AutoSF: Searching Scoring Functions for Knowledge Graph Embedding

Yongqi Zhang[†], Quanming Yao[‡], Wenyuan Dai[‡] and Lei Chen[†]
[†]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[‡]4Paradigm Inc., Beijing, China
[†]{yzhangee,leichen}@cse.ust.hk, [‡]{yaoquanming,daiwenyuan}@4paradigm.com

***Abstract*—Scoring functions (SFs), which measure the plausibility of triplets in knowledge graph (KG), have become the crux of KG embedding. Lots of SFs, which target at capturing different kinds of relations in KGs, have been designed by humans in recent years. However, as relations can exhibit complex patterns that are hard to infer before training, none of them can consistently perform better than others on existing benchmark data sets. In this paper, inspired by the recent success of automated machine learning (AutoML), we propose to automatically design SFs (AutoSF) for distinct KGs by the AutoML techniques. However, it is non-trivial to explore domain-specific information here to make AutoSF efficient and effective. We firstly identify a unified representation over popularly used SFs, which helps to set up a search space for AutoSF. Then, we propose a greedy algorithm to search in such a space efficiently. The algorithm is further sped up by a filter and a predictor, which can avoid repeatedly training SFs with same expressive ability and help removing bad candidates during the search before model training. Finally, we perform extensive experiments on benchmark data sets. Results on link prediction and triplets classification show that the searched SFs by AutoSF, are KG dependent, new to the literature, and outperform the state-of-the-art SFs designed by humans.** [1]

I. Introduction

Knowledge Graph (KG) [28], [33], [40], as a special kind of graph structure with entities as nodes and relations as edges, is important to both data mining and machine learning, and has inspired various downstream applications, e.g., structured search [7], [33], [36], question answering [25] and recommendation [52]. In KGs, each edge is represented as a triplet with form *(head entity, relation, tail entity)*, denoted as (h, r, t). A fundamental issue is how to quantize the plausibility of triplets (h, r, t)s [14], [40]. KG embedding (KGE) has recently emerged and been developed as a promising method serving this purpose [6], [18], [19], [24], [29], [30], [39], [46], [54]. Basically, given a set of observed triplets, KGE attempts to learn low-dimensional vector representations of entities and relations so that the plausibility of triplets can be quantized. Scoring function (SF), which returns a score for (h, r, t) based on the embeddings, is used to measure the plausibility. Generally, SF is designed and chosen by humans and it has significant effects on embeddings' quality [20], [28], [40].

Ever since the invention of KGE, many SFs have been proposed in the literature. Let the embedded vectors of h, r and t be $\boldsymbol{h}$, $\boldsymbol{r}$ and $\boldsymbol{t}$ respectively. TransE [4], a representative embedding model, interprets the triplet (h, r, t) as a translation r from head entity h to tail entity t, i.e. the embeddings satisfy $\boldsymbol{h} + \boldsymbol{r} = \boldsymbol{t}$. Variants of TransE like TransH [43] and TransR [21], project the embedding vector into different spaces and enables the embedding to model relationships that are one-to-many, many-to-one or many-to-many. These models are categorized into translational distance models (TDMs). However, as proved in [41], [42], TDMs are not fully expressive and their empirical performance is inferior to other models. RESCAL [30], DistMult [46], ComplEx [39], Analogy [24] and more recently proposed SimplE [18], [19], use a bilinear function $\boldsymbol{h}^\top \boldsymbol{R}\ \boldsymbol{t}$ to model the plausibility of triplets, where $\boldsymbol{R}$ is a square matrix related to relation embedding. These models belong to the bilinear model (BLMs). Different BLMs use different constrains to regularize the relation matrix $\boldsymbol{R}$ in order to adapt to different datasets. Inspired by the success of deep networks [2], some neural network models (NNMs) have also been explored as SFs, e.g., MLP [7], NTM [34], Neural LP [47] and ConvE [6]. Even though neural networks are powerful and have strong expressive ability, the NNMs do not perform well in KGE domain because of not being well-regularized.

Among the existing SFs, BLM-based ones are the most powerful as indicated by both the state-of-the-art results [19] and theoretical guarantees on expressiveness [18], [41]. However, since different KGs have distinct patterns in relations [32], a SF which adapts well to one KG may not perform consistently well on other KGs. Besides, designing new SFs to outperform state-of-the-art SFs is challenging. Therefore, how to choose and design a good SF for a certain KG is a non-trivial and difficult problem.

Recently, automated machine learning (AutoML) [17], [50] has exhibited its power in many machine learning tasks and applications, e.g. model selection [13], image classification [23], [51] and recommendation [48]. In order to select proper models and hyper-parameters for different tasks, hyperparameter optimization (HPO) has been proposed [11], [13] to effectively and efficiently find better configurations, which previously requires great human efforts. Another hot trend in AutoML is to search better neural networks for deep learning models. Neural architecture search (NAS) [55] has identified networks with fewer parameters and better performance than networks designed by humans.

Inspired by the success of AutoML, we aim to design

[1]This work is done when Y. Zhang is an intern in 4Paradigm, and the correspondence is to Q. Yao.

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00044

better and novel data-dependent SFs for KGE. Specifically, we propose automated scoring function (AutoSF) which can automatically search an SF for a given KG. It can not only reduce human's effort in designing new SFs, but also make adaptation to different KGs. However, it is not easy to achieve the above goal. When applying AutoML, two important perspectives, i.e. search space, which helps to figure out important properties of the underling problems, and search algorithm, which determines the efficiency of finding better points in the space, need serious consideration. In this work, we have made the following contributions to achieve the goals:

- First, we make an important observation over existing SFs, which allows us to represent the BLM-based SFs in a unified form. Based on the unified representation, we formulate the design of SFs as an AutoML problem (i.e. AutoSF), and set up the corresponding search space. The space is not only specific enough to cover good SFs designed by humans, but also general enough to include novel SFs not visited in the literature.
- Second, we observe it is common that different KGs have distinct properties on relations that are symmetric, asymmetric, inverse, etc. This inspires us to conduct domain-specific analysis on the KGE models, and design constraints to effectively guide subsequent searches in the space.
- Third, we propose a progressive greedy algorithm to search through such a space. We further build a filter to avoid training redundant SFs and a predictor with specifically designed symmetry-related features (SRF) to select promising SFs. The search algorithm can significantly reduce the search space size by capturing the domain specific properties of candidate SFs.
- Finally, experimental results on five popular benchmarks on link prediction and triplet classification tasks demonstrate that the SFs searched by AutoSF outperform the start-of-the-art SFs designed by humans. In addition, the searched SFs are KG dependent and new to the literature. We further conduct case study on the searched SFs to provide means for analyzing KGs, which can inspire better understanding of embedding techniques for future researches.

Notations. We denote vectors by lowercase boldface, and matrix by uppercase boldface. A KG contains a set of triplets $\mathcal{S} = \{(h, r, t)\}$ with $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$, where $\mathcal{E}$ and $\mathcal{R}$ are the set of entities and relations, respectively. For simplicity, the embeddings are represented by letters of indices in boldface, e.g. $\boldsymbol{h}$, $\boldsymbol{r}$, $\boldsymbol{t}$ are embeddings of h, r, t, respectively, and $\boldsymbol{h}, \boldsymbol{t}$ share the same set of embedding parameters $\boldsymbol{e}$. $\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \rangle = \sum_{i=1}^{d} \boldsymbol{a}_i \cdot \boldsymbol{b}_i \cdot \boldsymbol{c}_i$ is the triple dot product and can be alternatively represented as $\boldsymbol{a}^\top \text{diag}(\boldsymbol{b})\, \boldsymbol{c}$, where $\text{diag}(\boldsymbol{b}) = \mathbf{D}^{\boldsymbol{b}} \in \mathbb{R}^{d \times d}$ is the diagonal matrix of $\boldsymbol{b}$. We denote the complex vector $\boldsymbol{v} = \boldsymbol{v}_{re} + i\boldsymbol{v}_{im} \in \mathbb{C}^d$ with $\boldsymbol{v}_{re}, \boldsymbol{v}_{im} \in \mathbb{R}^d$. The conjugate of a complex vector is $conj(\boldsymbol{v}) = \boldsymbol{v}_{re} - i\boldsymbol{v}_{im}$.

II. Related Works

A. Knowledge graph embedding (KGE)

Given a set of observed (positive) triplets, the goal of KGE is to learn low-dimensional vector representations of entities and relations so that the plausibility measured by $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$ of observed triplets (h, r, t)s are maximized while those of non-observed ones are minimized [40]. To build a KGE model, the most important thing is to design and choose a proper SF f, which measures the triplets' plausibility based on embeddings. Since different SFs have different strengths and weaknesses, the choice of f is critical for the KGE's performance [20], [40]. A large amount of KGE models with popular SFs follow the same framework (Alg.1) [40] using stochastic gradient descent. At step 5, negative triplets are sampled from $\tilde{\mathcal{S}}_{(h,r,t)}$, which contains all non-observed triplets for a current positive triplet (h, r, t), by some fixed distributions [43] or dynamic sampling schemes [54]. Next, the gradients are computed based on the given SF and embeddings, and are used to update the model parameters (step 6). Hinge loss [4] and logistic loss [46] are popularly used as ℓ. In this paper, we use the multi-class loss [19] since it currently achieves the best performance and has little variance.

Algorithm 1 Stochastic training of KGE [40].

