem even more serious, as these models cannot fully leverage existing facts. Besides, tensor decomposition models at least require a huge amount of parameters to be the expressive [25]. This makes them difficult to train and easy to overfit since there may not be enough training facts to activate the expressive power. In the sequel, we propose a new tensor model based on sharing embedding (Section 3.1) and sparse core tensors (Section 3.2) to address above issues.

3.1 Share Embedding

As discussed in Section 2.2, tensor decomposition models can only learn from the part of facts, i.e., facts with a specific arity n in N-ary relational data, which causes more severe data sparsity issue. Although they can be forced to jointly learn from facts with mixed arity by share the embedding across various arities [15, 39, 48], such embedding sharing scheme can be too restrictive and lead to poor performance. Thus, to alleviate the data-sparsity issue, we propose to segment embeddings and share different embedding parts across arities for the N-ary relational data learning.

First, given the maximum arity N and number of segments M (usually $M \le N \ll d$), we segment embeddings of relations and entities into M splits, i.e., $\boldsymbol{e}_i = [\boldsymbol{e}_i^1; \dots; \boldsymbol{e}_i^M]$ where $\boldsymbol{e}_i^j \in \mathbb{R}^{d/M}$, and same for relation $\boldsymbol{r}_{i_r}$. Then, given the arity n and $m = \min\{n, M\}$, we utilize first m-th segments of embeddings to compute the score. For example, given an entity vector $\boldsymbol{e}_i = [\boldsymbol{e}_i^1; \dots; \boldsymbol{e}_i^3]$, we use $[\boldsymbol{e}_i^1; \boldsymbol{e}_i^2]$ if it involves in a binary fact and use $[\boldsymbol{e}_i^1; \boldsymbol{e}_i^2; \boldsymbol{e}_i^3]$ for facts with arity 3 or even higher. Then, to handle n-ary facts, we build a core tensor $\mathcal{Z}^n$ for every arity n, where $\mathcal{Z}^n$ is a $(n+1)$-order tensor with size md/M (e.g., $\mathcal{Z}^2 \in \mathbb{R}^{2d/M \times 2d/M \times 2d/M}$). Overall, the proposed approach can handle the N-ary relational data with mixed arity by learning multiple core tensors $\{\mathcal{Z}^n\}_{n=2}^N$. Such embedding sharing with segments can make embeddings learn from the low-order information in the high-order fact training, but also retain a part of the high-order specific information.

Unfortunately, each $\mathcal{Z}^n$ requires $O((md/M)^{n+1})$ and may still lead to over-parameterization. Next, we introduce sparse core tensors that require much less complexity but maintains expressiveness.

3.2 Sparsify Core Tensor

Existing tensor decomposition models require a large number of parameters to maintain the expressiveness for the N-ary relational data, which makes the model inefficient and difficult to train. Thus, the question comes that *is it essential to learn a dense core tensor with so many trainable parameters for strong expressiveness*? To answer this question, we first review the domain-specific knowledge on binary relational data.

3.2.1 Motivation from Binary Relational Data. TuckER introduces the dense core tensor $\mathcal{G} \in \mathbb{R}^{d_r \times d_e \times d_e}$ to achieve outstanding performance in binary relational data. In (2), each entry $\mathcal{G}_{k_r,k_1,k_2}$ in $\mathcal{G}$ actually interprets the correlation among embeddings at the dimension level, i.e., the k_r-th dimension of $\boldsymbol{r}$, k_1-th dimension of $\boldsymbol{e}_1$, and k_2-th dimension of $\boldsymbol{e}_2$. However, such a redundant core tensor is hard to train and easy to overfit.

As mentioned in Section 2.1, other simple tensor-based models, such as ComplEx [34], and SimplE [19], can be regarded to have sparse core tensors with special patterns (see Figure 1). But these simple models are expressive and achieve relatively good performance without introducing dense core tensor. Consequently, it may be unnecessary to learn a smaller complex core tensor with an enormous amount of parameters in N-ary relational data. This motivates us to sparsify the core tensor $\{\mathcal{Z}^n\}_{n=2}^N$ in the n-ary case by only interpreting the correlation among embedding segments.

3.2.2 Structured Sparsity in Core Tensors. We first divide the core tensor $\mathcal{Z}^n$ into $K = m^{n+1}$ tensors, denoted as $\mathcal{Z}^n = \{\mathcal{Z}_k^n\}_{k=1}^K$, where $\mathcal{Z}_k^n$ is a $(n+1)$-order tensor with size d/M. After delving deep into tensor models on binary relational data (Figure 1), we observe that simple values (i.e., -1, 0 and 1) on the diagonal form of the core tensor are expressive for capturing interactions. We first

define such simple interaction in the high-order scenario. A tensor $\mathcal{I}$ is *diagonal* when $\mathcal{I}_{i,j,\ldots,k} \neq 0$ holds if and only if $i = j = \cdots = k$. We use $\mathcal{I}_v^n$ to denote a $(n+1)$-order tensor with size d/M, which is diagonal with v on the super-diagonal and zeros elsewhere. Then, we propose to select the appropriate diagonal tensor from $\{-\mathcal{I}_1^n, \mathcal{I}_0^n, \mathcal{I}_1^n\}$ to replace $\mathcal{Z}_k^n \in \mathcal{Z}^n$ as Figure 2 (a). Then, the diagonal tensor $\mathcal{I}_v^n$ encodes the correlation among embedding segments $(\boldsymbol{r}_{i_r}^{j_r}, \boldsymbol{e}_{i_1}^{j_1}, \ldots, \boldsymbol{e}_{i_n}^{j_n})$, where $-\mathcal{I}_1^n$ represents the negative correlation, $\mathcal{I}_0^n$ is no correlation, and $\mathcal{I}_1^n$ denotes the positive correlation. Note that any positive or negative value v can be used for $\mathcal{I}_v^n$ here. We utilize 1 and 0 for simplicity. Formally, we formulate the definition of sparse core tensor as:

Definition 1 (Sparse Core Tensor). *Given the embedding dimension d, the maximum arity N and a specific arity n, let $\mathcal{I}_v^n$ denote the $(n+1)$-order diagonal tensor with size d/M, and $\texttt{OP} = \{-\mathcal{I}_1^n, \mathcal{I}_0^n, \mathcal{I}_1^n\}$ denote the operation set of candidate diagonal tensors. Then, we propose to select every $\mathcal{Z}_k^n \in \mathcal{Z}^n$ from $\texttt{OP}$. Overall, the sparse core tensor is denoted to $\mathcal{Z}^n = \{\mathcal{Z}_k^n\}_{k=1}^K$, which interprets facts with the arity n.*

Accordingly, given any fact s with arity n_s, the scoring function based on $\mathcal{Z}^{n_s}$ is formulated as:

$$f_z(s, \boldsymbol{H}; \mathcal{Z}^{n_s}) = \sum_{j_r, j_1, \ldots, j_n} \mathcal{Z}_k^{n_s} \times_1 \boldsymbol{r}_{i_r}^{j_r} \times_2 \boldsymbol{e}_{i_1}^{j_1} \times_3 \cdots \times_{n_s+1} \boldsymbol{e}_{i_{n_s}}^{j_{n_s}}, \quad (4)$$

where any $j \in \{1, \ldots, m\}$ and $k \in \{1, \ldots, m^{n+1}\}$ corresponds to $(j_r, j_1, \ldots, j_n)$. Compared with GETD's core tensor $O(cd_{\max}^3)$, one sparse core tensor $\mathcal{Z}^n$ has a complexity of $O\left(m^{n+1}\right)$. But note that $m, n \ll d_e$ or d_r, and the arity n over 4 are really rare in the common knowledge bases [25]. Thus, we generally set the number of segments $M = 4$ for the N-ary relational data in practical, which leads to a constant complexity such as $4^5 = 1,024$. It is far smaller than the complexity of core tensor in GETD [25] in the real case (e.g., $4 \cdot 50^3 = 500,000$). And we theoretically demonstrate the expressiveness of S2S sparse core tensor design as in Theorem 1. The proof is presented in Appendix A.

