Efficient Relation-aware Scoring Function Search for Knowledge Graph Embedding

Shimin DI[1], Quanming YAO[2,3], Yongqi ZHANG[2], Lei CHEN[1]
[1]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[2]4Paradigm Inc. Hong Kong SAR, China
[3]Department of Electronic Engineering, Tsinghua University, Beijing, China
{sdiaa,leichen}@cse.ust.hk, {yaoquanming,zhangyongqi}@4paradigm.com

***Abstract*—The scoring function, which measures the plausibility of triplets in knowledge graphs (KGs), is the key to ensure the excellent performance of KG embedding, and its design is also an important problem in the literature. Automated machine learning (AutoML) techniques have recently been introduced into KG to design task-aware scoring functions, which achieve the state-of-the-art performance in KG embedding. However, the effectiveness of searched scoring functions is still not as good as desired. In this paper, observing that existing scoring functions can exhibit distinct performance on different semantic patterns, we are motivated to explore such semantics by searching relation-aware scoring functions. But the relation-aware search requires a much larger search space than the previous one. Hence, we propose to encode the space as a supernet and propose an efficient alternative minimization algorithm to search through the supernet in a one-shot manner. Finally, experimental results on benchmark datasets demonstrate that the proposed method can efficiently search relation-aware scoring functions, and achieve better embedding performance than state-of-the-art methods.** [1]

***Index Terms*—Knowledge Graph, Knowledge Graph Embedding, Neural Architecture Search, Automated Machine Learning**

I. Introduction

Knowledge Graph (KG) [1], [2], as one of the most effective ways to explore and organize knowledge base, applies to various problems, such as question answering [3], recommendation [4], and few-shot learning [5]. In KGs, every edge represents a knowledge triplet in the form of *(head entity, relation, tail entity)*, or (h, r, t) for simplicity. Given a triplet, several crucial tasks in KGs, such as link prediction and triplet classification [1], [2], can be used to verify whether such a fact exists to form this triplet. KG embedding has been proposed to address this issue. Basically, KG embedding targets to embed entities h, t and relations r into low-dimensional vector space such as $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^d$. Then based on the embeddings, a scoring function f is employed to compute a score $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$ to verify whether a triplet (h, r, t) is a fact. Triplets with higher scores are more likely to be facts.

In the last decade, various scoring functions have been proposed to significantly improve the quality of embeddings [1], [2], [6]. TransE [7], as a representative scoring function, interprets the relation r as a translation from head entity h to tail entity t, and optimizes the embeddings to satisfy $\boldsymbol{h} + \boldsymbol{r} = \boldsymbol{t}$. However, TransE [7] and its variants [8], [9] are not fully expressive and their empirical performance is inferior to the others as mentioned in [10]. Recently, some works [11]–[13] employ neural networks to design universal scoring functions. But these scoring functions are not well-regularized for the KG properties and are expensive to general predictions [14]. Furthermore, bilinear models (BLMs) [15]–[19] are proposed to compute the score by the weighted sum of pairwise interactions of embeddings. Currently, BLMs are the most powerful in terms of both empirical results and theoretical guarantees [19] on expressiveness [10], [17].

Generally, the models in BLMs share the form as $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \boldsymbol{h}^\top g(\boldsymbol{r})\boldsymbol{t}$, where $g(\boldsymbol{r})$ returns a square matrix referring to the relation $\boldsymbol{r}$. DistMult [15] regularizes $g(\boldsymbol{r})$ to be diagonal, such as $g(\boldsymbol{r}) = \text{diag}(\boldsymbol{r})$, in order to solve the overfitting problem. ComplEx [16] extends the embeddings to be complex values. SimplE [17] is another variant that regularizes the matrix $g(\boldsymbol{r})$ with a simpler but valid constraint. More recently, TuckER [20] extends BLMs to tensor models for KG embedding. However, the designing of scoring functions is still challenging because of the diversity of KGs [2]. A scoring function performs well on one task may not adapt well to the other tasks since different KGs usually own distinct patterns [14], especially the relation patterns [2].

Recently, automated machine learning (AutoML) [21], [22], as demonstrated via automated hyperparameter tuning [23] and neural architecture search (NAS) [24], has shown to be very useful in the design of machine learning models. Inspired by such a success, a pioneered work, AutoSF [14], has proposed to search an appropriate scoring function for any given KG data using AutoML techniques. AutoSF first defines the manual scoring functions design problem as a scoring function search problem and then proposes a search algorithm. It empirically shows that the searched scoring functions are KG dependent and outperform the state-of-the-art ones designed by human experts. In short, AutoSF can search for proper scoring functions, which depend on the given KG data and evaluation metric. In other words, AutoSF is task-aware, while traditional scoring functions are not.

However, the task-aware method AutoSF still follows the classic way that forcing all relations to share one scoring function. This is not relation-aware and may cause the effectiveness issue. Generally, common KG relations can be

[1]The work was done when S. Di was an intern in 4Paradigm Inc. mentored by Q. Yao; and Correspondence is to Q.Yao.

TABLE I: The summary of existing scoring functions. The expressiveness measures whether a scoring function can handle common patterns in KGs: symmetry, anti-symmetry, general asymmetry, inversion. We compare the inference cost on single triplet of scoring functions w.r.t. the embedding dimension d. N_e and N_r denote the number of entities and relations, respectively.

	Scoring functions	**Effectiveness**			**Efficiency**	
		expressive	task-aware	relation-aware	inference time	model complexity
Hand-designed	TransE [7]	×	×	×	$O(d)$	$O(N_e d + N_r d)$
	DistMult [15]	×	×	×	$O(d)$	$O(N_e d + N_r d)$
	NTN [11]	√	×	×	$O(d^2)$	$O(N_e d + N_r d^2)$
	TuckER [20]	√	×	×	$O(d^3)$	$O(d^3 + N_e d + N_r d)$
	ComplEx [16]	√	×	×	$O(d)$	$O(N_e d + N_r d)$
	HypER [25]	√	×	×	$O(d^2)$	$O(N_e d + N_r d)$
Searched (by AutoML)	AutoSF [14]	√	√	×	$O(d)$	$O(N_e d + N_r d)$
	ERAS	√	√	√	$O(d)$	$O(N_e d + N_r d)$

roughly categorized into different patterns based on their semantic properties, such as symmetry [15], anti-symmetry [16], [26], general asymmetry [18], and inversion [17]. Traditional models design universal scoring functions to cover more and more relation patterns. For example, DistMult [15] only handles symmetric relations, while TransE [7] covers another three kinds of relations except for symmetric relations. Furthermore, ComplEx [16] and SimplE [17] are able to cover all these four common relation patterns. Intuitively, the more patterns covered by the scoring function, the stronger ability it has to learn KGs. However, there are some potential risks in the pursuit of universal scoring functions. A universal scoring function may not perform well on certain relation patterns even though it can handle all kinds of relations [27]. For instance, it has been reported in [28] that HolEX [29] achieves unsatisfactory performance on symmetric relations in the FB15K data set [7], despite HolEX being a universal scoring function. This indicates that forcing all relations to share one scoring function may not be able to fully express the interactions between entities with different relations. As compared in Table I, none of the existing methods cover all the aspects in terms of the effectiveness.

Unfortunately, it is hard to extend the task-aware method (i.e., AutoSF [14]) to be relation-aware due to the efficiency issue. AutoSF adopts a progressive greedy search approach to find a universal scoring function. It requires separately training hundreds of scoring functions to convergence, which suffers from a lot of computational overhead. In general, AutoSF takes more than one GPU day to search on the smallest benchmark data set WN18RR, while it requires more than 9 GPU days on the larger data set YAGO. But the relation-aware search problem requires a much larger search space than the space of AutoSF. Therefore, a much more efficient search algorithm is needed for the relation-aware search.

In this paper, to address the issues mentioned above, we propose the *E*fficient search method for *R*elation-*A*ware *S*coring functions (ERAS) in KG embedding. We propose to search multiple scoring functions, which are expressive, task-aware, and relation-aware, for common relation patterns in any given KG data. More concretely, we propose a supernet to model the relation-aware search space and introduce an efficient search approach to the supernet. We suggest sharing KG embeddings on the supernet, so as to avoid training hundreds of candidate scoring functions from scratch as AutoSF does. In summary, we list the contribution we have made in this work as follows:

- Previous works mainly emphasize the expressiveness of scoring functions, which also motivates AutoSF to design task-aware scoring functions. However, they ignore that scoring functions should also be relation-aware as they model the semantics of relations. In this paper, to address such a problem, we propose an AutoML-based method to design relation-aware scoring functions.
- We define a novel supernet representation to model the relation-aware search space, where the relations are assigned into different groups and each group has a unique scoring function. The simple but effective supernet not only enables us to share KG embeddings to significantly accelerate the search but also protects our search from negative effects.
- Inspired by the one-shot architecture search (OAS) algorithms, we propose a stochastic algorithm, i.e., ERAS, that is efficient and suitable for the automated scoring function search task. It optimizes the search problem through alternative minimization, where embeddings are stochastically updated in the supernet, groups are assigned by Expectation-Maximization clustering, and scoring functions are updated by reinforcement learning.
- We conduct extensive experiments on five popular benchmark data sets on link prediction and triplet classification tasks. Experimental results demonstrate that ERAS can achieve state-of-the-art performance by designing relation-aware scoring functions. Especially, ERAS can consistently outperform literature at the relation type level of a given KG data. Besides, the search is much more efficient compared with AutoSF and the other popular automated methods.