Input: training set $\mathcal{S} = \{(h, r, t)\}$, scoring function f and loss function ℓ;
1: initialize embeddings $\boldsymbol{e}, \boldsymbol{r}$ for each $e \in \mathcal{E}$ and $r \in \mathcal{R}$.
2: **for** $i = 1, \cdots, T$ **do**
3: sample a mini-batch $\mathcal{S}_{\text{batch}} \subseteq \mathcal{S}$ of size m;
4: **for** each $(h, r, t) \in \mathcal{S}_{\text{batch}}$ **do**
5: sample $\tilde{m}$ negative triplets $\tilde{\mathcal{S}}_{(h,r,t)} \equiv \{(\tilde{h}_j, r, \tilde{t}_j)\}$ for the positive triplet (h, r, t);
6: update embedding parameters based on loss ℓ using selected positive and negative triplets;
7: **end for**
8: **end for**
9: **return** embeddings of entities in $\mathcal{E}$ and relations in $\mathcal{R}$.

Existing human-designed SFs mainly fall into three types:

- *Translational distance models (TDMs):* The translational approach exploits the distance-based SFs. Inspired by the word analogy results in word embeddings [2], the plausibility is measured based on the distance between two entities, after a translation carried out by the relation. In TransE [4], the SF is defined by the (negative) distance between $\boldsymbol{h} + \boldsymbol{r}$ and $\boldsymbol{t}$, i.e. $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = -||\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}||_1$. Other TDMs-based SFs, e.g., TransH [43], TransR [12], enhance over TransE by introducing extra mapping matrices.
- *BiLinear models (BLMs):* SFs in this group exploit the plausibility of a triplet by the product-based similarity. Generally, they share the form as $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \boldsymbol{h}^\top \boldsymbol{R}\boldsymbol{t}$ where $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ is a matrix referring to the embedding of relation r [40], [41]. RESCAL [30] models the embedding of each relation by directly using $\boldsymbol{R}$. DistMult [46] overcomes the overfitting problem of RESCAL by constraining $\boldsymbol{R}$ to be diagonal. ComplEx [39] allows $\boldsymbol{R}$ and $\boldsymbol{h}, \boldsymbol{t}$ to be complex values, which enables handling asymmetric relations. HolE

TABLE I
EXISTING SFS COVERED BY OUR SEARCH SPACE. FOR ANALOGY AND SIMPLE, THE EMBEDDING SPLITS INTO TWO PARTS, I.E. $\boldsymbol{h}^\top = [\hat{\boldsymbol{h}}^\top, \check{\boldsymbol{h}}^\top]$ AND $d = \hat{d} + \check{d}$ (SAME FOR $\boldsymbol{r}$ AND $\boldsymbol{t}$). THE RELATION TYPES ARE SUMMARIZED IN TAB. II.

scoring function	embeddings	definition	relation types that can model
DistMult [46]	symmetric $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^d$	$\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle$	symmetric
ComplEx [39] / HolE [29]	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{C}^d$	$\text{Re}(\langle \boldsymbol{h}, \boldsymbol{r}, \text{conj}(\boldsymbol{t}) \rangle)$	symmetric, anti-symmetric, asymmetric, inverse
Analogy [24]	$\hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \in \mathbb{R}^{\hat{d}}$, $\check{\boldsymbol{h}}, \check{\boldsymbol{r}}, \check{\boldsymbol{t}} \in \mathbb{C}^{\check{d}}$	$\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle + \text{Re}\big(\langle \check{\boldsymbol{h}}, \check{\boldsymbol{r}}, \text{conj}(\check{\boldsymbol{t}}) \rangle\big)$	symmetric, anti-symmetric, asymmetric, inverse
SimplE [18] / CP [19]	$\hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \in \mathbb{R}^d$, $\check{\boldsymbol{h}}, \check{\boldsymbol{r}}, \check{\boldsymbol{t}} \in \mathbb{R}^d$	$\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \check{\boldsymbol{t}} \rangle + \langle \check{\boldsymbol{h}}, \check{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle$	symmetric, anti-symmetric, asymmetric, inverse

[29] uses a circular correlation to replace the dot product operation, but is proven to be equivalent to ComplEx [16]. Other variants like Analogy [24], SimplE [18] regularize the matrix $\boldsymbol{R}$ in different ways.

- *Neural network models (NNMs):* Neural models aim to output the probability of the triplets based on neural networks which take the entities' and relations' embeddings as inputs. MLP proposed in [7] and NTN proposed in [34] are representative neural models. Both of them use a large amount of parameters to combine entities' and relations' embeddings. ConvE [6] takes advantage of convolutional neural network to increase the interaction among different dimensions of the embeddings.

As proved in [41], TDMs have less expressive ability than BLMs, which further leads to their inferior empirical performance. Based on the power of deep networks, NNMs are also introduced for KGE. However, due to the huge model complexity and increasing difficulty of training, as well as the lack of domain-specific constraints, their performance is still worse than BLMs [6], [19]. Therefore, we focus on BLMs in the sequel. The most representative BLMs are listed in Tab. I.

B. Automated machine learning (AutoML)

Automated machine learning (AutoML) [17], [50] has recently exhibited its power in easing the usage of and designing better machine learning models. Basically, AutoML can be regarded as a bi-level optimization problem where we need to update model parameters by the training data sets and tune hyper-parameters by the validation data sets. Regarding the success of AutoML, there are two important perspectives:

- *Search space:* This helps to figure out important properties of the underlying learning models and set up the search space for an AutoML problem. First, the space needs to be general enough to cover human wisdom as special cases. However, the space cannot be too general, otherwise searching in the space will be too expensive.
- *Search algorithm*: Unlike convex optimization, there is no universal and efficient optimization tools. Once the search space is determined, efficient algorithms should be developed to search good points in the space.

We take NAS and HPO as examples. The search space in NAS is spanned by network operations, e.g., convolution with different sizes, skip-connections. Various tailor-made algorithms, such as reinforcement learning [55], evolution algorithms [44], and one-shot algorithms [23], [51], have been proposed for efficient optimization. For HPO, Bayesian optimization [11], [13] is usually customized to search the space made up by the hyper-parameters of the learning tools.

This paper is the first step towards automated embedding of knowledge graphs. However, such a step is not trivial since previous AutoML methods used in NAS and HPO cannot be directly applied to KGE. The main problem is that we need to explore domain-specific properties in defining the search space and designing efficient search algorithm to achieve effectiveness with less cost.

III. THE SEARCH PROBLEM

As mentioned in Sec.II, new designs of SFs have continuously boosted the performance of KGEs in recent years. However, there is no absolute winner among the human-designed SFs. Besides, as different KGs usually exhibit distinct patterns in relations, how to choose a proper SF to achieve good performance is non-trivial. These raise one question: *can we automatically design a SF for a given KG with good performance guarantee*? In this part, we define AutoSF as a searching problem and make deep analysis on the search space based on KG properties to address the question.

A. AutoSF: Searching for SFs

Since SF is the crux to KGE and different KG has distinct properties, we are motivated to form the problem of designing new and better SFs as a searching problem. Specifically, we define it as follows:

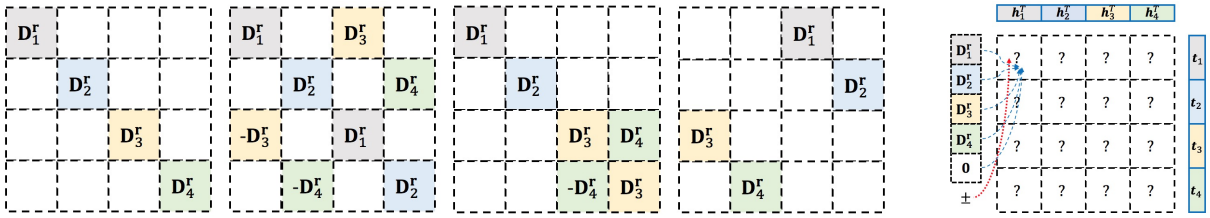
Definition 1 (AutoML Problem)**.** *Let $F(\boldsymbol{P}; g)$ be a KGE model (with indexed embeddings $\boldsymbol{P} = \{\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}\}$ and structure g), $\mathcal{M}\left(F(\boldsymbol{P}; g), \mathcal{S}\right)$ measures the performance (the higher the better) of a KGE model F on a set of triplets $\mathcal{S}$. The problem of searching the SF is formulated as:*

$$g^* \in \arg\max_{g \in \mathcal{G}} \mathcal{M}\left(F(\boldsymbol{P}^*; g), \mathcal{S}_{val}\right) \tag{1}$$

$$s.t. \quad \boldsymbol{P}^* = \arg\max_{\boldsymbol{P}} \mathcal{M}(F(\boldsymbol{P}; g), \mathcal{S}_{tra}), \tag{2}$$

where $\mathcal{G}$ contains all possible choices of g, $\mathcal{S}_{tra}$ and $\mathcal{S}_{val}$ denote training and validation data sets.

Same as NAS [44], [55] and HPO [11], [13], AutoSF is formulated as a bi-level optimization problem. We firstly need to train the model to obtain $\boldsymbol{P}^*$ (converged model parameters) on the training set $\mathcal{S}_{\text{tra}}$ by (2), and then search for a better g which is measured by the performance $\mathcal{M}$ on the validation set $\mathcal{S}_{\text{val}}$ by (1). However, in this sequel we can see the search space of g and search strategy in AutoSF are fundamentally different from previous AutoML works. They are closely related to KGE's domain and new to the AutoML literature.