Theorem 1. *Given any N-ary relational data S on the sets of entity E and relation R, there exists a set of sparse core tensors $\{\mathcal{Z}^n\}_{n=2}^N$ with embeddings $\boldsymbol{E}$ and $\boldsymbol{R}$ that is able to accurately represent that ground truth.*

In summary, we have enabled tensor decomposition models to learn from mixed arity and maintained the expressiveness of core tensors with less model complexity. However, it is still a non-trivial problem to design proper sparse core tensors $\{\mathcal{Z}^n\}_{n=2}^N$ due to a large number of candidates. Recall that $\mathcal{Z}_k^n \in \mathcal{Z}^n$ can be arbitrarily and independently chosen from $\texttt{OP}$ in Definition 1. Assume that $M = 4$, there are totally 3^{81} candidates for $\mathcal{Z}^3$. In the next, we will introduce how to find proper sparse core tensors by leveraging the Neural Architecture Search (NAS) method.

4 SEARCH ALGORITHM

In general, the scoring function design should be a data-specific problem. Since the N-ary relational data also owns specific prior-knowledge, it is crucial to search for a set of proper sparse core tensors that can lead to outstanding performance on various N-ary relational data.

4.1 Problem Formulation

Continuous formulation [23, 46] and stochastic formulation [1, 40] are two popular formulations in NAS literature, they both model choices from a given operation set as a differentiable optimization problem. The difference is that continuous relaxation directly couples all candidate operations together, while stochastic relaxation samples each candidate based on a learned distribution.

Considering that $-\mathcal{I}_1^n$ and $\mathcal{I}_1^n$ should not be coupled together since they are exactly the opposite, we follow stochastic relaxation and sample $\mathcal{Z}_k^n$ independently and stochastically from $\texttt{OP}$. For a $\mathcal{Z}^n = \{\mathcal{Z}_k^n\}_{k=1}^K$, let θ_{pk}^n denote the probability of $o_p \in \texttt{OP}$ to be sampled for $\mathcal{Z}_k^n$, where $\sum_p \theta_{pk}^n = 1$. Then, we utilize $\boldsymbol{\theta}^n = [\theta_{pk}^n]_{3\times K}$ maintain the probability weight for $\{\mathcal{Z}_k^n\}_{k=1}^K$, thus $\boldsymbol{\theta} = \{\boldsymbol{\theta}^n\}_{n=2}^N$ for all sparse core tensor $\{\mathcal{Z}^n\}_{n=2}^N$. Moreover, we utilize $\mathcal{Z} = \{\mathcal{Z}^n\}_{n=2}^N$ to represent the sampled sparse core tensor from the categorical distribution $p_{\boldsymbol{\theta}}(\mathcal{Z})$. Follow [11, 18, 45], we formulate the searching to sparsify core tensor problem as a bi-level optimization problem in Definition 2.

Definition 2 (Search Problem). *Given the training and validation facts S_{tra} and S_{val}, the sparse core tensor search problem is defined as follows:*

$$\bar{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathcal{Z})}\left[M(\bar{\boldsymbol{H}}, \mathcal{Z}; S_{val})\right], \quad (5)$$

$$s.t.\ \bar{\boldsymbol{H}} = \arg\min_{\boldsymbol{H}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathcal{Z})}\left[L(\boldsymbol{H}, \mathcal{Z}; S_{tra})\right]. \quad (6)$$

Note that L (resp. M) measures the loss (resp. mean reciprocal ranking [8, 38]) on the training (resp. validation) data. The bi-level formulation in Definition 2 is hard to optimize since both the embedding $\boldsymbol{H}$ and the sparse core tensor weight $\boldsymbol{\theta}$ are hierarchically coupled. In the sequel, we propose an efficient algorithm for optimization, which is motivated by recent NAS algorithms [1, 40].

4.2 Searching to Sparsify Core Tensor

Finally, we summarize the algorithm of searching to sparsify core tensor in Algorithm 1, where embedding $\boldsymbol{H}$ and core tensor weight $\boldsymbol{\theta}$ are alternatively updated. Alternating steepest ascent [1, 23, 40, 46] is a way to avoid computationally heavy optimization (5) and (6). For any sampled sparse core tensor $\mathcal{Z}$, we first optimize the embedding $\boldsymbol{H}$ on $\mathcal{Z}$ with a mini-batch data in steps 3-4. Then, we evaluate the performance of sampled $\mathcal{Z}$ on the updated $\boldsymbol{H}$, which leads to a fast evaluation mechanism. Thus, we are able to update the core tensor weight $\boldsymbol{\theta}$ every iteration in step 5-6. After searching, we derive the most likely sparse core tensor $\{\bar{\mathcal{Z}}^n\}_{n=2}^N$ with the fine-tuned $\bar{\boldsymbol{\theta}}$ in step 8. Finally, we learn the embedding $\boldsymbol{H}$ by training $\{\bar{\mathcal{Z}}^n\}_{n=2}^N$ from scratch in step 9.

Given the distribution $p_{\boldsymbol{\theta}}(\mathcal{Z})$, we propose to solve (6) by minimizing the expected loss L on the training data S_{tra}. Then, stochastic gradient descent can be performed to optimize the embedding $\boldsymbol{H}$. Based on Monte-Carlo (MC) sampling [16], we sample λ core tensor sets to approximate the gradient $\nabla_{\boldsymbol{H}}$ as

$$\nabla_{\boldsymbol{H}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathcal{Z})}[L] \approx \frac{1}{\lambda} \sum_{i=1}^{\lambda} \nabla_{\boldsymbol{H}} L(\boldsymbol{H}, \mathcal{Z}^{(i)}; S_{\text{tra}}), \quad (7)$$

where $\mathcal{Z}^{(i)}$ is a core tensor set that independent and identically distributed (i.i.d.) sampled from $p_{\boldsymbol{\theta}}(\mathcal{Z})$, and $L(\boldsymbol{H}, \mathcal{Z}^{(i)}; S_{\text{tra}})$ is

Algorithm 1 S2S: Searching to Sparsify Tensor Decomposition for N-ary relational data