II. Related Works

A. Neural Architecture Search (NAS)

1) General Principles: Generally, Neural Architecture Search (NAS) [21], [22], [24] is formed as a bi-level optimization problem [30] where we need to update the neural architectures on the upper-level and train the model parameters in the lower-level. Subsequently, three important aspects should be considered in NAS:

- *Search space:* it defines what network architectures in principle should be searched, e.g., Convolutional Neural

TABLE II: Notations.

Symbols	Meanings
E, R	The entity set and relation set.
S	The KB triples $\{(h, r, t)\}$, where $h, t \in E$ and $r \in R$.
$\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle$	The triple-dot product $\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle = \sum_i h_i \cdot r_i \cdot t_i$.
$\boldsymbol{\omega} = \{\boldsymbol{E}, \boldsymbol{R}\}$	The set of embeddings $E \in \mathbb{R}^{N_e \times d}$ and $R \in \mathbb{R}^{N_r \times d}$.
f	The scoring function, such as $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$.
$\mathcal{M}$	The performance measurement.
M, N	The number of relation blocks and relation groups.
$\mathcal{O}$	The operation set $\mathcal{O} \equiv \{\boldsymbol{0}, \pm\boldsymbol{r}_1, \cdots, \pm\boldsymbol{r}_M\}$.
$\boldsymbol{A}$	The weight of architecture.
$\boldsymbol{B}$	The weight of relation assignment.

Networks (CNNs) [31] and Recurrent Neural Networks (RNNs) [32]. A well-defined search space should be expressive enough to enable powerful models to be searched. But it cannot be too large to search.

- *Search algorithm:* it aims to efficiently search in the above space, e.g., bayesian optimization [33], reinforcement learning [24], evolution algorithm [34]. A search algorithm is required to perform an efficient search over the search space and be able to find architectures that achieve good performance.
- *Evaluation mechanism:* it determines how to evaluate the searched architectures in the search strategy. Fast and accurate evaluation of candidate architectures can significantly boost the search efficiency.

Unfortunately, classic NAS [24], [34] methods are computationally consuming since candidate architectures are trained by the *stand-alone* way, i.e., many architectures are trained from scratch to convergence and are evaluated separately.

2) One-shot Architecture Search (OAS): More recently, One-shot Architecture Search (OAS) methods [35]–[37], have been proposed to significantly reduce the search cost in NAS. OAS first represents the whole search space by a *supernet* [35], which is formed by a directed acyclic graph (DAG), where the nodes are the operations in neural networks (e.g., 3×3 conv in CNNs). Every neural architecture in the space can be represented by a path in the DAG. Then, instead of training independent model weights of each candidate architecture like the stand-alone approach, OAS keeps weights for the supernet and forces different architectures to share the same weights if they share the same edges in the DAG (i.e., *parameter-sharing* [35], [38]). In this way, architectures can be searched by training the supernet once (i.e., the *one-shot* manner), which makes NAS much faster.

Generally, OAS methods unify the search space with the form of DAG but differ in their way to search the optimal subgraph of DAG. Sampling OAS (e.g., ENAS [35]) employs a controller to sample architectures and search an optimal subgraph of the whole DAG, which maximizes the expected reward of the subgraph on the validation set. Instead of involving controllers, differentiable OAS (e.g., DARTS [36] and NASP [37]) relaxes the search space to be continuous so that the architectures can be optimized by gradient descent. However, differentiable OAS may not able to derive an architecture that results in high evaluation performance when the evaluation metric is not differentiable. In comparison, sampling OAS is more suitable for the non-differentiable scenario since it utilizes the policy gradient [39] to optimize the controller.

B. AutoSF: Searching Task-aware Scoring Functions

Given a KG, it is very empirical to choose a suitable scoring function from the above manual methods. To better adapt to different KG tasks, AutoSF [14] leverages the AutoML techniques to design and customize a proper scoring function on the given KG.

1) Search Problem: Motivated by the expressiveness guarantee and computational efficiency of BLMs, AutoSF proposes to partition embeddings $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^d$ into M splits (e.g., $\boldsymbol{h} = [\boldsymbol{h}_1; \cdots; \boldsymbol{h}_M]$ where $\boldsymbol{h}_i \in \mathbb{R}^{d/M}$), and represents scoring functions as:

$$f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \sum\nolimits_{i=1}^{M} \sum\nolimits_{j=1}^{M} \langle \boldsymbol{h}_i, \boldsymbol{o}, \boldsymbol{t}_j \rangle, \tag{1}$$

where $\boldsymbol{o} \in \mathcal{O}$ with $\mathcal{O} \equiv \{\boldsymbol{0}, \pm\boldsymbol{r}_1, \cdots, \pm\boldsymbol{r}_M\}$. Note that $\langle \boldsymbol{h}_i, \boldsymbol{o}, \boldsymbol{t}_j \rangle$ computes the triple-dot product and is named as the *multiplicative item*. Then previous outstanding scoring functions [15]–[18], [40] can be unified in f with different choices of $\boldsymbol{o}$ [14]. This indicates that f is general enough to represent good scoring functions which are designed manually. In this way, AutoSF generalizes from human wisdom and allows the discovery of better scoring functions, which are not visited in the literature. Subsequently, the search problem is defined as:

Definition 1 (AutoSF problem [14]): Let $\bar{f}$ denote the desired scoring function and $\mathcal{F}_a$ denote the set of all possible scoring functions in AutoSF expressed by (1)*. Then the scoring function search problem is defined as follows:*

$$\bar{f} = \arg\max\nolimits_{f \in \mathcal{F}_a} \mathcal{M}_{val}\left(f, \bar{\boldsymbol{\omega}}; S_{val}\right)$$
$$s.t.\ \bar{\boldsymbol{\omega}} = \arg\max\nolimits_{\boldsymbol{\omega}} \mathcal{M}_{tra}\left(f, \boldsymbol{\omega}; S_{tra}\right),$$

where $\mathcal{M}_{tra}$ and $\mathcal{M}_{val}$ measure the performance of scoring function f and KG embeddings $\boldsymbol{\omega}$ on corresponding KG data S (e.g., training set S_{tra} and validation set S_{val}), respectively.

Given the embeddings $\bar{\boldsymbol{\omega}}$ learned on the training data S_{tra}, AutoSF aims to search for a better scoring function f which leads to higher performance on the validation set S_{val}. Hence, AutoSF can find task-aware scoring functions that can achieve impressive performance on different KG tasks. However, it is non-trivial to efficiently search a proper scoring function from the AutoSF's search space due to its size $O((2M+1)^{M^2})$.

2) Search Algorithm: Since a large number of possible scoring functions exist in the unified scoring function search space, AutoSF then proposes a progressive greedy search algorithm to find a proper scoring function according to an inductive rule:

$$f^b = f^{b-1} + \langle \boldsymbol{h}_i, \boldsymbol{o}, \boldsymbol{t}_j \rangle, \tag{2}$$

where b is the burget of non-zero multiplicative terms (i.e., $\boldsymbol{o} \neq \boldsymbol{0}$) in (1). The intuition behind (2) is to gradually

TABLE III: Hit@1 (in %) results for existing scoring functions on the link prediction task.

SF Type	Methods	Symmetric relations				Anti-symmetric relations			
		WN18	WN18RR	FB15k	FB15k237	WN18	WN18RR	FB15k	FB15k237
Non-universal	TransE [7]	0.0	0.0	0.0	5.0	51.0	3.0	55.0	27.0
	DistMult [15]	93.0	90.0	73.0	7.0	65.0	9.0	74.0	25.0
Universal	ConvE [12]	93.0	93.0	42.0	1.0	94.0	6.0	61.0	25.0
	TuckER [20]	94.0	93.0	67.0	2.0	95.0	12.0	73.0	22.0
	ComplEx [16]	94.0	94.0	88.0	2.0	95.0	11.0	80.0	23.0
	SimplE [17]	92.0	93.0	74.0	5.0	94.0	5.0	64.0	13.0
	Analogy [18]	93.0	92.0	52.0	6.0	93.0	2.0	66.0	27.0
	AutoSF [14]	93.2	93.5	85.8	5.7	94.8	11.5	81.1	26.7

add nonzero multiplicative terms $\langle \boldsymbol{h}_i, \boldsymbol{o}, \boldsymbol{t}_j \rangle$ to achieve the final desired f. Each greedy step in the search algorithm mainly contains two parts: sampling scoring functions and evaluate the sampled scoring functions. We summarize the search algorithm in Algorithm 1.