(a) DistMult. (b) ComplEx. (c) Analogy. (d) SimplE. (e) AutoSF.

Fig. 1. A graphical illustration of $\boldsymbol{R}$ for existing SFs in Tab. I and the search space of AutoSF. Blank space is for zero matrix and $\boldsymbol{D}_i^r = \text{diag}(\boldsymbol{r}_i)\,, i = 1\dots 4$.

B. Search space: a unified representation of BLM SFs

To solve the AutoSF problem, the first question is: *what is a good search space* $\mathcal{G}$? As discussed in Sec.II-B, the space can neither be too specific nor too general. To motivate a good search space, let us look at some commonly used SFs (Tab. I) and dig out what are important properties of g.

As discussed in Sec.II-A, the state-of-the-art performance of KGE models is achieved by BLMs [18], [19], thus we limit our scope to them. RESCAL [30] is not considered since it does not have good scalability [24], [39] and neither empirically perform well. The other models in Tab. I regularize the number of trainable parameters of square matrix $\boldsymbol{R} \in \mathbb{R}^{d\times d}$ to be the same as entity embedding dimensions. Therefore, we constrain the relation embedding size to be the same as the entity's and learn different ways of mapping the relation embedding $\boldsymbol{r} \in \mathbb{R}^d$ into a square matrix $\boldsymbol{R} \in \mathbb{R}^{d\times d}$. Besides, as the summary of relation types in Tab. I, important properties are symmetric, anti-symmetric, asymmetric and inverse. These are important properties of good SFs. Thus, a search space should be able to handle the symmetric related properties. In addition, as will be discussed in Remark III.1, different SFs in BLMs differ in their way of regularizing the square matrix $\boldsymbol{R}$. Therefore, we are motivated to adaptively search how to regularize the relational matrix on different KGs.

To motivate such a space, we can see that there are two main differences among these SFs.

- The embedding can be either real or complex, e.g. DistMult v.s. ComplEx.
- When embedding vectors are split, different SFs combine them in distinct manners, e.g., Analogy v.s. SimplE.

1) Dealing with complex embeddings: A complex vector $\boldsymbol{v}\in\mathbb{C}^d$ with $\boldsymbol{v} = \boldsymbol{v}_{re}+i\boldsymbol{v}_{im}$ is composed of a real part $\boldsymbol{v}_{re}\in\mathbb{R}^d$ and an imaginary part $\boldsymbol{v}_{im} \in \mathbb{R}^d$. To deal with the complex embeddings, we can use $2d$-dimensional real vector $[\boldsymbol{v}_{re},\boldsymbol{v}_{im}]$ to represent the d-dimensional complex vector $\boldsymbol{v}$ [1], [39]. Let the complex embedding $\boldsymbol{h}=\boldsymbol{h}_{re}+i\boldsymbol{h}_{im}$, where $\boldsymbol{h}_{re}, \boldsymbol{h}_{im} \in \mathbb{R}^d$ (same for $\boldsymbol{r}, \boldsymbol{t}$), then ComplEx can be expressed as

$$\begin{aligned}\text{Re}\left(\langle \boldsymbol{h}, \boldsymbol{r}, \text{conj}(\boldsymbol{t})\rangle\right) = \langle \boldsymbol{h}_{re}, \boldsymbol{r}_{re}, \boldsymbol{t}_{re}\rangle + \langle \boldsymbol{h}_{im}, \boldsymbol{r}_{re}, \boldsymbol{t}_{im}\rangle \\ + \langle \boldsymbol{h}_{re}, \boldsymbol{r}_{im}, \boldsymbol{t}_{im}\rangle - \langle \boldsymbol{h}_{im}, \boldsymbol{r}_{im}, \boldsymbol{t}_{re}\rangle\,. \quad (3)\end{aligned}$$

Similarly, DistMult [46] with $2d$-dimensional embeddings can also be denoted by $[\boldsymbol{v}_{re}, \boldsymbol{v}_{im}]$ and represented as two parts

$$\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}\rangle = \langle \boldsymbol{h}_{re}, \boldsymbol{r}_{re}, \boldsymbol{t}_{re}\rangle + \langle \boldsymbol{h}_{im}, \boldsymbol{r}_{im}, \boldsymbol{t}_{im}\rangle\,. \quad (4)$$

2) Dealing with different splits: To make the training parameters consistent, we also use $2d$-dimensional real valued embeddings to represent Analogy and SimplE. As given in Tab. I, embeddings in Analogy [24] are split into a real part $\hat{\boldsymbol{h}} \in \mathbb{R}^d$, and a complex part $\check{\boldsymbol{h}}$, which can be denoted as a concatenated real vector $[\check{\boldsymbol{h}}_{re}, \check{\boldsymbol{h}}_{im}] \in \mathbb{R}^d$ in the similar way as ComplEx. And the SF is split as

$$\left\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}}\right\rangle + \text{Re}\left(\left\langle \check{\boldsymbol{h}}, \check{\boldsymbol{r}}, \text{conj}(\check{\boldsymbol{t}})\right\rangle\right). \quad (5)$$

In SimplE [18], two independent embedding vectors $\hat{\boldsymbol{h}} \in \mathbb{R}^d$ and $\check{\boldsymbol{h}} \in \mathbb{R}^d$ are used to represent each entity and relation. The resulting SF becomes

$$\left\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \check{\boldsymbol{t}}\right\rangle + \left\langle \check{\boldsymbol{h}}, \check{\boldsymbol{r}}, \hat{\boldsymbol{t}}\right\rangle. \quad (6)$$

3) The unified representation: In order to deal with the two different partitions, i.e. ComplEx v.s. DistMult and Analogy v.s. SimplE, we split embedding $\boldsymbol{h} \in \mathbb{R}^d$ as $\boldsymbol{h} = [\boldsymbol{h}_1; \boldsymbol{h}_2; \boldsymbol{h}_3; \boldsymbol{h}_4]$ (same for $\boldsymbol{r}$ and $\boldsymbol{t}$) to cover (3), (4), (5) and (6). Note that any splits k (with $k \geq 4$ and k is even) can be used to cover the SFs in Tab. I. We take $k = 4$ in order to ensure a tractable search space. The transformation of each SF is then summarized as:

$$\begin{aligned}\text{DistMult: } f(\mathbf{h},\mathbf{r},\mathbf{t}) &= \langle \mathbf{h}_1,\mathbf{r}_1,\mathbf{t}_1\rangle + \langle \mathbf{h}_2,\mathbf{r}_2,\mathbf{t}_2\rangle + \langle \mathbf{h}_3,\mathbf{r}_3,\mathbf{t}_3\rangle + \langle \mathbf{h}_4,\mathbf{r}_4,\mathbf{t}_4\rangle,\\ \text{ComplEx: } f(\mathbf{h},\mathbf{r},\mathbf{t}) &= \langle \mathbf{h}_1,\mathbf{r}_1,\mathbf{t}_1\rangle + \langle \mathbf{h}_1,\mathbf{r}_3,\mathbf{t}_3\rangle + \langle \mathbf{h}_3,\mathbf{r}_1,\mathbf{t}_3\rangle - \langle \mathbf{h}_3,\mathbf{r}_3,\mathbf{t}_1\rangle\\ &\quad + \langle \mathbf{h}_2,\mathbf{r}_2,\mathbf{t}_2\rangle + \langle \mathbf{h}_2,\mathbf{r}_4,\mathbf{t}_4\rangle + \langle \mathbf{h}_4,\mathbf{r}_2,\mathbf{t}_4\rangle - \langle \mathbf{h}_4,\mathbf{r}_4,\mathbf{t}_2\rangle,\\ \text{Analogy: } f(\mathbf{h},\mathbf{r},\mathbf{t}) &= \langle \mathbf{h}_1,\mathbf{r}_1,\mathbf{t}_1\rangle + \langle \mathbf{h}_2,\mathbf{r}_2,\mathbf{t}_2\rangle + \langle \mathbf{h}_3,\mathbf{r}_3,\mathbf{t}_3\rangle + \langle \mathbf{h}_3,\mathbf{r}_4,\mathbf{t}_4\rangle\\ &\quad + \langle \mathbf{h}_4,\mathbf{r}_3,\mathbf{t}_4\rangle - \langle \mathbf{h}_4,\mathbf{r}_4,\mathbf{t}_3\rangle,\\ \text{SimplE: } f(\mathbf{h},\mathbf{r},\mathbf{t}) &= \langle \mathbf{h}_1,\mathbf{r}_1,\mathbf{t}_3\rangle + \langle \mathbf{h}_2,\mathbf{r}_2,\mathbf{t}_4\rangle + \langle \mathbf{h}_3,\mathbf{r}_3,\mathbf{t}_1\rangle + \langle \mathbf{h}_4,\mathbf{r}_4,\mathbf{t}_2\rangle.\end{aligned}$$

Based on above formulations, all the scoring functions can be formed as $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \boldsymbol{h}^\top \boldsymbol{R}\boldsymbol{t}$. Let $\mathbf{D}_i^r = \text{diag}(\boldsymbol{r}_i)$ for $i \in \{1,2,3,4\}$, the forms of $\boldsymbol{R}$ for these SFs can be graphically represented as Fig. 1. In this way, we can see that the main difference between the four SFs is their way of filling the 4×4 block matrix (see Fig. 1(e)). Based on such a pattern, we identify the search space of BLM-based SFs in Definition 2.