1: Initialize the embedding $\boldsymbol{H}$, probability distribution $p_{\boldsymbol{\theta}}(\mathcal{Z})$.
2: **while** not converged **do**
3: Randomly sample a mini-batch B_{tra} from S_{tra} and sparse core tensor set $\mathcal{Z}$ from $p_{\boldsymbol{\theta}}(\mathcal{Z})$;
4: Update embeddings $\boldsymbol{H}$ with $\nabla_{\boldsymbol{H}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathcal{Z})}[L]$ in (7);
5: Randomly sample a mini-batch B_{val} from S_{val};
6: Update the weight $\boldsymbol{\theta}$ with $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathcal{Z})}[M]$ in (9);
7: **end while**
8: Derive final $\{\bar{\mathcal{Z}}^n\}_{n=2}^N$ from the fine tuned $\bar{\boldsymbol{\theta}}$, such as $\bar{\mathcal{Z}}_k^n = o_p$ where $p = \arg\max_p \theta_{pk}^n$;
9: Achieve the final embedding $\bar{\boldsymbol{H}}$ by training embeddings with $\{\bar{\mathcal{Z}}^n\}_{n=2}^N$ from scratch to convergence.

computed as:

$$L(\boldsymbol{H}, \mathcal{Z}^{(i)}; S_{\text{tra}}) = \sum\nolimits_{s \in S_{\text{tra}}} \ell\big(s, f_z(\boldsymbol{H}; \mathcal{Z}^{n_s})\big), \tag{8}$$

where $\ell(\cdot)$ is the extension of multi-class log-loss [22] in the n-ary case [25] for a single fact s. Similarly, the gradient w.r.t $\boldsymbol{\theta}$ can be approximated by MC sampling as:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathcal{Z})}[M] \approx \frac{1}{\lambda} \sum\nolimits_{i=1}^{\lambda} \nabla_{\boldsymbol{\theta}} M(\boldsymbol{H}, \mathcal{Z}^{(i)}; S_{\text{val}}). \tag{9}$$

Then, we propose to leverage ASNG [1], which is the state-of-the-art stochastic optimization technique in NAS for optimizing $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} M(\boldsymbol{H}, \mathcal{Z}^{(i)}; S_{\text{val}}) = \sum_{s \in S_{\text{val}}} m\big(s, f_z\left(\boldsymbol{H}; \mathcal{Z}^{n_s}\right)\big)\big(T\left(\mathcal{Z}^{n_s}\right) - \boldsymbol{\theta}^{n_s}\big),$$

where $m(\cdot)$ measures the MRR performance on a single fact s and $T(\cdot)$ denotes the sufficient statistic [1].

4.3 Comparison with AutoSF

The closest work in the literature of the N-ary relational data is AutoSF [50], which proposes a NAS approach to search data-specific and bilinear scoring functions. The proposed S2S differs from AutoSF from three perspectives: task scenario, search space, and search algorithm. AutoSF concerns the binary relational data based on the unified representation of embedding approaches. We generalize the task scenario from the binary to N-ary relational data. Correspondingly, we propose a novel search space where we can search for sparse core tensor in N-ary relational data. And the search space of AutoSF is a special case of our proposed sparse core tensor. Third, AutoSF develops an inefficient search algorithm, that requires training hundreds of candidates to convergence. However, the N-ary relational data requires a much larger search space, which results in the efficiency issue become even more severe. In this paper, we enable an efficient search algorithm ASNG [1] in our scenario, where the desired sparse core tensor can be searched by only training once.

5 EXPERIMENTS

All codes are implemented with PyTorch and run on a single Nvidia RTX2080Ti GPU.

5.1 Experimental Setup

5.1.1 Data Sets. To demonstrate the performance of the proposed method, we conduct experiments on N-ary relational data with both various fixed arity (i.e., $n = 2, 3, 4$) and mixed arity. The statistics of data sets are summarized in Table 3.

- *N-ary relational data.* We follow [15, 25, 30, 39, 48] to compare various models on WikiPeople [15] and JF17K [39]. WikiPeople mainly concerns the entities of typing humans, which is extracted from Wikidata. And JF17K is developed from Freebase [6]. Then, for 3-ary and 4-ary relational data, we follow GETD [25] to filter out all 3-ary and 4-ary facts from WikiPeople and JF17K respectively, named as JF17K-3, JF17K-4, WikiPeople-3, and WikiPeople-4.
- *Binary relational data (aka. knowledge graph).* We follow [3, 8, 19, 34, 50] to conduct experiments on four public benchmark data sets: WN18 [8], WN18RR [10], FB15k [8], FB15k237 [33]. WN18RR and FB15k237 are variants of WN18 and FB15k respectively by removing duplicate and inverse relations.

Table 3: Summary of benchmark N-ary relational data sets.

	Data set	#ent	#rel	#Tra	#Val	#Tst
fixed n-ary	WikiPeople-3	12,270	66	20,656	2,582	2,582
	WikiPeople-4	9,528	50	12,150	1,519	1,519
	JF17K-3	11,541	104	27,635	3,454	3,455
	JF17K-4	6,536	23	7,607	951	951
mixed n-ary	WikiPeople	47,765	707	305,725	38,223	38,281
	JF17K	28,645	322	76,379	-	24,568
binary	WN18	40,943	18	141,442	5,000	5,000
	WN18RR	40,943	11	86,835	3,034	3,134
	FB15k	14,951	1,345	484,142	50,000	59,071
	FB15k237	14,541	237	272,115	17,535	20,466

5.1.2 Evaluation Metrics. We test the performance of our proposed method on the link prediction task [49, 50], which is utilized to complete the N-ary relational data. Given a n-ary fact $s = (r_{i_r}, e_{i_1}, \ldots, e_{i_n})$, the embedding model assumes one entity in this fact is missing, then it ranks all candidate entities based their scores. We adopt the standard metrics [8, 38]:

- Mean Reciprocal Ranking (MRR): $1/|S| \sum_{i=1}^{|S|} 1/\text{rank}_i$, where rank_i is the ranking result, and
- Hits@T: $1/|S| \sum_{i=1}^{|S|} \mathbb{I}(\text{rank}_i \leq T)$, where $\mathbb{I}(\cdot)$ is the indicator function and $T \in \{1, 3, 10\}$.

Note that the higher MRR and Hits@T indicate the better quality of embeddings. And all metrics are reported in a "filter" setting [8], where the ranking computation is not include the corrupted facts that exist in train, valid and test data sets.