Algorithm 1 AutoSF: Progressive Greedy Search of Task-aware Scoring Functions

Input: B: number of nonzero blocks.

1: **for** b in $4, \cdots, B$ **do**
2: Randomly select N scoring functions $\{f^{b-1}\}$;
3: Sample N_1 scoring functions $\{f^b\}$ by adding relation blocks to $\{f^{b-1}\}$ as $f^b = f^{b-1} + \langle \boldsymbol{h}_i, \boldsymbol{o}, \boldsymbol{t}_j \rangle$;
4: Select top-K $\{f^b\}$ based on the *Predictor*.
5: Train the top-K $\{f^b\}$ to convergence separately and update *Predictor* with the evaluated performance.
6: **end for**
7: **return** Scoring function f^B with highest performance.

However, there are several issues in AutoSF. First, the evaluation mechanism in AutoSF is inefficient. In every greedy step, AutoSF trains all candidate scoring functions under budget b to convergence for performance evaluation, i.e., step 5 of Algorithm 1. Then well-trained embeddings of all scoring functions will be discarded in the next greedy search. It wastes a lot of effort to train KG embeddings for performance evaluation. Moreover, within the budget b, the predictor in AutoSF search algorithm can only leverage the prior experience that is smaller than the budget b, i.e., step 4 of Algorithm 1, which may also bring unnecessary KG embeddings training due to inaccurate prediction. Second, AutoSF pursuits a universal scoring function over a given KG data. A universal scoring function that can learn certain relationships does not necessarily mean that this scoring function can perform well on them [27], [28]. This will be further discussed in Section III-A.

III. Problem Definition

In this section, to further illustrate the motivation of relation-aware scoring functions, we first discuss the performance of existing scoring functions at the relation pattern level. Then, we formulate a relation-aware scoring function search problem.

A. Motivation of Relation-aware Scoring Functions

As introduced in Section I, traditional scoring functions and AutoSF try to design universal scoring functions to cover as many as relation patterns as possible. However, being expressive does not mean achieving good performance as relations exhibit different patterns [27], [28]. We summarize the experimental results reported by Figure 14 and 15 from [27] in Table III, which demonstrate the performance (the higher the better) of popular scoring functions on symmetric and anti-symmetric relations (e.g., Table IV).

TABLE IV: Exemplar relations corresponding to relation patterns in the benchmark data sets (see Section V-A1 for details).

Relation Patterns	WN18/WN18RR	FB15k/FB15k237
Symmetric	*similar_to, synset_of*	*spouse_of*
Anti-symmetric	*hypernym, hyponym*	*child_of*

From Table III, it is worth noting that universal scoring functions may perform even worse than non-universal scoring functions at the relation pattern level. First, TransE performs badly on symmetric relations on all benchmark data sets [7], [12], [41] since it cannot handle the symmetric relations. But it achieves better performance on symmetric relations of FB15k237 [41] than several universal scoring functions, such as ConvE, TuckER, ComplEx.

Second, DistMult only covers symmetric relations. Therefore, DistMult achieves good performance on symmetric relations, while it performs unsatisfactorily on anti-symmetric relations. However, as reported in Table III, there are several cases that universal scoring functions perform worse than DistMult on anti-symmetric relations:

1. ConvE, SimplE, and Analogy in WN18RR [12].
2. ConvE, SimplE, and Analogy in FB15K [7].
3. TuckER, ComplEx, SimplE, and AutoSF in FB15k237 [41].

In summary, universal scoring functions may achieve unsatisfactory performance on specific relation patterns of certain KG. Such observation motivates us to design relation-aware scoring functions.

B. Problem Formulation

Inspired by the observation in Section III-A and the task-aware method AutoSF, we here propose to search relation-aware scoring functions for different relation patterns over any given KG data.

Recall that AutoSF targets to find a scoring function f that can achieve high $\mathcal{M}(f, \boldsymbol{\omega}; S)$ for given triplets S. But in relation-aware search, it is also important to assign relations to appropriate groups to better cover relation patterns. Let $\boldsymbol{B} \in \{0,1\}^{N_r \times N}$ record the relation assignment, where $B_{rn} = 1$ if the $r \in R$ is assigned to n-th group otherwise $B_{rn} = 0$, and f_n denote the scoring function for relations in n-th group. Then, relation-aware search aims to find a set of scoring functions $\{f_n\}$ and relation assignments $\boldsymbol{B}$ that can achieve high $\mathcal{M}(\{f_n\}, \boldsymbol{B}, \boldsymbol{\omega}; S)$. Formally, the problem in this paper is defined as:

Definition 2 (ERAS problem): Let N denote the total number of relation groups, and $\mathcal{F}_e$ denote the set of all possible relation-aware scoring functions. Then the relation-aware scoring function search problem is defined as:

$$\{\bar{f}_n\}_{n=1}^{N} = \arg\max_{f_n \in \mathcal{F}_e} \ \mathcal{M}_{val}\left(\{f_n\}_{n=1}^{N}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{\omega}}; S_{val}\right), \quad (3)$$

$$s.t. \begin{cases} \bar{\boldsymbol{\omega}} = \arg\max_{\boldsymbol{\omega}} \ \mathcal{M}_{tra}\left(\{f_n\}_{n=1}^{N}, \bar{\boldsymbol{B}}, \boldsymbol{\omega}; S_{tra}\right) \\ \bar{\boldsymbol{B}} = \arg\min_{\boldsymbol{B}} \ \mathcal{L}_B\left(\boldsymbol{B}, \boldsymbol{\omega}\right) \end{cases}, \quad (4)$$

where $\mathcal{L}_B$ is the loss of relation assignments, and $\mathcal{M}_{tra}$, $\mathcal{M}_{val}$ measures the performance on the training set S_{tra} and validation set S_{val}, respectively.

Generally, the relation-aware scoring function search problem is based on the bi-level optimization problem in Definition 2. Compared with the single-level objective, bi-level optimization allows the model to optimize parameters in different ways, which is more suitable for deriving scoring functions, embeddings, and relation assignments. This definition looks similar to AutoSF in Definition 1, but it is quite different in essence. First, we need to assign relations to proper relation groups in the lower-level objective (4). Second, there are multiple targeted scoring functions $\{f_n\}_{n=1}^{N}$ for handling N relation groups. Therefore, the search space $\mathcal{F}_e$ for ERAS with size $O((2M+1)^{NM^2})$ is much larger than that for AutoSF with size $O((2M+1)^{M^2})$. The larger search space requires ERAS to have a more efficient search algorithm.

IV. Search Algorithm

Here, we propose a new algorithm to solve the search problem in Definition 2. We can see that there are three types of parameters need to be simultaneously optimized in (3), i.e.,

- *Group Assignments*: The relation assignment mechanism should be flexible enough to update $\boldsymbol{B}$ during the whole search procedure.
- *Architectures*: We pursue $\{f_n\}_{n=1}^{N}$ with high performance. But it is difficult to maximize (3) because performance evaluation in KG embedding is usually non-differentiable.
- *Embeddings*: Training KG embeddings $\boldsymbol{\omega}$ for evaluating candidate scoring functions consume a lot of computation overhead in AutoSF. It is essential to tackle this issue since ERAS has a much larger search space than AutoSF.

Due to these challenges, neither AutoSF nor other existing NAS algorithms can be applied (see discussions in Section IV-D1 and IV-D2). Thus, we propose to deal with the above challenges using alternative minimization. Specifically, we incorporate three key components in search algorithm: *Expectation-Maximization (EM) clustering* for updating $\boldsymbol{B}$, *policy gradient* for searching $\{f_n\}_{n=1}^{N}$, and *embedding sharing* for updating $\boldsymbol{\omega}$. Details are in the sequel.

A. Update Group Assignments by EM Clustering

In this part, we illustrate how to assign relations to proper groups in the search procedure. Intuitively, relations with similar semantic meanings should be grouped. Since the KG embeddings are designed to encode the semantic meanings [2], [7], we propose to assign relations based on the given KG embeddings $\boldsymbol{\omega}$, i.e., minimizing $\mathcal{L}_B(\boldsymbol{B}, \boldsymbol{\omega})$ in Definition 2. Specifically, given a set of relations R and N groups, let $\boldsymbol{c}_n$ denote the vector representation of the n-th group (i.e., $\boldsymbol{C}$ for all groups), and B_{rn} define the degree of membership between the relation r with n-th group. Then, the objective $\mathcal{L}_B$ for relation clustering in Definition 2 is defined as:

$$\min_{\boldsymbol{B}} \mathcal{L}_B(\boldsymbol{B}, \boldsymbol{\omega}) = \min_{\boldsymbol{B}, \boldsymbol{C}} \sum\nolimits_r \sum\nolimits_n B_{rn} \|\boldsymbol{r} - \mathbf{c}_n\|^2, \quad (5)$$

which can be solved by the Expectation-Maximization (EM) algorithm [42].