Definition 2 (Search space $\mathcal{G}$)**.** *Let $g(\boldsymbol{r})$ return a 4×4 block matrix, of which the elements in each block is given by $[g(\boldsymbol{r})]_{ij} = diag(\boldsymbol{a}_{ij})$ where $\boldsymbol{a}_{ij} \in \{\boldsymbol{0}, \pm\boldsymbol{r}_1, \pm\boldsymbol{r}_2, \pm\boldsymbol{r}_3, \pm\boldsymbol{r}_4\}$ for $i, j \in \{1, 2, 3, 4\}$. Then, SFs can be represented by*

$$f_{unified}(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \sum\nolimits_{i,j} \langle \boldsymbol{h}_i, \mathbf{a}_{ij}, \boldsymbol{t}_j\rangle = \boldsymbol{h}^\top g(\boldsymbol{r})\,\boldsymbol{t}.$$

Remark III.1 (Searching to regularize BLMs)**.** *Note that the SFs shown in Fig. 5 constrain the relation matrix $\boldsymbol{R}$ in*

TABLE II
COMMON RELATIONS IN KGS AND RESULTING REQUIREMENTS ON f, AND SEARCH CANDIDATES IN $g(\boldsymbol{r})$.

common relations	requirements on f	requirements on $g(\boldsymbol{r})$	examples from WN18/FB15K
symmetric [46]	$f(\boldsymbol{t},\boldsymbol{r},\boldsymbol{h}) = f(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t})$	$g(\boldsymbol{r})^\top = g(\boldsymbol{r})$	*IsSimilarTo*, *Spouse*
anti-symmetric [29], [39]	$f(\boldsymbol{t},\boldsymbol{r},\boldsymbol{h}) = -f(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t})$	$g(\boldsymbol{r})^\top = -g(\boldsymbol{r})$	*LargerThan*, *Hypernym*
general asymmetric [24]	$f(\boldsymbol{t},\boldsymbol{r},\boldsymbol{h}) \neq f(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t})$	$g(\boldsymbol{r})^\top \neq g(\boldsymbol{r})$	*LocatedIn*, *Profession*
inverse [18]	$f(\boldsymbol{t},\boldsymbol{r},\boldsymbol{h}) = f(\boldsymbol{h},\boldsymbol{r}',\boldsymbol{t}), \mathbf{r} \neq \mathbf{r}'$	$g(\boldsymbol{r})^\top = g(\boldsymbol{r}')$	*Hypernym*, *Hyponym*

different forms, which can be regarded as different regularization schemes. Viewed in this way, AutoSF aims to search how to regularize the relational matrix that can adapt to different relation properties in different KGs. In addition, the data dependent regularization cannot be easily formed as a constraint in training procedure, which motivates us to use AutoML to search based on validation sets performance.

Remark III.2 (General Search Space)**.** *Due to the recent success of deep networks [15] and the approximation ability of multilayer perceptron (MLP) [5], one may want to use a MLP as $\mathcal{F}$ for* (2)*. However, the design of the MLP is also a searching problem, which is very time-consuming [55]. Besides, an arbitrarily large MLP will lead to an extremely large space. As verified in [49], the general approximator MLP is a bad choice for NAS and performs worse than those using reinforcement learning [55].*

IV. THE SEARCH STRATEGY

Here, we propose an efficient search strategy to address the AutoSF problem based on domain-specific properties in KGs.

A. Challenges in algorithm design

Same as the other AutoML problems, the search problem of AutoSF is black-box, the search space is huge, and each step in the search is very expensive since both model training and evaluation should be involved. These problems have previously been touched by algorithms such as reinforcement learning [55], Bayes optimization [13] and genetic programming [44]. However, they are not a good choice here since we have domain-specific problems, i.e. expressiveness and invariance, in KGE, which are more challenging.

1) Expressiveness: It is clear that not all SFs $g \in \mathcal{G}$ (from Definition 2) are equally good. Expressiveness (Definition 3), which means f should be better able to handle common relations in KGs , is of big concern for SFs. Their consequent requirements on f and $g(\boldsymbol{r})$ are summarized in Tab. II.

Definition 3 (Expressiveness [18], [39], [41])**.** *If f can handle symmetric, anti-symmetric, general asymmetric, and inverse relations, then f is expressive.*

To ensure that f can handle those common relations, we propose Proposition 1.

Proposition 1. *If $g(\boldsymbol{r})$ can be symmetric for some $\boldsymbol{r} \in \mathbb{R}^d$, i.e. $g(\boldsymbol{r})^\top = g(\boldsymbol{r})$, and skew-symmetric for some $\boldsymbol{r}' \in \mathbb{R}^d$, i.e. $g(\boldsymbol{r}')^\top = -g(\boldsymbol{r}')$. Then the formulated f is expressive. (Proofs in Appendix A).*

With this Proposition and to avoid trivial solutions, we introduce the following constraints on g:

(C1). $g(\boldsymbol{r})$ can be symmetric with proper $\boldsymbol{r}$ and skew-symmetric with proper $\boldsymbol{r}'$.

(C2). $g(\boldsymbol{r})$ has no zero rows/columns, covers all $\boldsymbol{r}_1$ to $\boldsymbol{r}_4$, and has no repeated rows/columns.

For (C1), the symmetric property of $g(\boldsymbol{r})$ determines what kind of relation the given SF can model based on Proposition 1. For (C2), if there are zero rows/columns in g, the corresponding embedding dimensions will be useless. It means that these dimensions will never be optimized during training.

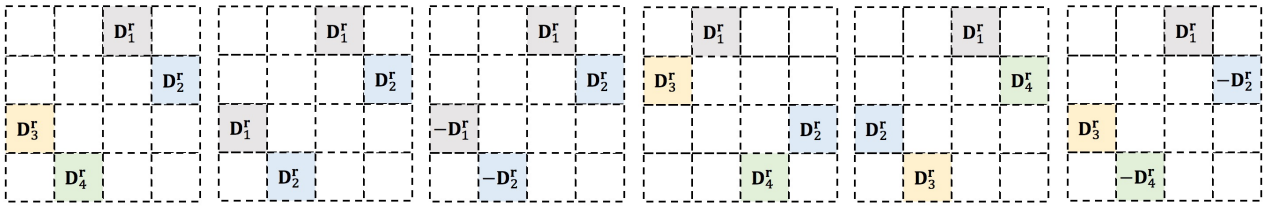
The above constraints are important for finding potentially good candidate $g \in \mathcal{G}$, and they play a key role in filtering out the bad g's for the design of an efficient search algorithm. As in Definition 3 and Proposition 1, we need to deal with Constraint (C1) for expressiveness. It is challenging since g only represents a structure, however the exact check of (C1) relies on the values in $\boldsymbol{r}$, which are unknown in advance. Fortunately, we can check it by value assignment. Take the SF in Fig. 2(a) for example. We can see that $g(\boldsymbol{r})$ can be symmetric by assigning $\boldsymbol{r}_3 = \boldsymbol{r}_1$ and $\boldsymbol{r}_4 = \boldsymbol{r}_2$ as in Fig. 2(b), and skew-symmetric by setting $\boldsymbol{r}_3 = -\boldsymbol{r}_2$ and $\boldsymbol{r}_4 = -\boldsymbol{r}_1$ like in Fig. 2(c). This is the key idea of addressing expressiveness.

2) Invariance: As defined in Sec.III-B, the embeddings are split into 4 parts, i.e. $\boldsymbol{r} = [\boldsymbol{r}_1; \boldsymbol{r}_2; \boldsymbol{r}_3; \boldsymbol{r}_4]$. Prior to the embedding training, permuting the $\boldsymbol{r}_i$'s will lead to equivalent structure since"1,2,3,4" here are only identity of each component and these components are equivalent at this stage. For example, we can permute $\boldsymbol{r} = [\boldsymbol{r}_1; \boldsymbol{r}_2; \boldsymbol{r}_3; \boldsymbol{r}_4]$ into $\boldsymbol{r} = [\boldsymbol{r}_2; \boldsymbol{r}_1; \boldsymbol{r}_3; \boldsymbol{r}_4]$. Even though $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ change their position, the learned embedding could be the same by changing corresponding values after training. Therefore, the structure of SFs is invariance to permutation of $\boldsymbol{r}_i$'s. Similarly, since $\boldsymbol{h}$ and $\boldsymbol{t}$ share the same embedding parameters $\boldsymbol{e}$, the generated SFs are also equivalent by simultaneously permuting the $\boldsymbol{h}_i$'s and $\boldsymbol{t}_i$'s. Moreover, if we flip the signs of some $\boldsymbol{r}_i$, we can learn equal embeddings by flipping the true value of those $\boldsymbol{r}_i$ after training. In summary, there exist three kinds of invariance: permuting entity embedding $\boldsymbol{h}_i$'s and $\boldsymbol{t}_i$'s, permuting relation embedding $\boldsymbol{r}_i$'s, and flipping signs. The example of three cases are given in Fig. 2(d-f). Let $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}$ be the embeddings of the SF g_1 and $\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}, \bar{\boldsymbol{t}}$ be the embedding of another SF g_2, and g_2 is formed through the invariance changes of g_1. Then we will have $\boldsymbol{h}^\top g_1(\boldsymbol{r})\boldsymbol{t} = \bar{\boldsymbol{h}}^\top g_2(\bar{\boldsymbol{r}})\bar{\boldsymbol{t}}$ after the model training. Therefore, it is tedious to train and evaluate the equivalents once we know the performance of one SF among them.