5.1.3 Hyper-parameter Settings. The proposed method mainly contains two steps, searching for sparse core tensor, and training the searched core tensor to convergence. In the search strategy, we utilize the default hyper-parameters implemented in ASNG [1] for optimizing the core tensor weight. Then, we train the embeddings on the searched hyper-parameter set, which is achieved by tuning

Table 4: The link prediction results on the WikiPeople-3/4.

model type	model	WikiPeople-3				WikiPeople-4			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
translation	RAE [48]	0.239	0.168	0.252	0.379	0.150	0.080	0.149	0.273
neural network	NaLP [25]	0.301	0.226	0.327	0.445	0.342	0.237	0.400	0.540
	HINGE [30]	0.338	0.255	0.360	0.508	0.352	0.241	0.419	0.557
	NeuInfer [14]	0.355	0.262	0.388	0.521	0.361	0.255	0.424	0.566
tensor decomposition	n-CP [25]	0.330	0.250	0.356	0.496	0.265	0.169	0.315	0.445
	n-TuckER [25]	0.365	0.274	0.400	0.548	0.362	0.246	0.432	0.570
	GETD [25]	0.373	0.284	0.401	0.558	0.386	0.265	0.462	0.596
	S2S	**0.386**	**0.299**	**0.421**	**0.559**	**0.391**	**0.270**	**0.470**	**0.600**

Table 5: The link prediction results on the JF17K-3/4.

model type	model	JF17K-3				JF17K-4			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
translation	RAE [48]	0.505	0.430	0.532	0.644	0.707	0.636	0.751	0.835
neural network	NaLP [25]	0.515	0.431	0.552	0.679	0.719	0.673	0.742	0.805
	HINGE [30]	0.587	0.509	0.621	0.738	0.745	0.700	0.775	0.842
	NeuInfer [14]	0.622	0.533	0.658	0.770	0.765	0.722	0.808	0.871
tensor decomposition	n-CP [25]	0.700	0.635	0.736	0.827	0.787	0.733	0.821	0.890
	n-TuckER [25]	0.727	0.664	0.761	0.852	0.804	0.748	0.841	0.902
	GETD [25]	0.732	0.669	0.764	0.856	0.810	0.755	0.844	0.913
	S2S	**0.740**	**0.676**	**0.770**	**0.860**	**0.822**	**0.761**	**0.853**	**0.924**

CP/n-CP [22] with the help of HyperOpt [5]. This hyper-parameter set includes the learning rate, decay rate, batch size, and embedding dimension. Besides, we optimize the embedding with Adam algorithm [20]. To determine the sparse core tensor for evaluation, we run S2S five times and report average results.

5.2 N-ary Relational Data with Fixed Arity

We first compare our S2S with other models in N-ary relational data with fixed arity, i.e., WikiPeople-3, WikiPeople-4, JF17K-3, and JF17K-4. We adopt the n-ary tensor decomposition models, n-CP [22], n-TuckER [3], and GTED [25], as baselines. As for the translational model, we only include the advanced RAE [48] since it is an upgraded version of m-TransH [39]. And we also compare the neural network models NaLP [15], HINGE [30], and NeuInfer [14].

5.2.1 Benchmark Comparison. We demonstrate the performance on N-ary relational data with fixed arity in Table 4-5. We can observe that tensor decomposition models (n-CP, n-TuckER, GETD, and S2S) generally have better performance than other models in Table 4-5. That is mainly because tensor decomposition models have strong expressiveness. Then, although n-CP requires the lowest complexity $O(n_e d_e + n_r d_r)$ among tensor decomposition models, it does not achieve the high performance as other tensor decomposition models (e.g., n-TuckER, GETD, and S2S). That is because n-CP does not introduce a core tensor like tensor decomposition models, which can enable the embedding to share the domain knowledge. Furthermore, we can observe that GETD performs better than n-TuckER since GETD partially addresses the over-parameterized problem in n-TuckER. Overall, our proposed S2S consistently achieves state-of-the-art performance on all benchmark data sets by the data-specific core tensor design.

5.2.2 Training Efficiency. Moreover, we show the learning curve of several tensor decomposition models to compare the efficiency in Figure 3. n-CP converges fastest due to the lowest model complexity. The convergence rate of n-TuckER is the slowest since it requires the most complexity. GETD converges much faster than n-TuckER because it reduces the complexity of the core tensor. And the convergence of S2S is only slower than that of n-CP and faster than GETD and n-TuckER due to our sparse core tensor design.

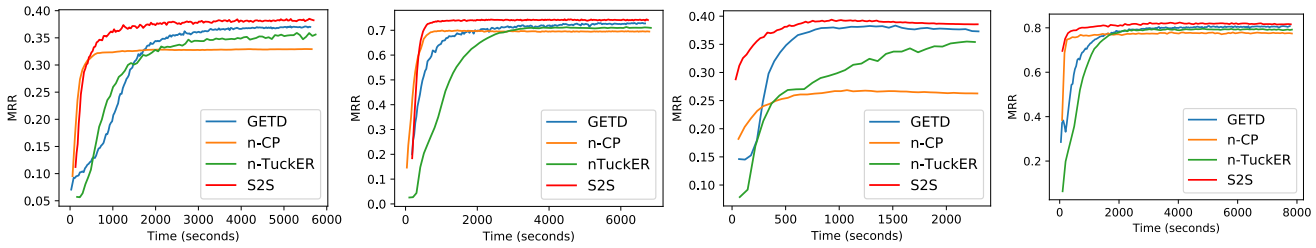
5.3 N-ary Relational Data with Mixed Arity

To demonstrate the importance of mixed arity and superiority of our S2S, we compare it with other advanced models on the N-ary relational data, i.e., Wiki-People [15] and JF17K [39]. We include the advanced translational model RAE [48], the neural networks models NaLP [15], HINGE [30] and NeuInfer [14], and a hybrid model HypE [13].

5.3.1 Benchmark Comparison. We show the performance on N-ary relational data with mixed arity in Table 6. Because of lack of expressive ability, the translational model RAE does not achieve good performance. The neural network models [14, 15, 30] generally outperform the translational model RAE by leveraging complex networks. On the contrary, S2S leads to state-of-the-art performance because of the expressive guarantee.

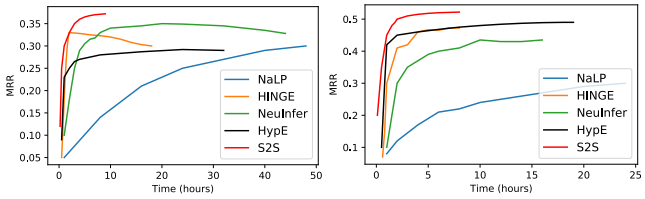
Table 6: The link prediction results on the multi-relational data set with mixed arity.

model	WikiPeople				JF17K			
	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
RAE [48]	0.172	0.102	0.182	0.320	0.310	0.219	0.334	0.504
NaLP [15]	0.338	0.272	0.364	0.466	0.366	0.290	0.391	0.516
HINGE [30]	0.333	0.259	0.361	0.477	0.473	0.397	0.490	0.618
NeuInfer [14]	0.350	**0.282**	0.381	0.467	0.517	0.436	0.553	0.675
HypE [13]	0.292	0.162	0.375	0.502	0.494	0.408	0.538	0.656
S2S	**0.372**	0.277	**0.439**	**0.533**	**0.528**	**0.457**	**0.570**	**0.690**



(a) WikiPeople-3. (b) JF17K-3. (c) WikiPeople-4. (d) JF17K-4.

Figure 3: Testing MRR v.s. clock time (seconds) with fixed arity.



(a) WikiPeople. (b) JF17K.

Figure 4: Testing MRR v.s. clock time (hours) with mixed arity.

5.3.2 Training Efficiency. In Figure 4, it is obvious that the neural network models, i.e., NaLP [15] and NeuInfer [14], require quite a long time to convergence. That is because these two models utilize complex neural networks for training. On the contrary, another neural network model HINGE [30] proposes a simple way to train the embeddings, which converges much fast. Among all models, S2S achieves the fastest convergence since it requires less complexity with the sparse core tensor.