B. Updating Architectures by Reinforcement Learning

Our target in (3) is to derive an optimal relation-aware scoring function $\{\bar{f}_n\}_{n=1}^{N}$, which maximizes the evaluation performance on the given KG data. It is natural that we use Mean Reciprocal Ranking (MRR) on the validation data as the evaluation measurement $\mathcal{M}_{\text{val}}$. However, MRR is non-differentiable, which indicates that directly optimizing (3) by gradient descent (e.g., differentiable OAS [36], [37]) is not suitable (see discussions in Section V-E1).

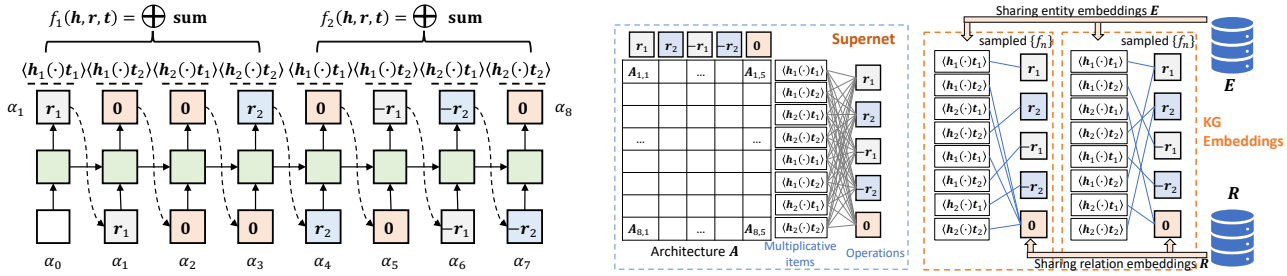
1) Reinforcement Learning Formulation: To handle the non-differentiable MRR, we first formulate the search problem of $\{f_n\}$ as a multi-step decision problem. Then we adopt reinforcement learning to solve (3).

Recall that each f_n is a summation of multiplication terms $\langle \boldsymbol{h}_i, \boldsymbol{o}, \boldsymbol{t}_j \rangle$ in (1). We can sequentially determine which operation $\boldsymbol{o} \in \mathcal{O}$ is selected for the (i,j)-th multiplicative item in f_n. Let v denote the index of all multiplicative items in $\{f_n\}$ and α_v denote the operation selected for the v-th multiplicative item. Then, as shown in Figure 1 (a), the search process of $\{f_n\}$ can be viewed as a multi-step decision problem: a list of tokens $\{\alpha_v\}_{v=1}^{V}$ ($V = NM^2$) that needs to be predicted. It is intuitive to adopt reinforcement learning to solve this problem. Let $\boldsymbol{A} \in \{0,1\}^{NM^2 \times (2M+1)}$, where $A_{vk} = 1$ if α_v chooses k-th operation $\boldsymbol{o}_k$ in $\mathcal{O}$ otherwise $A_{vk} = 0$. Then (3) can be reformulated as:

$$\max_{\theta} \mathcal{J}(\theta) \equiv \mathbb{E}_{\boldsymbol{A} \sim \pi(\boldsymbol{A}; \theta)}[Q\left(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{val}}\right)], \quad (6)$$

where $\pi(\boldsymbol{A}; \theta)$ is a policy parameterized by θ for generating $\boldsymbol{A}$, and $Q\left(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{val}}\right)$ measures MRR performance as:

$$Q\left(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{val}}\right) = \sum\nolimits_n \sum\nolimits_{(h,r,t) \in S_{\text{val}}} B_{rn} \cdot q(f_n(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})).$$



(a) Modelling the generation of f as a multi-step decision process. (b) Architecture of supernet. (c) Illustration to embedding sharing.

Fig. 1: Set $M, N = 2$ in all examples. (a) An example of recurrently generating the relation-aware scoring functions $\{f_1, f_2\}$: $f_1(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_2 \rangle$ and $f_2(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = -\langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_2 \rangle - \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_1 \rangle$; (b) The illustration to supernet in the form of bipartite graph and architecture weight $\boldsymbol{A}$; (c) The example of sharing embeddings between two sampled relation-aware scoring functions.

Note that $q(\cdot)$ measures MRR of a triplet and f_n is the n-th scoring function based on $\boldsymbol{A}$. Then, the gradient of θ in (6) can be computed by REINFORCE gradient [39] as

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{A}\sim\pi(\boldsymbol{A};\theta)} \left[\sum_{v=1}^{V} \nabla_\theta \log P(\alpha_v | \alpha_{(v-1):1}; \theta) Q \right]$$
$$\approx \frac{1}{U} \sum_{u=1}^{U} \sum_{v=1}^{V} \nabla_\theta \log P(\alpha_v | \alpha_{(v-1):1}; \theta)(Q_u - b), \quad (7)$$

where Q_u denotes $Q(\boldsymbol{A}^u, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{val}})$, $\boldsymbol{A}^u$ is u-th sampled architecture from $\pi(\boldsymbol{A}; \theta)$, b is a moving average baseline to reduce variance, and U denote the number of sampled scoring functions. We can see that solving (6) has been converted to optimize θ in (7), and whether Q is differentiable does not affect the gradient computation w.r.t θ.

Inspired by [24], [35], we also adopt a Long Short-term Memory (LSTM) [43] to parameterize θ for learning the policy $\pi(\boldsymbol{A}; \theta)$. Specifically, the controller samples decisions in an autogressive way: the decided operation α_v in the previous multiplicative item is carried out and then fed into the next to predict α_{v+1} (see Figure 1 (a)). Finally, (7) is used to update the LSTM policy network θ.

2) Encoding Prior of Architectures: To fully evaluate the search space, we expect that every relation segmentation $\boldsymbol{r}_i \in \mathcal{O} \backslash \boldsymbol{0}$ must be selected at least once in the searched scoring functions, named as the *exploitative constraint*. Thus, we constrain $\boldsymbol{A}$ with this exploitative constraint. If the sampled $\boldsymbol{A}$ does not satisfy it, we will directly set the reward Q to 0.

C. Update Embedding in Shared Supernet

AutoSF follows the classic NAS way to evaluate the stand-alone performance of candidate scoring functions, which requires separately training KG embeddings hundreds of times. As discussed in Section II-A2, OAS methods propose a more efficient evaluation mechanism by forcing all candidates sharing parameters. Inspired by OAS works, we design a simple but effective *supernet* that enables candidate scoring functions sharing the KG embeddings for accelerating search.

1) Design of Supernet: To enable fast evaluation, we propose a supernet view of the relation-aware scoring function search space. Specifically, the f_n is represented as:

$$f_n(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \sum_i \sum_j \sum_k A_{vk} \cdot \langle \boldsymbol{h}_i, \boldsymbol{o}_k, \boldsymbol{t}_j \rangle. \quad (8)$$

Recall that v is the index of multiplicative items in $\{f_n\}$ and k is the index of operations in $\mathcal{O}$. Then, as shown in Figure 1 (b), we can take $\boldsymbol{A}$ as the adjacency matrix of a bipartite graph (i.e., supernet) from above (8), where multiplicative items and operations are nodes, and A_{vk} records the edge weight between multiplicative items and operations. Based on the supernet design, Figure 1 (c) illustrates that any relation-aware $\{f_n\}$ can be realized by taking subgraphs of the supernet. Then ERAS forces all subgraphs to share embeddings, thereby different scoring functions can be evaluated based on the same KG embedding. It enables us to evaluate candidate scoring functions faster by avoiding repetitive embedding training.

2) Update Embeddings: Given the fixed controller's policy $\pi(\boldsymbol{A}; \theta)$ and relation assignment $\boldsymbol{B}$, we propose to solve $\mathcal{M}_{\text{tra}}$ in objective (4) by minimizing the expected loss $\mathcal{L}$ on the training data, such as $\mathbb{E}_{\boldsymbol{A}\sim\pi(\boldsymbol{A};\theta)}[\mathcal{L}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{tra}})]$. Then stochastic gradient descent (SGD) can be performed to optimize $\boldsymbol{\omega}$. We approximate the gradient $\nabla_{\boldsymbol{\omega}}$ according to:

$$\nabla_{\boldsymbol{\omega}} \mathbb{E}_{\boldsymbol{A}\sim\pi(\boldsymbol{A};\theta)}[\mathcal{L}] \approx \frac{1}{U} \sum_{u=1}^{U} \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{A}^u, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{tra}}), \quad (9)$$

where U is the number sampled scoring functions, and

$$\mathcal{L}(\boldsymbol{A}^u, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{tra}}) = \sum_n \sum_{(h,r,t)\in S_{\text{tra}}} B_{rn} \cdot \ell(f_n(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})).$$

Note that $\ell(\cdot)$ is the multiclass log-loss [19] and f_n is the n-th scoring function based on sampled $\boldsymbol{A}^u$. Hence, (9) can be represented as:

$$\nabla_{\boldsymbol{\omega}} \mathbb{E}_{\boldsymbol{A}\sim\pi(\boldsymbol{A};\theta)}[\mathcal{L}] \approx \frac{1}{U} \sum_{u=1}^{U} \sum_{n} \sum_{(h,r,t)\in S_{\text{tra}}} \nabla_{\boldsymbol{\omega}} B_{rn} \cdot \ell(f_n(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})).$$

3) Performance Evaluation: Recall that Q_u denotes the reward of the u-th sampled scoring function in (7). In the supernet, every sampled scoring function can be regarded as a subgraph of the supernet where some edges are activated. Hence, we can evaluate Q_u based on the sampled scoring functions by activating the subgraph on the supernet.