B. Progressive greedy search

As in Sec.III-B, adding one more block into g indicates adding one more nonzero multiplicative term into f, namely

$$f^{b+1} = f^b + s\,\langle \boldsymbol{h}_i, \boldsymbol{r}_j, \boldsymbol{t}_k \rangle\,, \tag{7}$$



(a) SimplE. (b) sym. (c) skew-sym. (d) permute $\boldsymbol{h}_i$ and $\boldsymbol{t}_i$. (e) permute $\boldsymbol{r}_i$. (f) flip sign.

Fig. 2. Illustration of *Invariance* and *Expressiveness*. (a) SimplE model; (b) assign $\boldsymbol{r}_3 = \boldsymbol{r}_1, \boldsymbol{r}_4 = \boldsymbol{r}_2$; (c) assign $\boldsymbol{r}_3 = -\boldsymbol{r}_1, \boldsymbol{r}_4 = -\boldsymbol{r}_2$; (d) permute $[\boldsymbol{h}_1; \boldsymbol{h}_2; \boldsymbol{h}_3; \boldsymbol{h}_4]$ into $[\boldsymbol{h}_1; \boldsymbol{h}_3; \boldsymbol{h}_2; \boldsymbol{h}_4]$ and do the same for $\boldsymbol{t}$; (e) permute $[\boldsymbol{r}_1; \boldsymbol{r}_2; \boldsymbol{r}_3; \boldsymbol{r}_4]$ into $[\boldsymbol{r}_1; \boldsymbol{r}_4; \boldsymbol{r}_2; \boldsymbol{r}_3]$; (f) flip the signs of $\boldsymbol{r}_2$ and $\boldsymbol{r}_4$.

where $s \in \{\pm 1\}$ and $i, j, k \in \{1, 2, 3, 4\}$. In order to search efficiently, we propose a progressive greedy algorithm based on the inductive rule (7), which can significantly cut down the search space in a stage-wise manner. The intuition of using (7) to progressively generate SFs it to gradually adjust the relation matrix $g(\boldsymbol{r})$. However, greedy search usually leads to sub-optimal solutions [38], which can be more serious when faced with the expressive and invariance challenges in AutoSF. Therefore, we enhance the greedy search with a filter and a predictor to specifically deal with the expressiveness and invariance discussed in Sec.IV-A.

Algorithm 2 Progressive greedy search algorithm.

Input: B: number of nonzero blocks in g, a learnable predictor $\mathcal{P}$;
1: **for** b in $4, 6, 8, \cdots, B$ **do**
2: **repeat**
3: randomly select a top-K_1 model $f^{b-2} \in \mathcal{T}^{b-2}$;
4: generate 6 integers $i_1, j_1, k_1, i_2, j_2, k_2 \in \{1, 2, 3, 4\}$ and s_1, s_2 from $\{\pm 1\}$, and form $f^b \leftarrow f^{b-2} + s_1 \langle \boldsymbol{h}_{i_1}, \boldsymbol{r}_{j_1}, \boldsymbol{t}_{k_1} \rangle + s_2 \langle \boldsymbol{h}_{i_2}, \boldsymbol{r}_{j_2}, \boldsymbol{t}_{k_2} \rangle$;
5: **if** f^b satisfies filter $\mathcal{Q}$ **then** $\mathcal{H}^b \leftarrow \mathcal{H}^b \cup \{f^b\}$;
6: **until** $|\mathcal{H}^b| = N$
7: select top-K_2 f^bs in $\mathcal{H}^b$ based on the *predictor* $\mathcal{P}$;
8: *train* embeddings with the selected f^bs;
9: *evaluate* the obtained embedding from the selected f^bs;
10: $\mathcal{T}^b \leftarrow$ add and record f^b and their performance;
11: update the predictor $\mathcal{P}$ with records in $\mathcal{T}$.
12: **end for**
13: **return** desired SFs in $\mathcal{T}^B$.

1) Complete procedures: Alg.2 shows our progressive greedy algorithm. As in Definition 2, let the number of nonzero blocks in g be B and the SF in this group be f^B. The idea of progressive search is that given the desired B, we start from small blocks b and then gradually add in more blocks until $b = B$. Thus, we can greedily generate candidates based on the top SFs in $\mathcal{T}^{b-2}$ at step 2-6 to reduce search space. Specifically, we greedily pick up the top-K_1 f^{b-2} in the previously evaluated models in $\mathcal{T}^{b-2}$. N candidates will then be generated by adding two more multiplicative term in step 4 to deal with Constraint (C1) since adding one block each step will result in simply lying on the diagonal. All the candidates are generated from $b = 4$ and are checked through the Filter $\mathcal{Q}$ (see Sec.IV-B2) to guarantee Constraint (C2) and avoid training equivalents. Next, we use the predictor $\mathcal{P}$ (see Sec.IV-B3) to further select K_2 promising candidates, which will then be trained and evaluated using Alg.1, in step 7 of Alg.2. The training data for $\mathcal{P}$ is gradually collected with the trained SFs in $\mathcal{T} = \mathcal{T}^4 \cup \mathcal{T}^6 \cup \cdots$ at step 10.

2) Invariance - Using a filter: The filter $\mathcal{Q}$ we used in Alg.2 has two functions: 1) deal with Constraint (C2) and 2) remove equivalent structures due to invariance. Constraint (C2) is easy to check, given the structure of g, we can directly map it into a 4×4 substitute matrix and use $\{0, \pm 1, \pm 2, \pm 2, \pm 4\}$ to represent $[g(\boldsymbol{r})]_{ij} \in \{\boldsymbol{0}, \pm \boldsymbol{r}_1, \pm \boldsymbol{r}_2, \pm \boldsymbol{r}_3, \pm \boldsymbol{r}_4\}$. Then checking requirements in (C2) is a trivial task, i.e. checking if the 4×4 substitute matrix satisfies the Constraint (C2).

For the invariance, once a candidate f^b fulfilling Constraint (C2) is generated, we use the invariance property to generate a set of equivalents $\mathcal{G}_{f^b}$. Specifically, we can permute the entity parts, relation parts, or flip signs to get $4! \times 4! \times 2^4 = 9,216$ equivalents of f^b. If $\mathcal{G}_{f^b} \cap \mathcal{H}^b \cap \mathcal{T}^b \neq \emptyset$, we throw f^b away since there are equivalent structures in the sampled set $\mathcal{H}^b$ and history record $\mathcal{T}^b$. This step can dramatically help us to reduce the cost in training equivalent structures. Take f^4 as an example, the whole space is reduced from $A_{16}^4 \times 2^4$ to 5 through the filter, namely there are only five good and unique candidates in f^4. Besides, we add exception for the condition in step 5 of Alg.2 for f^4 since the number of candidates is smaller than N.

3) Expressiveness - Constructing a predictor: Even though the filter helps to throw away many unpromising candidates, it does not deal with Constraint (C1). Hence, after collecting N candidates, we use the predictor $\mathcal{P}$ to further select K_2 promising ones among them. Considering that the performance of SFs on a specific KG is closely related to how the SF is formed, we can use a learning model, i.e. the predictor $\mathcal{P}$ to predict the performance and select good candidates in advance. In general, we need to extract features for points which have been visited by the search algorithm, and then use a learning model to predict validation performance based on those features [13], [22]. The following are principles a good predictor needs to meet

(P1). *Correlate well with true performance*: the predictor needs not to accurately predict the exact values of validation performance, instead it should rank good candidates over bad ones;

(P2). *Learn from small samples*: as the real performance of each point in the search space is expensive to acquire, the complexity of the predictor should be low so that it can learn from few samples.

Based on Principle (P1), the extracted features from g should be closely related to the quality of defined SF. Mean-

while, the features should be cheap to construct, i.e. they should not depend on values of $\boldsymbol{r}$, which are unknown before training. For (P2), the number of features should be small to guarantee a simple predictor. Therefore, we are motivated to design the symmetry-related features (SRFs), which can effectively capture to what extent $g(\boldsymbol{r})$ can be symmetric or skew-symmetric (Proposition 2), and has low complexity.

Similar as the filter, we also use a 4×4 substitute matrix to represent g. As in Fig. 3, we use $\boldsymbol{v} = [v_1; v_2; v_3; v_4]$ to represent $[\boldsymbol{r}_1; \boldsymbol{r}_2; \boldsymbol{r}_3; \boldsymbol{r}_4]$, then the symmetric and skew-symmetric property of g can be checked through $g(\boldsymbol{v}) - g(\boldsymbol{v})^\top$ and $g(\boldsymbol{v}) + g(\boldsymbol{v})^\top$. Since $g(\boldsymbol{v})$ is a simple 4×4 matrix, the checking procedure is very cheap. Then by assigning different values to $\boldsymbol{v}$ (details in Appendix C), a 22-dimensional SRF will be returned. Considering that the correlation of SRFs with SFs' performance is guaranteed under Proposition 2, we can use a simple two-layer MLP (22-2-1) as the predictor $\mathcal{P}$. Other regression models with low complexity may also work here.