5.4 Binary Relational Data

To further demonstrate the performance of the proposed method, we also compare S2S with classical embedding approaches on binary relational data, i.e., WN18 [8], WN18RR [10], FB15k [8], FB15k237 [33]. We include the most advanced translational model RotatE [32] due to its outstanding performance among translational models. We also compare two popular neural network models, ConvE [10] and HypER [4]. As for tensor-based models, we include DistMult [44], ComplEx [34], SimplE [19], HolEX [42], QuatE [49], and TuckER [3]. Moreover, we include the recent scoring function search method, AutoSF [50], which only concerns the binary relational data as mentioned in Section 4.3.

5.4.1 Benchmark Comparison. The ranking performance is in Table 7. It is clear that classical models cannot consistently achieve good performance on various data sets, since these models are not data-specific. AutoSF can search for a suitable scoring function for each data set and consistently achieve outstanding performance. The proposed S2S is also data-specific, which aims to search proper sparse core tensor for any given data. Overall, S2S consistently achieves state-of-the-art performance in all data sets.

5.5 Search Efficiency

To investigate the search efficiency of the proposed method, we summarize the running time of S2S and other models on 4 binary data sets in Table 8. We compare S2S with AutoSF in terms of the score function search time, and stand-alone training time of searched score function. Note that S2S sets the embedding dimension to 512 in the search procedure for all data sets. As for stand-alone training, we set embedding dimension for all models at 1024. We utilize the simplest tensor decomposition model DistMult [44] as the benchmark. In stand-alone training, S2S significantly reduces the training time compared with TuckER since it sparsifies the core tensor of TuckER. And the training time of the scoring function searched by S2S is a little longer than DistMult. That is because S2S searches a slightly more complex core tensor than DistMult's as illustrated in Figure 1 (b) and Figure 2 (a). Compared with another search approach AutoSF, S2S significantly reduces the search cost. AutoSF adopts the stand-alone evaluation mechanism, which requires training the hundreds of candidate scoring functions to convergence. But the proposed S2S enables an efficient search algorithm ASNG [1], where the proper scoring

Table 7: Comparison of the proposed S2S and state-of-the-art scoring functions on the link prediction task.

model	model	WN18			WN18RR			FB15k			FB15k237		
		MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10
translation	RotatE [32]	0.949	0.944	0.959	0.476	0.428	0.571	0.797	0.746	0.884	0.338	0.241	0.533
neural network	ConvE [10]	0.943	0.935	0.956	0.460	0.390	0.480	0.754	0.670	0.873	0.316	0.239	0.491
	HypER [4]	0.951	0.947	0.958	0.465	0.436	0.522	0.790	0.734	0.885	0.341	0.252	0.520
tensor decomposition	HolEX [42]	0.938	0.930	0.949	-	-	-	0.800	0.750	0.886	-	-	-
	QuatE [49]	0.950	0.945	0.959	0.488	0.438	**0.582**	0.782	0.711	0.900	0.348	0.248	0.550
	DistMult [44]	0.821	0.717	0.952	0.443	0.404	0.507	0.817	0.777	0.895	0.349	0.257	0.537
	ComplEx [34]	0.951	0.945	0.957	0.471	0.430	0.551	0.831	0.796	0.905	0.347	0.254	0.541
	SimplE [19]	0.950	0.945	0.959	0.468	0.429	0.552	0.830	0.798	0.903	0.350	0.260	0.544
	TuckER [3]	0.953	**0.949**	0.958	0.470	0.443	0.526	0.795	0.741	0.892	0.358	0.266	0.544
	GETD [25]	0.948	0.944	0.954	-	-	-	0.824	0.787	0.888	-	-	-
NAS	AutoSF [50]	0.952	0.947	0.961	0.490	0.451	0.567	**0.853**	**0.821**	**0.910**	0.360	0.267	0.552
	S2S	**0.955**	**0.949**	**0.963**	**0.498**	**0.455**	0.577	0.850	0.820	**0.910**	**0.368**	**0.270**	**0.559**

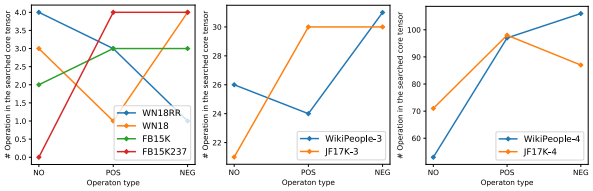
Table 8: Running time (in hours) analysis of several models.

data set	DistMult	S2S		AutoSF		TuckER
		Search	Training	Search	Training	
WN18	1.9±0.1	2.0±0.2	2.4±0.1	65.7±3.0	2.4±0.1	25.4±1.5
WN18RR	0.4±0.1	1.3±0.1	0.6±0.1	38.6±1.9	0.6±0.1	18.7±1.1
FB15k	8.4±0.2	4.8±0.2	11.1±0.4	127.1±5.2	10.9±0.3	38.7±2.9
FB15k237	2.6±0.1	3.3±0.3	4.8±0.2	61.1±2.8	4.6±0.2	21.3±1.8

function can be searched by only training once (i.e., one-shot manner). Furthermore, S2S searches only take a bit more time than DistMult since it needs to update the architecture parameter in search. In summary, the proposed method is very efficient in terms of search and stand-alone training.

5.6 Case Study

Here, we demonstrate the number of operations of searched core tensor in the below Figure 5. It indicates that S2S is data-specific, which can search various sparse core tensor $\mathcal{Z}^n$ for different data sets.



(a) binary. (b) 3-ary. (c) 4-ary.

Figure 5: The number of operations searched by S2S in several data sets. Note that NO, POS, NEG represents $\mathcal{I}_0^n$, $\mathcal{I}_1^n$, and $-\mathcal{I}_1^n$ respectively.

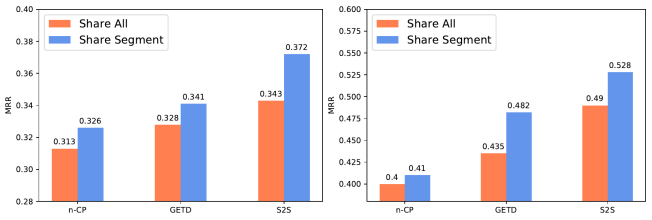
5.7 Ablation Study

5.7.1 The Influence of the Joint Learning. As discussed in Section 1, the tensor decomposition models only learn embedding from part of N-ary relational data, which causes the data sparsity issue to become more severe. To verify this claim, we include another S2S (mixed) learned from N-ary relational data with mixed arity to compare the S2S (fixed) reported in Table 4 and Table 5, which is learned from fixed arity. It is obvious that S2S (mixed) achieves better performance, which demonstrates that only leveraging part of N-ary relational data indeed suffers from the data-sparsity issue. This verifies that we need to propose a tensor decomposition model for the N-ary relational data learning. We further discuss the effectiveness of proposed embedding sharing in Section 5.7.2.