TABLE V: Comparison of AutoSF (Algorithm 1) and ERAS (Algorithm 2) in terms of NAS principles (Section II-A1).

	space		**algorithm**	**evaluation**
	size	property		
AutoSF	$O((2M+1)^{M^2})$	task-aware	progressive greedy search	by stand-alone
ERAS	$O((2M+1)^{NM^2})$	task-/relation-aware	alternative minimization (EM cluster + reinforcement learning)	in embedding shared supernet

D. Complete Algorithm

The proposed algorithm ERAS is summarized in Algorithm 2, where KG embedding $\boldsymbol{\omega}$, relation assignments $\boldsymbol{B}$ and architectures $\boldsymbol{A}$ are alternatively updated in every epoch. To improve the efficiency of scoring function search, we represent the search space as a supernet and propose to share KG embeddings across different scoring functions in step 3 (see Section IV-C). Thus, ERAS is capable of avoiding wasting a lot of computation on training embeddings from scratch. To enable relation-aware scoring function search, we introduce EM clustering in step 4 to dynamically assign relations $\boldsymbol{B}$ based on the learned embeddings (see Section IV-A). To handle the non-differentiable measurement of scoring functions, we use reinforcement learning and perform policy gradient in step 5-6 (see Section IV-B). After searching, we derive several sampled scoring functions with the well-trained controller and compute its reward on a mini-batch of the validation data in step 9-11. Finally, we take the scoring functions with the highest reward and re-train it from scratch.

Algorithm 2 ERAS: Efficient Relation-aware Scoring Functions Search.

Input: Initialize embeddings $\boldsymbol{\omega}$, relation groups $\boldsymbol{B}$, and controller's parameter θ.

// search relation- and task-aware scoring functions

1: **while** not converge **do**
2: Sample a set of scoring functions from $\pi(\boldsymbol{A};\theta)$;
3: Update *shared embeddings* $\boldsymbol{\omega}$ with (9);
4: Update *relation assignments* $\boldsymbol{B}$ according to (5);
5: Sample a mini-batch data B_{val} from validation data S_{val};
6: Update *architectures* policy $\pi(\boldsymbol{A};\theta)$ with (7);
7: **end while**

// Derive the final scoring function $\bar{\boldsymbol{A}}$

8: ERAS samples K scoring functions $\mathcal{A}^K$ from $\pi(\boldsymbol{A};\theta)$;
9: **for** $\boldsymbol{A} \in \mathcal{A}^K$ **do**
10: Compute $Q(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\omega}; S_{\text{val}})$;
11: **end for**
12: **return** The scoring function $\bar{\boldsymbol{A}}$ with highest reward on validation set, and train it from scratch to convergence.

1) Comparison with AutoSF: To compare ERAS with the pioneering work AutoSF, we summarize them from the perspective of NAS principles in Table V. First, to capture the inherent properties of relation patterns in KGs, ERAS targets to solve the relation-aware scoring function search problem in this paper. This leads to the larger search space in ERAS than that in AutoSF. Second, ERAS proposes an alternative minimization way to solve the non-differentiable problem of $\mathcal{M}_{\text{val}}$. Finally, AutoSF evaluates every scoring function based on their stand-alone performance, which requires repeatedly training KG embeddings hundreds of times. On the contrary, ERAS shares KG embeddings across different scoring functions, which extremely reduces time expense for performance evaluation of candidate scoring functions.

2) Comparison with Existing OAS Algorithms: Inspired by parameter sharing in OAS works, we propose to share KG embedding in the supernet, thereby ERAS avoids repeatedly training KG embedding hundreds of times. However, there exist some concerns about parameter-sharing in OAS [38], [44]. Specifically, while parameter sharing enables an alternative way to train all architectures in the search space in a cheaper way, it can lead to a biased evaluation problem: the evaluation of candidate architectures in the search phase (i.e., $\mathcal{M}_{\text{val}}$) is inconsistent with the stand-alone training phase. Especially, the correlation between one-shot and stand-alone performance will probably be unstable as the supernet goes deep and complex.

Therefore, to make parameter sharing work for our problem, we first construct a search space, which can be represented by a supernet in (8). Then, to avoid the supernet being deep and complex, we design a shallow and simple supernet with the form of a bipartite graph, which differs from the complex DAG supernet in classic OAS works. We demonstrate that the ERAS's supernet design does not suffer from a biased evaluation problem in Section V-E1. In summary, we leverage the domain-knowledge of KG embedding to make embedding sharing works.

V. Empirical Study

Here we mainly show that ERAS can improve effectiveness in KG embedding with high efficiency, and provide some insight views. All codes are implemented with PyTorch [47] and experiments are run on a single TITAN Xp GPU.

A. Experiment Setup

1) Data Sets: In our experiments, we mainly conduct experiments on five public benchmarks data sets: WN18 [7], WN18RR [12], FB15k [7], FB15k237 [41], and YAGO3-10 [12], that have been employed to compare KG embedding models in [7], [15]–[18]. Note that WN18RR and FB15k237 remove duplicate and inverse duplicate relations from WN18 and FB15k, respectively. The statistics of five data sets are summarized in Table VII.

2) Hyperparameter Settings: The hyperparameters in this work can be mainly categorized into searching and evaluation parameters. To fairly compare existing scoring functions, including human-designed and searched ones, we search a stand-alone parameter set on the SimplE with the help of HyperOpt,

TABLE VI: Comparison of the best scoring functions identified by ERAS and the state-of-the-arts on the link prediction task. The bold numbers mean the best performance and the underline ones mean the second best. [♣]: results are taken from [14]; [†]: from [45]; [‡]: from [11]; §: from [25]; [♢]: from [12]; [∗]: from [20]; [+]: from [29]; [♠]: from [46].

type	model	WN18			WN18RR			FB15k			FB15k237			YAGO3-10		
		MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR	Hit@1	Hit@10
TDMs	TransE♣	0.500	-	94.1	0.178	-	45.1	0.495	-	77.4	0.256	-	41.9	-	-	-
	TransH♣	0.521	-	94.5	0.186		45.1	0.452	-	76.6	0.233	-	40.1	-	-	-
	RotatE†	0.949	94.4	95.9	0.476	42.8	57.1	0.797	74.6	88.4	0.338	24.1	53.3	-	-	-
NNMs	NTN‡	0.53	-	66.1	-	-	-	0.25	-	41.4	-	-	-	-		-
	ConvE♢	0.942	93.5	95.5	0.460	39.0	48.0	0.745	67.0	87.3	0.316	23.9	49.1	0.520	45.0	66.0
	HypER§	0.951	94.7	95.8	0.465	43.6	52.2	0.790	73.4	88.5	0.341	25.2	52.0	0.533	45.5	67.8
TBMs	TuckER ∗	**0.953**	94.9	95.8	0.470	44.3	52.6	0.795	74.1	89.2	0.358	26.6	54.4	-	-	-
	HolEX+	0.938	93.0	94.9	-	-	-	0.800	75.0	88.6	-	-	-	-	-	-
	QuatE♠	0.950	94.5	95.9	0.488	43.8	**58.2**	0.782	71.1	90.0	0.348	24.8	55.0	-	-	-
	DistMult♣	0.821	71.7	95.2	0.443	40.4	50.7	0.817	77.7	89.5	0.349	25.7	53.7	0.552	47.6	69.4
	ComplEx♣	0.951	94.5	95.7	0.471	43.0	55.1	0.831	79.6	90.5	0.347	25.4	54.1	0.566	49.1	70.9
	Analogy♣	0.950	94.6	95.7	0.472	43.3	55.8	0.829	79.3	90.5	0.348	25.6	54.7	0.565	49.0	71.3
	SimplE♣	0.950	94.5	95.9	0.468	42.9	55.2	0.830	79.8	90.3	0.350	26.0	54.4	0.565	49.1	71.0
Rule-based	AnyBURL	0.950	94.6	95.9	0.480	44.6	55.5	0.830	80.8	87.6	0.310	23.3	48.6	0.540	47.7	67.3
AutoML	AutoSF♣	0.952	94.7	96.1	0.490	45.1	56.7	0.853	82.1	91.0	0.360	26.7	55.2	0.571	50.1	71.5
	ERAS$^{N=1}$	0.951	94.7	96.0	0.490	45.0	56.8	0.853	82.0	91.2	0.361	26.6	55.2	0.570	50.2	71.5
	ERAS	**0.953**	**95.0**	**96.2**	**0.492**	**45.2**	56.8	**0.855**	**82.3**	**91.4**	**0.365**	**26.8**	**55.5**	**0.577**	**50.3**	**71.7**

TABLE VII: Summary of KG benchmark data sets.