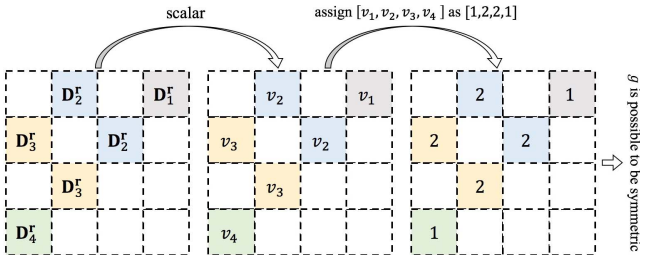


Fig. 3. Example of generating a feature of SRF.

Proposition 2. *The extracted SRFs (see Appendix C) are (i) invariant to both the permutations and flipping signs of blocks in $\boldsymbol{R}$ and (ii) give predictions related to symmetric or anti-symmetric properties.*

C. Search complexity analysis

There are 16 blocks and each can be filled with 9 different contents $\{\mathbf{0}, \pm\mathbf{D}_1^r, \pm\mathbf{D}_2^r, \pm\mathbf{D}_3^r, \pm\mathbf{D}_4^r\}$. Thus the whole space size is 9^{16}, which is extremely large. The greedy strategy, predictor and filter cut down the space in different perspectives. Specifically, in each greedy step:

- *Greedy*: Considering that f^b is progressively generated on f^{b-2} for $b = 6, 8, \ldots$, there can be $C_{16-(b-2)}^2 \times 4^2 \times 2^2$ candidates ($C_{16-(b-2)}^2$ is to choose location, 4^2 is for the two $\boldsymbol{r}_i$s and 2^2 is for signs). In comparison, there can be $C_{16}^b \times 4^b \times 2^b$ possible SFs in f^b. Take $b = 6$ for example, there are 2×10^9 possible candidates. Since f^6 is generated based on the 5 good candidates in f^4, we reduce the space size from 2×10^9 to approximately 3×10^4 based on the greedy scheme.
- *Filter*: The filter we designed is mainly used to deal with invariance properties. Permuting $\boldsymbol{r}_i$'s leads to $4! = 24$ equivalent structures. Simultaneously permuting $\boldsymbol{h}_i$'s and $\boldsymbol{t}_i$'s also gives 24 equivalents. Besides, there are $2^4 = 16$ possible signs patterns. Therefore, given a $g(\boldsymbol{r})$, we can generate at most (there may exist same structures in this set) $24 \times 24 \times 16 = 9216$ equivalent SFs, which should perform the same. Besides, by constraining the SF under Constraint (C2), many bad candidates can also be filtered out. Take f^4 as an example, only 5 candidates are selected to be trained among approximately $700k$ possible structures.
- *Predictor*: Once N candidates are generated, the predictor will select K_2 ones based on their predicted performance. Thus, the reducing ratio of predictor is about N/K_2.

While it is difficult to directly quantize to which extend the three steps together can help to reduce the search space, we can observe the significance of efficiency gained through each component. Besides, we perform an empirical study in Sec.V-E to show the performance gaining of these steps.

D. Comparison with existing AutoML approaches

The most related work in the AutoML literature is PNAS [22], which combines a greedy algorithm with a performance predictor to search a cell structure for the convolutional neural network (CNN). However, the filter is not used in PNAS as the search space for AutoSF is fundamentally different from that of CNN. Besides, PNAS adopts direct one-hot encoding for the predictor, which has a bad empirical performance here (see Sec.V-E1). As for the other AutoML approaches, even though the search problem of AutoSF is similarly defined as HPO [3], [9] and NAS [10], the search space and search algorithm of AutoSF are novel and specifically designed for KGE. There is no direct way for them to deal with the challenges in Sec.IV-A.

V. Empirical Study

All of the algorithms are written in python with PyTorch framework [31]. Experiments are run on 8 TITAN Xp GPUs.

A. Experiment setup

1) Datasets: Five data sets, i.e. WN18, FB15k, WN18RR, FB15k237 and YAGO3-10 are considered (statistics in Tab. III). WN18RR and FB15k237 are variants that remove near-duplicate or inverse-duplicate relations from WN18 and FB15k respectively, [37], [42]. YAGO3-10 is much larger than the others. These are benchmark datasets, and are popularly used to compare KGE models in the literature [4], [18], [19], [24], [39], [46]. The number of symmetric, anti-symmetric, inverse pairs and general asymmetric are computed in the following way: Given a relation r, let the number of positive triplets (h, r, t) be n_r. (i) If the number of (t, r, h) is larger than $0.9n_r$, then we regard it as symmetric; (ii) If the number of (t, r, h) is zero and the size of joint set of h and t is at least $0.1n_r$ (this is to ensure that they have same type), we regard as anti-symmetric; (iii) If there exist another relation r' that has at least $0.9n_r$ (t, r', h), then r and r' are inverse pairs; (iv) others are regarded as general asymmetric. The threshold 0.9 and 0.1 are hand-made and just used to roughly (other values are fine) indicate the relation properties for each data set.

2) Hyper-parameters: Since the searched embedding models belong to BLMs, we fairly compare different SFs with a fixed set of hyper-parameters. In order to reduce training time, we set the dimension d as 64 during the search procedure. First, we use SimplE [18] as the benchmark model and

TABLE III

STATISTICS OF THE DATA SETS USED IN EXPERIMENTS. "SYM" AND "ANTI-SYM" DENOTES THE SYMMETRIC AND ANTI-SYMMETRIC RELATIONS.

data set	#entity	#relation	#train	#valid	#test	#sym	#anti-sym	#inverse	#general
WN18 [4]	40,943	18	141,442	5,000	5,000	4	7	7	0
FB15k [4]	14,951	1,345	484,142	50,000	59,071	66	38	556	685
WN18RR [6]	40,943	11	86,835	3,034	3,134	4	3	1	3
FB15k237 [37]	14,541	237	272,115	17,535	20,466	33	5	20	179
YAGO3-10 [26]	123,188	37	1,079,040	5,000	5,000	8	0	1	28

TABLE IV

COMPARISON OF THE BEST SF IDENTIFIED BY AUTOSF AND THE STATE-OF-THE-ART SFS. THE BOLD NUMBER MEANS THE BEST PERFORMANCE, AND THE UNDERLINE MEANS THE SECOND BEST. DISTMULT, COMPLEX, ANALOGY AND SIMPLE ARE OBTAINED FROM OUR IMPLEMENTATION, OTHERS ARE COPIED FROM THE CORRESPONDING REFERENCE PAPER. STD IS LESS THAN 0.001, THUS NOT REPORTED.

		WN18			FB15k			WN18RR			FB15k237			YAGO3-10		
type	model	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10
TDM	TransE [54]	0.500	—	94.1	0.495	—	77.4	0.178	—	45.1	0.256	—	41.9	—	—	—
	TransH [54]	0.521	—	94.5	0.452	—	76.6	0.186	—	45.1	0.233	—	40.1	—	—	—
	RotatE [35]	0.949	94.4	95.9	0.797	74.6	88.4	0.476	42.8	**57.1**	0.338	24.1	53.3	—	—	—
NNM	NTN [46]	0.53	—	66.1	0.25		41.4	—	—	—	—	—	—	—	—	—
	Neural LP [47]	0.94	—	94.5	0.76	—	83.7	—	—	—	0.24	—	36.2	—	—	—
	ConvE [6]	0.942	93.5	95.5	0.745	67.0	87.3	0.46	39.	48.	0.316	23.9	49.1	0.52	45.	66.
BLM	TuckER [1]	**0.953**	**94.9**	95.8	0.795	74.1	89.2	0.470	44.3	52.6	0.358	26.6	54.4	—	—	—
	HolEX [45]	0.938	93.0	94.9	0.800	75.0	88.6	—	—	—	—	—	—	—	—	—
	QuatE [53]	0.950	94.5	95.9	0.782	71.1	90.0	0.488	43.8	**58.2**	0.348	24.8	55.0	—	—	—
	DistMult	0.821	71.7	95.2	0.817	77.7	89.5	0.443	40.4	50.7	0.349	25.7	53.7	0.552	47.6	69.4
	ComplEx	0.951	94.5	95.7	0.831	79.6	90.5	0.471	43.0	55.1	0.347	25.4	54.1	0.566	49.1	70.9
	Analogy	0.950	94.6	95.7	0.829	79.3	90.5	0.472	43.3	55.8	0.348	25.6	54.7	0.565	49.0	71.3
	SimplE/CP	0.950	94.5	95.9	0.830	79.8	90.3	0.468	42.9	55.2	0.350	26.0	54.4	0.565	49.1	71.0
AnyBURL [27]		0.95	94.6	95.9	0.83	80.8	87.6	0.48	44.6	55.5	0.31	23.3	48.6	0.54	47.7	47.3
AutoSF		0.952	94.7	**96.1**	**0.853**	**82.1**	**91.0**	**0.490**	**45.1**	56.7	**0.360**	**26.7**	**55.2**	**0.571**	**50.1**	**71.5**

tune hyper-parameters with the help of HyperOpt, a hyper-parameter optimization framework based on TPE [3]. The searching ranges are given as follows: learning rate η in $[0, 1]$, L2 penalty λ in $[10^{-5}, 10^{-1}]$, decay rate in $[0.99, 1.0]$, batch size m in $\{256, 512, 1024\}$. All the models are trained until converge to avoid the influence of different convergence speed. Besides, we use Adagrad [8] as the optimizer since it tends to perform better as indicated in [19], [39]. Once a good hyper-parameter configuration is selected, we use it to train and evaluate different searched SFs. After the search procedure, we pick up the best SF evaluated by the MRR performance on the validation data set as the searched SF. When comparing the searched SFs with human-designed ones, we increase the dimension from 64 to $d \in \{256, 512, 1024, 2048\}$ as in [19]. As mentioned in [42], KGE models are sensitive to hyper-parameters. For a fair comparison, we use the same set of hyper-parameters to train and evaluated different models on each dataset.