Table 9: The performance comparison of S2S between fixed learning and joint learning.

data set	S2S (fixed)		S2S (mixed)	
	MRR	Hits@10	MRR	Hits@10
WikiPeople-3	0.386	0.559	0.408	0.577
WikiPeople-4	0.391	0.600	0.418	0.617
JF17K-3	0.740	0.860	0.752	0.870
JF17K-4	0.822	0.924	0.831	0.934

5.7.2 The Influence of the Embedding Sharing Way. In Section 5.7.1, we show that the sparsity issue exists when models only leverage part of N-ary relational data. As discussed in Section 3.1, it is hard for tensor decomposition models to handle the N-ary relational data with mixed arity. Directly sharing all embeddings across arities is too restrictive and lead to poor performance [15, 39, 48]. Therefore, we propose to share embeddings based on segments. To verify claims and investigate the influence of embedding sharing ways, we demonstrate the performance of several tensor decomposition models on WikiPeople and JF17K as in Figure 6. Appendix B.1 introduces the details of implementing embedding sharing into tensor decomposition models.

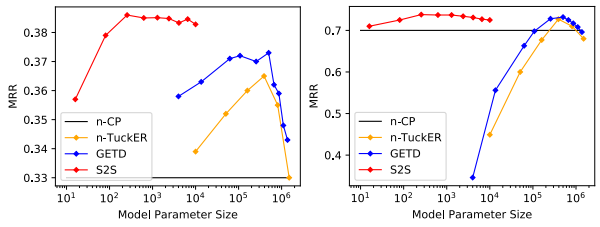


(a) WikiPeople. (b) JF17K.

Figure 6: The influence of different embedding sharing in tensor models.

First, we can observe that all tensor decomposition models achieve better performance with sharing embedding segments. That is because embedding sharing not only makes the embedding learn from the low-arity fact in the high-order training but also maintain a part of high-order knowledge. Second, it is clear that GTED and S2S achieve better performance than n-CP in N-ary relational data. Unlike n-CP, GTED and S2S need to learn a core tensor for facts with every arity n. The core tensor can encode the arity-specific knowledge, that further enhance the performance in joint learning.

5.7.3 The Influence of the Model Complexity. Previously, we discuss the negative effect of the over-parameterized issue in existing tensor decomposition models. As mentioned in Section 1, cubic or even larger model complexity is easy to make the model difficult to train. Therefore, we here investigate the influence of model parameter size in Figure 7. Note that we do not include the embedding as the model parameter since every model at least require $O(n_e d_e + n_r d_r)$ for embedding. Thus we plot n-CP [22] as a horizontal line since it has no extra parameter. We can observe that S2S can achieve outstanding performance by requiring a small number of parameters. And its performance does not vary greatly with the increase of model parameters. On the contrary, GETD and n-TuckER require much larger parameter size to achieve the high performance. And their model parameter setting will lead to significant differences in performance. This may bring a difficulty to the training in practical, such as the careful selection of the size of model parameters.



(a) WikiPeople-3. (b) JF17K-3.

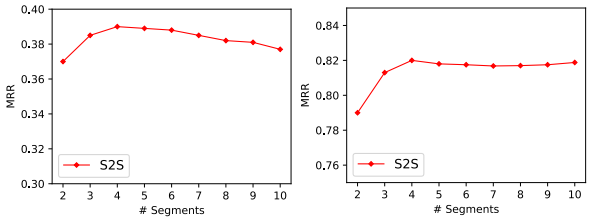
Figure 7: The influence of model parameters.

5.7.4 The Influence of the Structured Sparse Core Tensor. We demonstrate the over-parameterization issue in Section 5.7.3. And we can observe that S2S achieves outstanding performance in Table 3-6. To investigate the effectiveness of the proposed structured sparse core tensor, we compare S2S with S2S(L0-reg), which encourages the sparse core tensor by ℓ_0 constraint. S2S(L0-reg) has the same number of non-zero elements as S2S, its sparse pattern is not structured and non-zero elements can be arbitrarily distributed across the core tensor. Results are in Table 10. We can observe that the performance of S2S(L0-reg) is much worse than the performance of S2S as reported in Table 4-5. That is because the unstructured sparse core tensor cannot capture the correlation between embeddings as well as the structured one. The implementation details have been introduced in Appendix B.2.

Table 10: The link prediction performance of S2S(L0-reg).

data set	S2S(L0-reg)		S2S	
	MRR	Hits@10	MRR	Hits@10
WikiPeople-3	0.289	0.426	0.386	0.559
WikiPeople-4	0.288	0.457	0.391	0.600
JF17K-3	0.665	0.774	0.740	0.860
JF17K-4	0.755	0.822	0.822	0.924

5.7.5 The Impact of the Number of Segments. We here investigate the effect of the different number of segments (i.e., M) on the N-ary relational data learning with fixed arity in Figure 8. We can observe that S2S has good performance when the number of segments is set to 4. And the effect is not sensitive to the parameter setting.



(a) WikiPeople-4. (b) JF17K-4.

Figure 8: The effects of the number of segments in S2S.

5.7.6 Single v.s. Bi-level Formulation. We follow NAS to formulate Definition 2 into a bi-level optimization problem. To investigate the impact of optimization level, we add a variant of S2S named S2S(sig), which optimizes (5) based on training data S_{tra}. As shown in Table 11, the effect of S2S(sig) is generally lower than S2S. That is because using validation data to optimize $\boldsymbol{\theta}$ will encourage the model to find core tensors that generalize well, rather than fitting the training data well.

Table 11: The link prediction performance of S2S(sig).

data set	S2S(sig)		S2S	
	MRR	Hits@10	MRR	Hits@10
WikiPeople-3	0.377	0.545	0.386	0.559
WikiPeople-4	0.380	0.592	0.391	0.600
JF17K-3	0.727	0.839	0.740	0.860
JF17K-4	0.800	0.908	0.822	0.924

6 CONCLUSION

In this paper, we propose a new tensor decomposition model, i.e., S2S, to learn embedding from the N-ary relational data. First, to alleviate the data-sparsity issue, we propose to segment embeddings into multiple parts and share them across arities by different segments. Then, the proposed tensor decomposition model is able to learn from the N-ary relational data with mixed arity. Next, we present a new sparsifying method to address the over-parameterization issue in existing tensor decomposition models but maintain the expressiveness. Experimental results on benchmark data sets demonstrate the effectiveness and efficiency of our proposed model S2S.

For future works, one interesting direction is to incorporate the N-ary relational data into kinds of applications. For example, [9] applies the link prediction task on KGs to the recommendation system. However, it only leverages the binary relational data, which is a special form of N-ary relational data. Since this paper provides a light way to handle the N-ary relational data, we may be able to leverage the web-scale KBs to improve the performance of those applications. Another direction worth trying is to model the N-ary relational data with multi-relational hypergraphs and apply graph neural networks [43]. It could be a more natural way to model the web-scale KBs instead of multiple tensors.

7 ACKNOWLEDGEMENTS

This work is partially supported by National Key Research and Development Program of China Grant no. 2018AAA0101100, the Hong Kong RGC GRF Project 16202218 , CRF Project C6030-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, Didi-HKUST joint research lab project, and Wechat and Webank Research Grants.