Data set	#relation	#entity	#training	#validation	#testing
WN18	18	40,943	141,442	5,000	5,000
WN18RR	11	40,943	86,835	3,034	3,134
FB15k	1,345	14,951	484,142	50,000	59,071
FB15k237	237	14,541	272,115	17,535	20,466
YAGO3-10	37	123,188	1,079,040	5,000	5,000

TABLE VIII: Hit@1 results for ERAS$^{N=1}$ and ERAS on the link prediction task at the relation pattern level.

Methods	Symmetric relations			Anti-symmetric relations		
	WN18RR	FB15k	FB15k237	WN18RR	FB15k	FB15k237
Best in Table III	94.0	88.0	7.0	12.0	81.0	27.0
ERAS$^{N=1}$	93.2	86.5	5.3	11.6	80.4	26.9
ERAS	94.3	90.0	8.8	13.2	82.1	27.9

a hyperparameter optimization framework [33]. The tuned parameter set includes: learning rate, L2 penalty, decay rate, batch size, embedding dimensions. Then we compare the stand-alone performance of different scoring functions on the tuned parameter set. Besides, the searching parameters of ERAS are the number of segments M, relation groups N, sampled scoring functions U in (7) and (9). Moreover, we optimize embeddings $\boldsymbol{\omega}$ with Adagrad [48] algorithm and the controller θ with Adam [49] algorithm.

B. Comparison with KG Embedding Methods

As in [14], [20], [46], we perform experiments with link prediction and triplet classification task, as they are important testing bed for scoring functions.

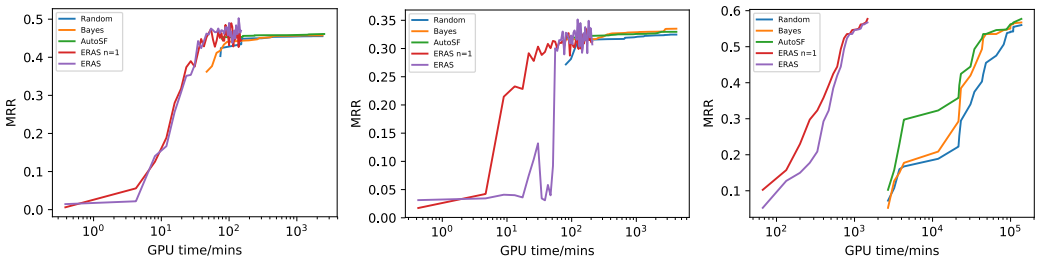
1) Link Prediction: We first test the performance of our proposed method on the link prediction task. This is the test bed to measure KG embedding models and works as an important task in KG completion. Given the triplet $(h, r, t) \in S_{\text{val}} \cup S_{\text{test}}$, the KG embedding model obtains the rank of h through computing the score of (h', r, t) for all entities, and the same for t. We adopt the classic metrics [7], [8]:

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

Note that the higher MRR, Hit@1 and Hit@10 values mean better embedding quality. We compare the proposed ERAS (Algorithm 2) with the popular KG embedding models mentioned in Section I:

- Translational models (TDMs): TransE [7], TransH [8], and RotatE [45];
- Neural network models (NNMs): NTN [11], ConvE [12], and HypER [25];
- Tensor-based models (TBMs): TuckER [20], HolEX [29], QuatE [46], DistMult [15], ComplEx [16], Analogy [18] and SimplE [17];
- The rule-based model: AnyBURL [50];
- The scoring function search method: AutoSF [14].

The comparison of the global effectiveness between ERAS and other methods is in Table VI. Firstly, it is clear that traditional scoring functions, such as TDMs, NNMs, and TBMs, are not task-aware since no scoring functions can perform consistently over the benchmark data sets. This indicates that a single scoring function is hard to adapt to different KGs even though it is a universal scoring function like, as discussed in Table III. The task-aware method AutoSF can search the KG-dependent scoring functions on five data sets and perform consistently better than traditional scoring functions. Then, we compare AutoSF with a variant of ERAS, i.e., ERAS$^{N=1}$, that aims to search a universal scoring function since all relations are assigned into one group (i.e., only task-aware as AutoSF). For five benchmark data set, ERAS$^{N=1}$ shows almost the same performance with AutoSF. Moreover, as a task-aware and relation-aware method, ERAS performs better than AutoSF



(a) WN18RR. (b) FB15K237. (c) YAGO3-10.

Fig. 2: Search efficiency comparison of ERAS with the other popular search algorithms in AutoML.

TABLE IX: Running time analysis of the automated approaches on single GPU in hours.

Methods	AutoSF		ERAS$^{N=1}$		ERAS		Handed-designed	
	greedy search	evaluation	supernet training	evaluation	supernet training	evaluation	DistMult	QuatE
WN18	65.7±3.0	5.5±0.5	3.29±0.2	2.1±0.1	3.54±0.1	2.2±0.1	1.9±0.1	2.0±0.1
FB15K	127.1±5.2	20.5± 1.3	4.55±0.2	19.0±0.2	4.86±0.2	19.49±0.3	8.36±0.2	11.1±0.4
WN18RR	38.6±1.9	3.72±0.6	2.97±0.2	0.50±0.1	3.19±0.1	0.52±0.1	0.42±0.1	0.95±0.1
FB15k237	61.1±2.8	8.5±0.4	3.22±0.1	4.7±0.1	3.54±0.1	4.8±0.2	2.6±0.1	5.0±0.3
YAGO	219.9±5.1	18.9±2.0	17.5±0.3	29.5±1.1	19.8±0.3	30.3±1.9	26.4±1.5	32.6±2.0

and other manually designed scoring functions.

As discussed in Section III-A, the existing scoring functions may achieve unsatisfactory performance on specific relation types of certain KG data. Corresponding to Table III, we investigate the performance of ERAS$^{N=1}$ and ERAS at the relation type level in Table VIII. Obviously, the relation-aware method ERAS can consistently achieve outstanding performance on various relation types of any KG data. Especially, ERAS improves the performance of symmetric relations in FB15k and FB15k237. However, since the fact of symmetric relation only accounts for 3% of the test data in FB15k and FB15k237 [27], the global improvement is not so notable.

2) Triplet Classification: To further demonstrate the effectiveness of ERAS, we also conduct the triplet classification experiments on FB15k, WN18RR, and FB15k237, where the positive and negative triplets are provided. We compare our methods with those that have reported results in public papers. This task aims to answer whether a given (h, r, t) exists or not. In this task, we utilize the accuracy to evaluate the scoring functions. We set the same decision rule of classification in literature [14]: predicting a (h, r, t) is positive if $f(h, r, t) > \theta_r$ otherwise negative, where θ_r is a relation-specific threshold and is inferred by maximizing the accuracy on S_{val}. As shown in Table X, the scoring function searched by relation-aware ERAS consistently outperforms other BLMs or searched scoring functions.

TABLE X: Comparison of the best scoring functions identified by ERAS and the state-of-the-art scoring functions for triplet classification. [♣]: results are taken from [14].

Data set	FB15k	WN18RR	FB15k237
DistMult♣	80.8	84.6	79.8
Analogy♣	82.1	86.1	79.7
ComplEx♣	81.8	86.6	79.6
SimplE♣	81.5	85.7	79.6
AutoSF♣	82.7	87.7	81.2
ERAS	**82.9**	**88.0**	**81.4**

C. Comparison with AutoML Search Methods

ERAS enables embedding sharing to improve the scoring function search efficiency. Hence we also compare the scoring function search efficiency of ERAS and ERAS$^{N=1}$ with other automated search algorithms, i.e. AutoSF [14], random search [44], and Bayes algorithm [33], over three benchmark datasets. As shown in the Figure 2, both ERAS and ERAS$^{N=1}$ complete the search very quickly. That is because other AutoML methods have to train hundreds of candidate scoring functions to convergence, while ERAS and ERAS$^{N=1}$ avoids the time-consuming training by one-shot way. Compared with ERAS, ERAS$^{N=1}$ convergess faster in search procedure since it does not dynamically assign relations and search corresponding scoring functions for different groups. It is worth noting that ERAS has unstable performance at beginning of the search. That is because FB15k237 has much more relations than another 2 data sets. It takes some time to find proper relation assignments at the start. Furthermore, other automated methods can achieve higher performance than ERAS in the search strategy. ERAS consistently updates the candidate scoring functions in every mini-batch data, which results in that the searched scoring functions cannot be well trained with one or several mini-batches. But other automated methods train the searched scoring functions with all training data until convergence.