3) Meta Hyper-parameters: The hyper-parameters K_1, K_2 and N have little influence to the search procedure. We use $K_1 = K_2 = 8$ and $N = 256$ for all data sets. Besides, steps 2-11 in Alg.2 run based on an inner loop. We train 8 models in parallel and iterate for 32 times (16 times for YAGO3-10), namely we evaluate 256 f^bs for each $b > 4$.

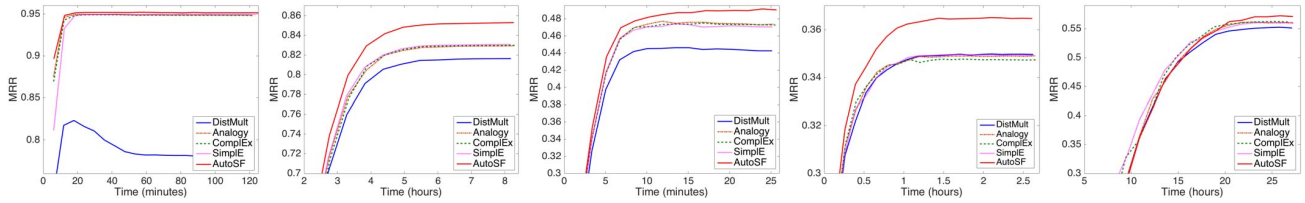
B. Comparison with existing SFs on link prediction

We compare our AutoSF with the state-of-the-art KGE models discussed in Sec.II-A, which are designed by humans, i.e. TransE [4], TransH [43], and RotatE [35] from TDMs; NTM [34], Neural LP [47], and ConvE [6] from NNMs; TuckER [1], HolE/HolEX [29], [45], Quat [53], DistMult [46], ComplEx [39], Analogy [24] and SimplE [18] from BLMs; and a rule-based method AnyBURL [27]. Hyper-parameters are selected by the MRR value on the validation set.

Following [6], [18], [24], [39], [46], we test KGE's performance based on *link prediction*. For each triplet $(h, r, t) \in \mathcal{S}$, where $\mathcal{S}$ is the validation or testing set, we compute the score of (h', r, t) for all $h' \in \mathcal{E}$ and get the rank of h, the same for t based on scores of (h, r, t') over all $t' \in \mathcal{E}$, r is not compared as in the literature [40]. Same as above mentioned papers, we adopt the following metrics: (i) Mean reciprocal ranking (MRR): $1/|\mathcal{S}| \sum_{i=1}^{|\mathcal{S}|} 1/\text{rank}_i$, where $\text{rank}_i, i \in \{1, \ldots, |\mathcal{S}|\}$ is a set of ranking results and (ii) H@10: $1/|\mathcal{S}| \sum_{i=1}^{|\mathcal{S}|} \mathbb{I}(\text{rank}_i < 10)$, where $\mathbb{I}(\cdot)$ is the indicator function. We report the performance in a "filtered" setting as in [4], [43], where larger MRR and H@10 indicate higher embedding quality.

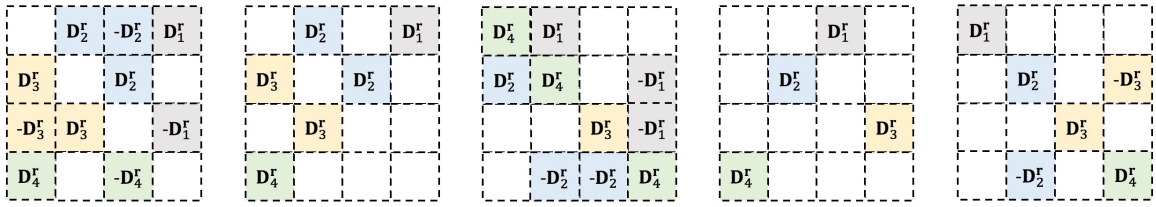
1) Effectiveness: A comparison of the testing performance of AutoSF and the current state-of-the-art SFs are shown in Tab. IV. Firstly, we can see that there is no absolute winner among the baseline SFs. For example, TuckER is the best on WN18, but is the worst among human-designed BLMs on FB15k. DistMult generally performs worse on the benchmarks except for FB15k237 since it does not follow Proposition 1. A single model is hard to adapt to different KGs. However, AutoSF performs consistently well among these five data sets. i.e. the best among FB15k, WN18RR, FB15k237 and YAGO3-10, and the runner-up on WN18.

Besides, we plot the learning curves of DistMult, Analogy, ComplEx, SimplE and the best SF searched by AutoSF in Fig. 4. As shown, the searched SFs not only outperform baselines, but also converge faster, which may due to these SFs can better capture relations in these datasets.



(a) WN18. (b) FB15k. (c) WN18RR. (d) FB15k237. (e) YAGO3-10.

Fig. 4. Comparison on clock time (in hours) of model training v.s testing MRR between search SFs (by AutoSF) and human-designed ones.



(a) WN18. (b) FB15k. (c) WN18RR. (d) FB15k237. (e) YAGO3-10.

Fig. 5. A graphical illustration of SFs identified by our AutoSF on each data set.

2) Case study: Distinctiveness: To show the searched SFs are KG-dependent and novel to the literature, we plot them in Fig. 5. It is obvious that these SFs are different from each other, and they are not equivalent regarding invariance properties. As shown in Tab. III, WN18 and FB15k have many symmetric, anti-symmetric relations and inverse relation pairs, the best SF searched on them are very similar and have the same SRF. The other three data sets are more realistic and contains less symmetric, anti-symmetric and inverse relations, thus have different SRFs with fewer entry being non-zero.

The most special case is FB15k237, which can only be symmetric under (S11). Viewing the values in Tab. IV, we can see that the leading performance on FB15k237 is achieved by DistMult and AutoSF, both of which cannot be skew-symmetric. As given in the statistic information in Tab. III, FB15k237 has relatively fewer anti-symmetric relations. This may explain why skew-symmetric is not that important for $g(\boldsymbol{r})$. However, SRFs still work for these cases since it can be aware that skew-symmetric property is not that essential and focus more on searching different local structures.

TABLE V
MRRs OF APPLYING SF SEARCHED FROM ONE DATA SET (INDICATED BY EACH ROW) ON ANOTHER DATA SET (INDICATED BY EACH COLUMN).

	WN18	FB15k	WN18RR	FB15k237	YAGO3-10
WN18	**0.952**	0.841	0.473	0.349	0.561
FB15k	0.950	**0.853**	0.470	0.350	0.563
WN18RR	0.951	0.833	**0.490**	0.345	0.568
FB15k237	0.894	0.781	0.462	**0.360**	0.565
YAGO3-10	0.885	0.835	0.466	0.352	**0.571**

Besides, we pick up the best SF searched from one data set and test it on another data set in Tab. V. We can readily find that these SFs get the best performance on the data sets where they are searched. This again demonstrate that SFs found by AutoSF on different KGs are distinct from each other.

C. Comparison with existing SFs on triplet classification

To further demonstrate the effectiveness of the searched SFs, we do triplet classification as in [43]. This task is to confirm whether a given (h, r, t) is correct or not and is more helpful in answering yes-or-no questions. The decision rule of classification is as follows: for each (h, r, t), if its score is larger than the relation-specific threshold σ_r, which we predict to be positive, otherwise negative. The threshold σ_r is determined by maximizing the accuracy of the validation set. We test this task on FB15k, WN18RR and FB15k237, in which the positive and negative triplets are provided. As shown in Tab. VI, searched SFs consistently outperform human-designed BLMs.

TABLE VI
COMPARISON OF SEARCHED SFS WITH THE STATE-OF-THE-ART SFS ON ACCURACY (IN %) FOR TRIPLET CLASSIFICATION. STD<0.2.

	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**

D. Comparison with other AutoML approaches

In this part, we compare AutoSF with the other search algorithms. WN18RR and FB15k237 are used here, and all algorithms share the same set of hyper-parameters. First, to show the effectiveness of the search space in BLM, we train a general approximator (*Gen-Approx*), i.e. MLP (in Appendix D), on the validation set. Then, AutoSF is compared with *Random* search and *Bayes* algorithm [3] on f^6. As shown in Fig. 6, the general approximator performs much worse than BLM since it is too flexible to consider domain-specific constraints and easily overfits. For BLM settings, the Bayes algorithm can improve the efficiency upon random search. However, it will easily fall into local optimum and does not take the domain property into account. Among them, AutoSF is the most efficient and has the best any-time performance.

E. Ablation study

We use WN18RR and FB15k237 to illustrate the importance of different components in the proposed searching algorithm.

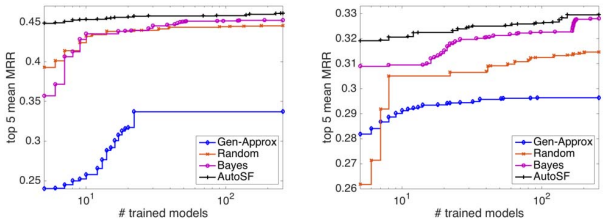


Fig. 6. Comparison of AutoSF with other AutoML approaches.