REFERENCES

[1] Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, and K. Nishida. 2019. Adaptive Stochastic Natural Gradient Method for One-Shot Neural Architecture Search. In *ICML*. 171–180.
[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.
[3] I. Balazevic, C. Allen, and T. Hospedales. 2019. TuckER: Tensor Factorization for Knowledge Graph Completion. In *EMNLP*. 5188–5197.
[4] Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019. Hypernetwork knowledge graph embeddings. In *ICANN*. Springer, 553–565.
[5] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*. 115–123.
[6] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. 1247–1250.
[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Irreflexive and hierarchical relations as translations. *arXiv preprint arXiv:1304.7158* (2013).
[8] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*. 2787–2795.
[9] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. 2019. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *The world wide web conference*. 151–161.
[10] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *AAAI*.
[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).
[12] Patrick Ernst, Amy Siu, and Gerhard Weikum. 2018. Highlife: Higher-arity fact harvesting. In *WWW*. 1013–1022.
[13] Bahare Fatemi, Perouz Taslakian, David Vazquez, and David Poole. 2019. Knowledge hypergraphs: Prediction beyond binary relations. *arXiv preprint arXiv:1906.00137* (2019).
[14] Saiping Guan, Xiaolong Jin, Jiafeng Guo, Yuanzhuo Wang, and Xueqi Cheng. 2020. NeuInfer: Knowledge Inference on N-ary Facts. In *ACL*. 6141–6151.
[15] Saiping Guan, Xiaolong Jin, Yuanzhuo Wang, and Xueqi Cheng. 2019. Link prediction on n-ary relational data. In *WWW*. 583–593.
[16] W Keith Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. (1970).
[17] Frank L Hitchcock. 1927. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics* 6, 1-4 (1927), 164–189.
[18] F. Hutter, L. Kotthoff, and J. Vanschoren. 2018. *Automated Machine Learning: Methods, Systems, Challenges*. Springer.
[19] S. Kazemi and D. Poole. 2018. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*. 4284–4295.
[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[21] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.
[22] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. 2018. Canonical tensor decomposition for knowledge base completion. *ICML* (2018), 2863–2872.
[23] H. Liu, K. Simonyan, and Y. Yang. 2018. DARTS: Differentiable architecture search. In *ICLR*.
[24] Hanxiao Liu, Yuexin Wu, and Yiming Yang. 2017. Analogical inference for multi-relational embeddings. *arXiv preprint arXiv:1705.02426* (2017).
[25] Yu Liu, Quanming Yao, and Yong Li. 2020. Generalizing Tensor Decomposition for N-ary Relational Knowledge Bases. In *WebConf*. 1104–1114.
[26] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. *arXiv preprint arXiv:1712.01312* (2017).
[27] D. Lukovnikov, A. Fischer, J. Lehmann, and S. Auer. 2017. Neural network-based question answering over knowledge graphs on word and character level. In *WWW*. International World Wide Web Conferences Steering Committee, 1211–1220.
[28] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. 2015. A review of relational machine learning for knowledge graphs. *Proc. IEEE* 104, 1 (2015), 11–33.
[29] Jay Pujara, Eriq Augustine, and Lise Getoor. 2017. Sparsity and noise: Where knowledge graph embeddings fall short. In *EMNLP*. 1751–1756.
[30] Paolo Rosso, Dingqi Yang, and Philippe Cudré-Mauroux. 2020. Beyond triplets: hyper-relational knowledge graph embedding for link prediction. In *WebConf*. 1885–1896.
[31] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. 697–706.
[32] Z. Sun, Z. Deng, J. Nie, and J. Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*.
[33] K. Toutanova and D. Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Workshop on CVSMC*. 57–66.
[34] T. Trouillon, Christopher R., É. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. 2017. Knowledge graph completion via complex tensor factorization. *JMLR* 18, 1 (2017), 4735–4772.
[35] Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 3 (1966), 279–311.
[36] Q. Wang, Z. Mao, B. Wang, and L. Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *TKDE* 29, 12 (2017), 2724–2743.
[37] Y. Wang, R. Gemulla, and H. Li. 2018. On multi-relational link prediction with bilinear models. In *AAAI*.
[38] Z. Wang, J. Zhang, J. Feng, and Z. Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*.
[39] Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. 2016. On the representation and embedding of knowledge bases beyond binary relations. *arXiv preprint arXiv:1604.08642* (2016).
[40] S. Xie, H. Zheng, C. Liu, and L. Lin. 2019. SNAS: stochastic neural architecture search. In *ICLR*.
[41] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th international conference on world wide web*. 1271–1279.
[42] Y. Xue, Y. Yuan, Z. Xu, and A. Sabharwal. 2018. Expanding holographic embeddings for knowledge completion. In *NeurIPS*. 4491–4501.
[43] Naganand Yadati. 2020. Neural Message Passing for Multi-Relational Ordered and Recursive Hypergraphs. *Advances in Neural Information Processing Systems* 33 (2020).
[44] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

[45] Q. Yao and M. Wang. 2019. *Taking human out of learning applications: A survey on automated machine learning.* Technical Report. arXiv preprint.
[46] Q. Yao, J. Xu, W. Tu, and Z. Zhu. 2020. Efficient Neural Architecture Search via Proximal Iterations. In *AAAI.*
[47] F. Zhang, N. Jing Yuan, D. Lian, X. Xie, and W. Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *SIGKDD.* ACM, 353–362.
[48] Richong Zhang, Junpeng Li, Jiajie Mei, and Yongyi Mao. 2018. Scalable instance reconstruction in knowledge bases via relatedness affiliated embedding. In *WWW.* 1185–1194.
[49] S. Zhang, Y. Tay, L. Yao, and Q. Liu. 2019. Quaternion knowledge graph embeddings. In *NeurIPS.* 2731–2741.
[50] Y. Zhang, Q. Yao, W. Dai, and L. Chen. 2020. AutoSF: Searching Scoring Functions for Knowledge Graph Embedding. In *ICDE.* IEEE.
[51] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. 2016. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535* (2016).

A THEORETICAL ANALYSIS OF THEOREM 1

We first introduce two lemmas that will be used in the proof of Theorem 1.

Lemma 2. *Given any N-ary relational data S on the entity set E and relation set R, n-CP [22] can accurately represents the ground truth with $|S|$-dimensional embeddings, such as $\boldsymbol{E}, \boldsymbol{R} \in \mathbb{R}^{|S|}$.*

Proof. For any k-th fact in N-ary relational data S, such that $s = (r_{i_r}, e_{i_1}, \ldots, e_{i_n})$. Let the k-th element of $\boldsymbol{r}_{i_r}, \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n}$ be 1, and set the k-th element of other $\boldsymbol{r} \in \boldsymbol{R}$ and $\boldsymbol{e} \in \boldsymbol{E}$ not involved in s to 0. Then, n-CP [22] can accurately predict the given fact $s = (r_{i_r}, e_{i_1}, \ldots, e_{i_n})$ is plausible if and only if $\langle \boldsymbol{r}_{i_r}, \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n} \rangle \geq 1$, otherwise the fact is not fake. If $\langle \boldsymbol{r}_{i_r}, \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n} \rangle \geq 1$, there must at least have one dimension k leads to $[\boldsymbol{r}_{i_r}]_k = [\boldsymbol{e}_{i_1}]_k = \cdots = [\boldsymbol{e}_{i_n}]_k = 1$. Therefore, the given fact $(r_{i_r}, e_{i_1}, \ldots, e_{i_n})$ is the k-th fact in the data S. Similarly, if $(r_{i_r}, e_{i_1}, \ldots, e_{i_n})$ exists, there must have $\langle \boldsymbol{r}_{i_r}, \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n} \rangle \geq 1$. □