We also summarize the running time on the five data sets in Table IX. AutoSF sets the embedding dimension d to 64 for all data sets, while ERAS enables much faster search and hence sets $d = 512$. The larger dimensionality enables us to evaluate the scoring function more accurately. As shown in Table IX, the search time of AutoSF is significantly reduced by ERAS$^{N=1}$. But recall that the effectiveness of ERAS$^{N=1}$ is the same with AutoSF in Table VI. This indicates that ERAS$^{N=1}$ can extremely shorten the task-aware search time but maintain the same effectiveness with AutoSF. Moreover, although ERAS searches from a larger search space that is

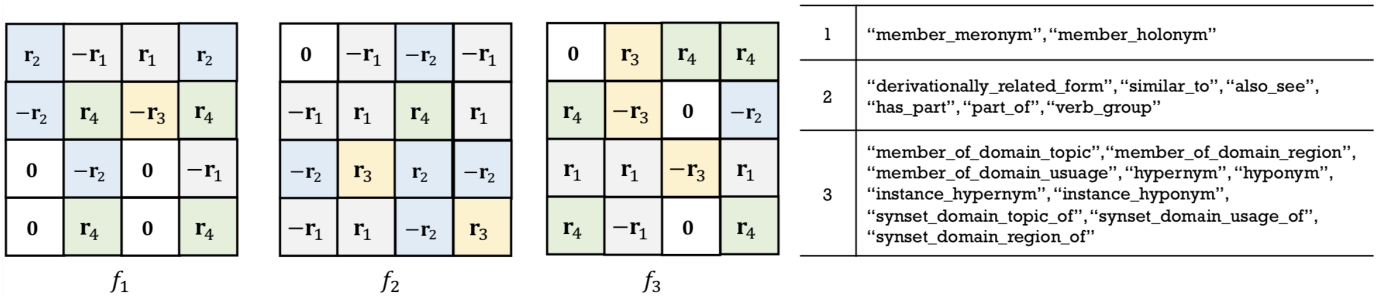


Fig. 3: The example of searched relation-aware scoring functions by ERAS on WN18.

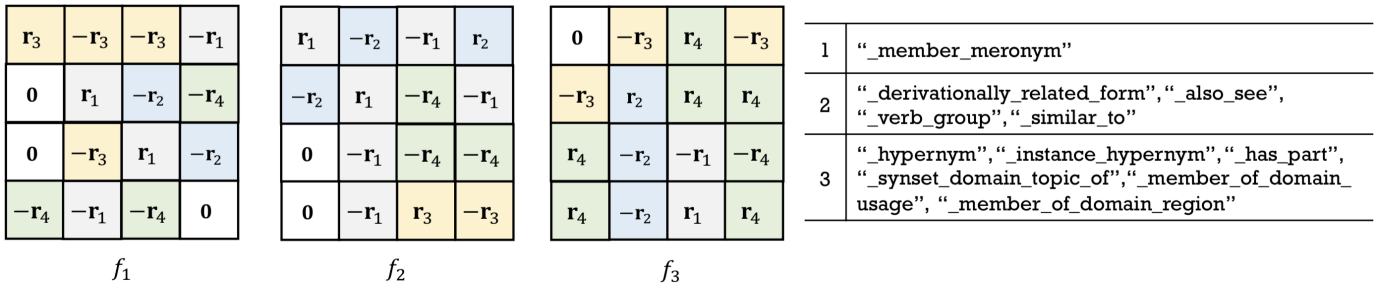


Fig. 4: The example of searched relation-aware scoring functions by ERAS on WN18RR.

relation-aware, it reduces the search time of AutoSF by one order with a large dimension size $d = 512$ and improves effectiveness as in Table VI. In summary, ERAS can more efficiently search for more effective scoring functions.

D. Case Study: The Searched scoring functions

To show the searched scoring functions by ERAS are relation-aware, we use searched scoring functions from WN18 and WN18RR as examples and plot them in Figure 3 and 4. As we can see, the three searched scoring functions have distinct patterns and are relation-aware. Moreover, the relations are grouped into general asymmetry, symmetry, and anti-symmetry. And the three SFs searched have distinct patterns and can handle their corresponding relations.

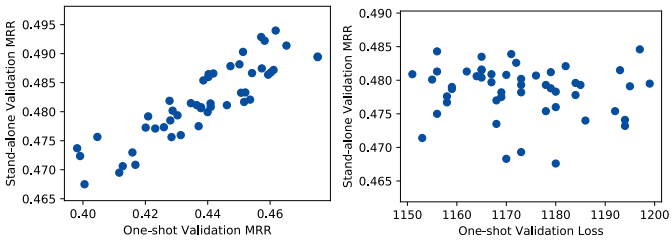
E. Ablation Study

To investigate the influence of different components of ERAS, we conduct several ablation studies.

1) Impact of Evaluation Measurement and Optimization Algorithm: As discussed in Section IV-D2, the deep and complex supernet design will probably lead to the biased evaluation problem. In Figure 5(a), we first demonstrate the correlation between stand-alone validation MRR with one-shot validation MRR (i.e., $\mathcal{M}_{\text{val}}$) of various scoring functions in ERAS. It is obvious that one-shot validation MRR has near positive correlation with the stand-alone validation MRR. Therefore, the simplified design of supernet makes embedding sharing work, and there is no biased evaluation problem.

To further investigate the impact of $\mathcal{M}_{\text{val}}$ and optimization algorithms, we compare ERAS with following variants:

- ERAS$^{\text{los}}$ utilizes the validation loss $\mathcal{L}$ to replace $\mathcal{M}_{\text{val}}$. Other steps are same with ERAS.

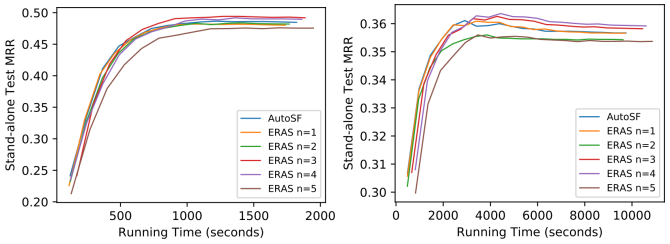


(a) Validation MRR as $\mathcal{M}_{\text{val}}$. (b) Validation loss as $\mathcal{M}_{\text{val}}$.

Fig. 5: The correlation between stand-alone validation MRR with different $\mathcal{M}_{\text{val}}$ settings of various searched scoring functions on WN18RR.

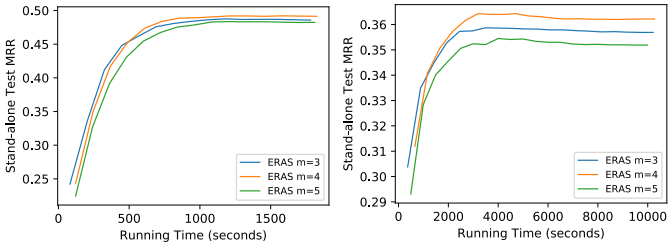
- ERAS$^{\text{dif}}$ first replaces $\mathcal{M}_{\text{val}}$ with $\mathcal{L}$ as ERAS$^{\text{los}}$ does. Then the differentiable measurement $\mathcal{L}$ enables the differentiable optimization algorithm [36], [37] for search. The detailed implementing ERAS$^{\text{dif}}$ is presented in Appendix.

We show the correlation between stand-alone validation MRR with different $\mathcal{M}_{\text{val}}$ settings in Figure 5. Obviously, compared with using MRR as $\mathcal{M}_{\text{val}}$ in Figure 5 (a), Figure 5 (b) shows that using $\mathcal{L}$ as $\mathcal{M}_{\text{val}}$ has a low correlation with stand-alone validation MRR. This indicates that validation loss in search strategy cannot well evaluate the stand-alone performance of scoring functions. Subsequently, in Table XI, we can observe that the performance of ERAS using MRR as $\mathcal{M}_{\text{val}}$ is better than that of two variants, i.e., ERAS$^{\text{los}}$ and ERAS$^{\text{dif}}$. Moreover, ERAS$^{\text{los}}$ is worse than ERAS$^{\text{dif}}$ because optimizing loss with the RL approach could not make use of the differentiable nature of $\mathcal{L}$.



(a) WN18RR. (b) FB15k237.

Fig. 6: Comparison on time (sec) of model training vs. testing MRR with different number of groups N in ERAS.



(a) WN18RR. (b) FB15k237.

Fig. 7: Comparison on time (sec) of model training vs. testing MRR with different number of blocks M in ERAS.

TABLE XI: Comparison of the variants of ERAS on the link prediction task.

Section	Variant	WN18	WN18RR	FB15K	FB15k237	YAGO3-10
		MRR	MRR	MRR	MRR	MRR
V-E1	ERAS$^{\text{los}}$	0.944	0.485	0.840	0.344	0.560
	ERAS$^{\text{dif}}$	0.949	0.485	0.848	0.355	0.565
V-E2	ERAS$^{\text{sig}}$	0.945	0.480	0.844	0.338	0.559
V-E3	ERAS$^{\text{pde}}$	0.950	0.489	0.850	0.349	0.570
	ERAS$^{\text{smt}}$	0.948	0.485	0.845	0.347	0.565
	ERAS	**0.953**	**0.492**	**0.855**	**0.365**	**0.577**

2) Impact of Optimization Level: In this paper, Definition 2 formulates the problem with a bi-level optimization objective. As stated in Section III-B, bi-level optimization can benefit ERAS by updating the scoring functions and embeddings separately. To investigate the impact of optimization level, we add another variant of ERAS as ERAS$^{\text{sig}}$, which utilizes the training set to update (3) in Definition 2 (i.e., single-level problem). In Table XI, compared with ERAS$^{\text{sig}}$, ERAS demonstrates the bi-level optimization is needed because optimizing scoring functions on the validation set encourages ERAS to select scoring functions that generalize well rather than scoring functions that overfit on the training data.

3) Impact of Grouping Approaches: To explore more about the influence of different grouping approaches, we set two variants of ERAS as follows:

- ERAS$^{\text{pde}}$ does not update $\boldsymbol{B}$ during search. Instead, it fixes groupings based on the embedding trained from SimplE.
- ERAS$^{\text{smt}}$ groups relations based on their semantic meanings (i.e., symmetric, anti-symmetric, asymmetric and inverse).

We compare these two variants with ERAS as shown in Table XI. Generally, the performance of ERAS$^{\text{smt}}$ is unsatisfactory due to the bias between human understanding of relation groups with the proper groups derived from data. In short, the performance comparison also indicates the importance of dynamically assigning relation groups in the search strategy, which encourages relations to be assigned to the appropriate scoring function.

4) Impact of Grouping Numbers: To investigate the impact of grouping numbers in ERAS, we summarize the performance of searched scoring functions by AutoSF and different settings of ERAS in Figure 6, i.e., ERAS^{N} ($N \in \{1, 2, \cdots, 5\}$) on WN18RR and FB15k237. Generally, the more groups there are, the longer the running time is. And ERAS achieves the best performance when $N = 3$ or 4. Comparing AutoSF and ERAS$^{N=1}$, they have a similar learning curve since their model complexities are the same.

5) Impact of Block Numbers: AutoSF fixes the block number M to 4 due to the efficiency issue. Once M is changed (e.g., $M = 3$ or $M = 5$), all design details in AutoSF must be re-done. On the contrary, the efficiency of ERAS allows more flexible settings of M. Here we try $M \in \{3, 4, 5\}$ in order to learn more about how the block number M influences the ERAS performance. As shown in Figure 7, we can observe that $M = 4$ does have excellent performance among $\{3, 4, 5\}$.

VI. Conclusion

In this paper, we propose a new automated machine learning (AutoML) method for designing scoring functions (scoring functions) in knowledge graph embedding. First, we design a relation-aware search space, which is motivated by our analysis of how existing scoring functions adapt to different relations. Then, we represent the new search space as a supernet in the form of a graph and propose to search through the supernet by one-shot architecture search methods. Experimental results on benchmark data sets well demonstrate not only the efficiency of our approach but also the competitive effectiveness.

For future works, one interesting direction to connect ERAS with graph neural network [51]; another direction worth to try is utilizing path instead of triplet to exploit higher-order information in KGs [52].

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

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

[3] D. Lukovnikov, A. Fischer, J. Lehmann, and S. Auer, "Neural network-based question answering over knowledge graphs on word and character level," in *WWW*, pp. 1211–1220, 2017.

[4] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *SIGKDD*, pp. 353–362, 2016.

[5] X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs," in *ICPR*, pp. 6857–6866, 2018.

[6] Y. Lin, X. Han, R. Xie, Z. Liu, and M. Sun, "Knowledge representation learning: A quantitative review," tech. rep., 2018.

[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, pp. 2787–2795, 2013.

[8] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014.

[9] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015.

[10] Y. Wang, R. Gemulla, and H. Li, "On multi-relational link prediction with bilinear models," in *AAAI*, 2018.

[11] R. Socher, D. Chen, C. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *NIPS*, 2013.

[12] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2D knowledge graph embeddings," in *AAAI*, 2018.

[13] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *SIGKDD*, 2014.

[14] Y. Zhang, Q. Yao, W. Dai, and L. Chen, "AutoSF: Searching scoring functions for knowledge graph embedding," in *ICDE*, pp. 433–444, 2020.

[15] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," in *ICLR*, 2015.

[16] T. Trouillon, C. R., É. Gaussier, J. Welbl, S. Riedel, and G. Bouchard, "Knowledge graph completion via complex tensor factorization," *JMLR*, vol. 18, no. 1, pp. 4735–4772, 2017.

[17] S. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," in *NeurIPS*, pp. 4284–4295, 2018.

[18] H. Liu, Y. Wu, and Y. Yang, "Analogical inference for multi-relational embeddings," in *ICML*, pp. 2168–2178, JMLR. org, 2017.

[19] T. Lacroix, N. Usunier, and G. Obozinski, "Canonical tensor decomposition for knowledge base completion," in *ICML*, pp. 2863–2872, 2018.

[20] I. Balazevic, C. Allen, and T. Hospedales, "Tucker: Tensor factorization for knowledge graph completion," in *EMNLP-IJCNLP*, pp. 5188–5197, 2019.

[21] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.

[22] Q. Yao and M. Wang, "Taking human out of learning applications: A survey on automated machine learning," tech. rep., arXiv:1810.13306, 2018.

[23] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *NIPS*, pp. 2962–2970, 2015.

[24] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2016.

[25] I. Balažević, C. Allen, and T. Hospedales, "Hypernetwork knowledge graph embeddings," in *ICANN*, pp. 553–565, Springer, 2019.

[26] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *AAAI*, 2016.

[27] A. Rossi, D. Firmani, A. Matinata, P. Merialdo, and D. Barbosa, "Knowledge graph embedding for link prediction: A comparative analysis," tech. rep., 2020.

[28] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla, and H. Stuckenschmidt, "Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion," in *ISWC*, pp. 3–20, 2018.

[29] Y. Xue, Y. Yuan, Z. Xu, and A. Sabharwal, "Expanding holographic embeddings for knowledge completion," in *NeurIPS*, pp. 4491–4501, 2018.

[30] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of Operations Research*, vol. 153, no. 1, pp. 235–256, 2007.

[31] Y. LeCun, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998.

[32] G. E. H. Rumelhart, David E. and R. J. Williams, "Learning representations by back-propagating errors," in *Nature*, 1986.

[33] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," 2013.

[34] L. Xie and A. Yuille, "Genetic CNN," in *ICCV*, pp. 1388–1397, 2017.

[35] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, pp. 4092–4101, 2018.

[36] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *ICLR*, 2018.

[37] Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu, "Efficient neural architecture search via proximal iterations," in *AAAI*, 2020.

[38] G. Bender, P.-J. Kinderm, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *ICML*, pp. 549–558, 2018.

[39] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *ML*, vol. 8, no. 3-4, pp. 229–256, 1992.

[40] M. Nickel, V. Tresp, and H. Kriegel, "A three-way model for collective learning on multi-relational data.," in *ICML*, vol. 11, pp. 809–816, 2011.

[41] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Workshop on CVSMC*, pp. 57–66, 2015.

[42] A. Dempster, N. M. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," tech. rep., 2019.

[45] Z. Sun, Z. Deng, J. Nie, and J. Tang, "RotatE: Knowledge graph embedding by relational rotation in complex space," in *ICLR*, 2019.

[46] S. Zhang, Y. Tay, L. Yao, and Q. Liu, "Quaternion knowledge graph embeddings," in *NeurIPS*, pp. 2731–2741, 2019.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, pp. 8024–8035, 2019.

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *JMLR*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[49] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

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

[51] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," tech. rep., 2018.

[52] Y. Zhang, Q. Yao, and L. Chen, "Interstellar: Searching recurrent architecture forknowledge graph embedding," in *NeurIPS*, 2019.

APPENDIX

DETAILS OF IMPLEMENTING ERAS$^{\text{DIF}}$

We propose a supernet view of the scoring function search space as in (8). This supernet design allows us to employ differentiable OAS methods when we use the loss $\mathcal{L}$ as $\mathcal{M}_{\text{val}}$. Following NASP [37], ERAS$^{\text{dif}}$ can update the architecture weight $\boldsymbol{A}$ by gradient descent as:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} - \epsilon \sum\nolimits_{n} \sum\nolimits_{(h,r,t)\in S_{\text{val}}} \nabla_{\boldsymbol{A}} B_{rn} \cdot \ell(f_n(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t})).$$

Then (3) can be optimized by above equation. Other steps of ERAS$^{\text{dif}}$ are same with ERAS.