Lemma 3. *The n-CP [22] can be viewed as a special case of the S2S sparse core tensor.*

Proof. Given the embedding $\boldsymbol{H} = \{\boldsymbol{E} \in \mathbb{R}^{n_e \times d}, \boldsymbol{R} \in \mathbb{R}^{n_e \times d}\}$, we first segment the embedding into m parts, such as $\boldsymbol{e}_i = [\boldsymbol{e}_i^{(1)}; \cdots; \boldsymbol{e}_i^{(m)}]$. Then, n-CP's [22] scoring function to measure $s = (r_{i_r}, e_{i_1}, \ldots, e_{i_n})$ is defined as:

$$f(s, \boldsymbol{H}) = \left\langle \boldsymbol{r}_{i_r}, \boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n} \right\rangle = \sum_{j=1}^{m} \left\langle \boldsymbol{r}_{i_r}^{(j)}, \boldsymbol{e}_{i_1}^{(j)}, \ldots, \boldsymbol{e}_{i_n}^{(j)} \right\rangle. \tag{10}$$

Next we prove that (10) is a special case of S2S's scoring function, which is initially defined with a sparse core tensor $\mathcal{Z}^n = \{\mathcal{Z}_k^n\}_{k=1}^K$ as:

$$f_z(\boldsymbol{H}, s; \mathcal{Z}^n) = \sum_{j_r, j_1, \ldots, j_n} \mathcal{Z}_k^n \times_1 \boldsymbol{r}_{i_r}^{j_r} \times_2 \boldsymbol{e}_{i_1}^{j_1} \times_3 \cdots \times_{n+1} \boldsymbol{e}_{i_n}^{j_n}, \tag{11}$$

where $j_r, j_1, \ldots, j_r \in \{1, \ldots, m\}$ and $\mathcal{Z}_k^n \in \mathtt{OP} = \{\mathcal{I}_{-1}^n, \mathcal{I}_0^n, \mathcal{I}_1^n\}$. Because $\mathcal{I}_v$ is super-diagonal with v, (11) actually perform the tensor computation as follows:

$$\begin{aligned} f_z(\boldsymbol{H}, s; \mathcal{Z}^n) &= \sum_{j_r, j_1, \ldots, j_n} \mathcal{Z}_k^n \times_1 \boldsymbol{r}_{i_r}^{j_r} \times_2 \boldsymbol{e}_{i_1}^{j_1} \times_3 \cdots \times_{n+1} \boldsymbol{e}_{i_n}^{j_n}, \\ &= \sum_{j_r, j_1, \ldots, j_n} v \cdot \left\langle \boldsymbol{r}_{i_r}^{(j_r)}, \boldsymbol{e}_{i_1}^{j_1}, \ldots, \boldsymbol{e}_{i_n}^{j_n} \right\rangle. \end{aligned}$$

Then, we let $v = 1$ if and only if $j_r = j_1 = \cdots = j_n$. The above equation will converted to:

$$f_z(\boldsymbol{H}, s; \mathcal{Z}^n) = \sum_{j_r = j_1 \cdots = j_n = 1}^{m} 1 \cdot \left\langle \boldsymbol{r}_{i_r}^{(j_r)}, \boldsymbol{e}_{i_1}^{j_1}, \ldots, \boldsymbol{e}_{i_n}^{j_n} \right\rangle,$$

that is exactly same with $f(s, \boldsymbol{H})$ in (10). Therefore, n-CP [22] is actually a special case of S2S. □

According to Lemma 2, n-CP [22] is expressive enough to handle any N-ary relational data, and n-CP is a special case of S2S as shown in Lemma 3. Therefore, S2S has the sparse core tensor to represent the ground truth of any N-ary relational data.

B EXPERIMENTAL IMPLEMENTATION

B.1 Embedding Sharing in Other Tensor Decomposition Models

In Section 5.7.2, we implement the embedding sharing idea mentioned in Sec 3.1 into other tensor decomposition models. Here we briefly introduce the exact implementation.

Same with S2S, given the maximum arity N and number of segments M, we segment embeddings into M splits, i.e., $\boldsymbol{e}_i = [\boldsymbol{e}_i^1; \ldots; \boldsymbol{e}_i^M]$. Then, given a fact s with arity n, we utilize first m-th (i.e., $m = \min\{n, M\}$) segments of embeddings to compute the score in DistMult [44] and GETD [25]. The corresponding DistMult's scoring functions is defined as:

$$f(s, \boldsymbol{H}) = \sum_{j=1}^{m} \left\langle \boldsymbol{r}_{i_r}^j, \boldsymbol{e}_{i_1}^j, \ldots, \boldsymbol{e}_{i_n}^j \right\rangle.$$

Moreover, the TuckER's scoring function is defined as:

$$\begin{aligned} f(s, \boldsymbol{H}) &= \mathcal{G}^n \times_1 \boldsymbol{r}_{i_r}^{1:m} \times_2 \boldsymbol{e}_{i_1}^{1:m} \times_3 \cdots \times_{n+1} \boldsymbol{e}_{i_n}^{1:m} \\ &\approx \mathrm{TR}(\mathcal{W}_1, \cdots, \mathcal{W}_c) \times_1 \boldsymbol{r}_{i_r}^{1:m} \times_2 \boldsymbol{e}_{i_1}^{1:m} \times_3 \cdots \times_{n+1} \boldsymbol{e}_{i_n}^{1:m} \end{aligned}$$

where $\boldsymbol{r}_{i_r}^{1:m}, \boldsymbol{e}_i^{1:m}$ represent the vector with first m-th segments (e.g., $\boldsymbol{e}_i^{1:m} = [\boldsymbol{e}_i^1; \ldots; \boldsymbol{e}_i^m]$), and $\mathcal{G}^n$ is a $n+1$-order Tucker core tensor with size md/M (e.g., $\mathcal{G}^2 \in \mathbb{R}^{2d/M \times 2d/M \times 2d/M}$). Then, $\mathrm{TR}(\cdot)$ is achieved by Tensor Ring computation [51] as mentioned in Section 2.2.

B.2 Sparsify Core Tensor with L0 Constraint

Here we introduce the details of S2S(L0-reg), i.e., how to sparsify the core tensor with ℓ_0 constraint as in Section 5.7.4. To optimize the core We first give the optimization objective as:

$$\arg\min_{\mathcal{Z}} L(\boldsymbol{H}, \mathcal{Z}; S_{\mathrm{val}}) + \epsilon \left\| \mathcal{Z} \right\|_0, \tag{12}$$

where ϵ is a trade-off weight for the multi-class log loss L and regularization. The ℓ_0 norm penalizes the number of non-zero entries in the core tensor $\mathcal{Z} = \{\mathcal{Z}^n\}_{n=2}^N$ (e.g., $\mathcal{Z}_{j_r, j_1, \ldots, j_n}^n \neq 0$). Note that S2S(L0-reg) has the same number of non-zero elements as S2S, i.e., m^{n+1} for $\mathcal{Z}^n$. Optimizing above objective is computationally intractable because of its non-differentiability and the exponential complexity. To minimize the objective, we adopt the technique proposed in [26], which utilizes the reparameterization trick to make it differentiable.