1) Filter and predictor: To show the effectiveness of the filter and predictor, we remove them from AutoSF and make comparisons in Fig. 7. As shown, the greedy algorithm is more efficient than random search. Both filter and predictor are important. Removing either the filter or predictor will lead to degenerated efficiency. Besides, compared with *Greedy*, i.e. no filter and no predictor, they can both improve efficiency through reducing the search space.

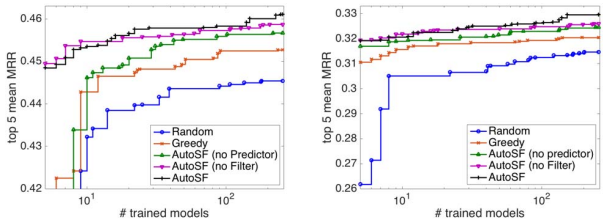


Fig. 7. Ablation study on the predictor and filter in AutoSF.

2) SRF features: As in Sec.V-D, one-hot representation [22] can also be used as an alternative to SRFs. We compare the two kind of features in Fig. 8. For AutoSF (with one-hot), a 96-8-1 fully connected neural network is used, and a 22-2-1 network is used for AutoSF (with SRF). AutoSF (no predictor) is shown here as a baseline and it is the same as that in Fig. 7.

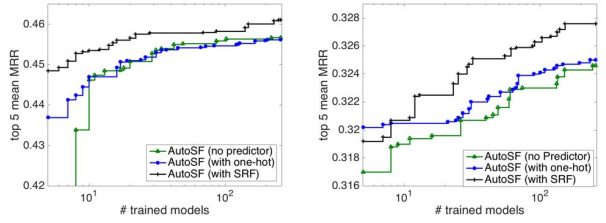


Fig. 8. Comparison of the proposed SRF with one-hot encoding in [22].

3) Sensitivity of meta hyper-parameters: There are three meta hyper-parameters N, K_1, K_2 used in our search Alg.2. The results we reported in previous parts are based on $N = 256, K_1 = 8, K_2 = 8$. We change the value of N to 128 and 512, K_2 to 4 and 16, and show the searching curve on f^6 in Fig. 9. The parameter K_1 that selects top candidates in f^{b-2} is not compared for $b = 6$ since there are only 5 candidates in f^4. As can be seen, all the different settings perform similar and obviously outperform the Greedy baseline.

4) Running time analysis: We show the running time of different components in AutoSF in Tab. VII. First, the filter and the predictor (including SRF computation) take much shorter running time compared with that of the model training. Then, as each greedy step contains 256 model training, the best SFs can be searched within only several hours (on 8 GPUs), except for YAGO3-10 which takes more than one day to evaluate 128 candidates. In comparison, search problem based on reinforcement learning [55] runs over 4 days across 500 GPUs; genetic programming [44] takes 17 days on single GPU; and Bayes optimization [13] trains for several days on CPUs. Thus, the proposed AutoSF makes the search problem on KGE tractable, and it is very efficient in AutoML literature.

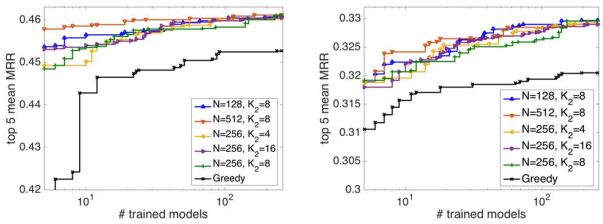


Fig. 9. Comparison of different meta hyper-parameters. Greedy is added here as a contrast. 256 models are evaluated in each setting.

TABLE VII
RUNNING TIME ANALYSIS. WE SHOW THE RUNNING TIME (MIN) PER GREEDY STEP (STEP 2-11 IN ALG.2). APART FROM STEP 2-6 (FILTER), STEP 7,10-11 (PREDICTOR), STEP 8 (TRAIN) AND STEP 9 (EVALUATION), ALL OTHER STEPS TAKE LESS THAN 0.1 MINUTES.

steps	filtering	predictor	train	evaluate
	2-6	7,10-11	8	9
WN18	15.9±0.5	1.8±0.1	475.9±9.5	41.3±0.8
FB15K	16.8±0.7	1.9±0.1	886.3±21.8	153.7±3.9
WN18RR	16.1±1.0	1.8±0.1	271.4±5.1	27.9±0.5
FB15k237	16.6±1.1	1.9±0.1	439.2±11.2	63.5±1.9
YAGO3-10	16.6±0.9	1.7±0.1	1631.1±85.5	141.9±8.9

In addition, since AutoSF search SFs with dimension 64 and then fine-tune the hyper-parameters with $d \in \{256, 512, 1024, 2048\}$ as in [18], [19]. In comparison, the searching cost is comparable with the fine-tune cost which generally needs to train and evaluate hundreds of hyper-parameter settings with large dimension size. In this view, the searching cost is not that expensive.

VI. CONCLUSION

In this paper, we propose AutoSF, an algorithm to automatically design and discover better SFs for KGE. By using a progressive greedy search algorithm enhanced by a filter and a predictor with domain-specific knowledge, AutoSF can efficiently design promising SFs that are KG dependent, new to the literature, and outperform the state-of-the-art SFs designed by humans from the huge search space. In future work, a promising direction is to explore how to efficiently search the network structure for NNMs under domain-specific constraints. The greedy algorithm used in AutoSF somehow limits the exploration in the search space, which is also a potential problem to be addressed.

ACKNOWLEDGMENT

The work is partially supported by the Hong Kong RGC GRF Project 16202218, CRF project C6030-18G, AOE project AoE/E-603/18, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC grants ITS/044/18FX and

ITS/470/18FX, Didi-HKUST joint research lab Grant, Microsoft Research Asia Collaborative Research Grant, Wechat Research Grant and Webank Research Grant.

REFERENCES

[1] I. Balažević, C. Allen, and T. M Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *EMNLP*, 2019.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 35(8):1798–1828, 2013.

[3] J. S Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, pages 2546–2554, 2011.

[4] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, pages 2787–2795, 2013.

[5] B. Csáji. Approximation with artificial neural networks. Technical report, Dept. Science, Eotvos Lorand Univ., 2001.

[6] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2D knowledge graph embeddings. In *AAAI*, 2017.

[7] 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*, pages 601–610, 2014.

[8] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011.

[9] K. Eggensperger, F. Hutter, H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, 2015.

[10] T. Elsken, Jan H. Metzen, and F. Hutter. Neural architecture search: A survey. *JMLR*, 20(55):1–21, 2019.

[11] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*, pages 1436–1445, 2018.

[12] M. Fan, Q. Zhou, E. Chang, and T. F. Zheng. Transition-based knowledge graph embedding with relational mapping properties. In *PACLIC*, 2014.

[13] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *NeurIPS*, pages 2962–2970, 2015.

[14] L. Getoor and B. Taskar. *Introduction to statistical relational learning*. The MIT Press, 2007.

[15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. The MIT press, 2016.

[16] K. Hayashi and M. Shimbo. On the equivalence of holographic and complex embeddings for link prediction. In *ACL*, volume 2, pages 554–559, 2017.

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

[18] M. Kazemi and D. Poole. SimplE embedding for link prediction in knowledge graphs. In *NeurIPS*, 2018.

[19] T. Lacroix, N. Usunier, and G. Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, 2018.

[20] Y. Lin, X. Han, R. Xie, Z. Liu, and M. Sun. Knowledge representation learning: A quantitative review. Technical report, arXiv:1812.10901, 2018.

[21] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187, 2015.

[22] C. Liu, B. Zoph, S. Jonathon, W. Hua, L. Li, F.-F. Li, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[23] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

[24] H. Liu, Y. Wu, and Y. Yang. Analogical inference for multi-relational embeddings. In *ICML*, pages 2168–2178, 2017.

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

[26] F. Mahdisoltani, J. Biega, and F. M Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2013.

[27] C. Meilicke, M. W. Chekol, D. Ruffinelli, and H. Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, 2019.

[28] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[29] M. Nickel, L. Rosasco, and T. Poggio. Holographic embeddings of knowledge graphs. In *AAAI*, pages 1955–1961, 2016.

[30] M. Nickel, V. Tresp, and H. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *ICLR*, 2017.

[32] H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

[33] A. Singhal. Introducing the knowledge graph: Things, not strings. *Official Google blog*, 5, 2012.

[34] R. Socher, D. Chen, C. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, pages 926–934, 2013.

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

[36] J. Tang. AMiner: Mining deep knowledge from big scholar data. In *IW3C2*, pages 373–373. International WWW Committee, 2016.

[37] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In *Workshop on CVSMC*, pages 57–66, 2015.

[38] J. A Tropp. Greed is good: Algorithmic results for sparse approximation. *TIT*, 50(10):2231–2242, 2004.

[39] T. Trouillon, C. Dance, E. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. Knowledge graph completion via complex tensor factorization. *JMLR*, 18(1):4735–4772, 2017.

[40] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 29(12):2724–2743, 2017.

[41] Y. Wang, R. Gemulla, and H. Li. On multi-relational link prediction with bilinear models. In *AAAI*, 2017.

[42] Y. Wang, D. Ruffinelli, S. Broscheit, and R. Gemulla. On evaluating embedding models for knowledge base completion. Technical report, arXiv:1812.06410, 2018.

[43] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, volume 14, pages 1112–1119, 2014.

[44] L. Xie and A. Yuille. Genetic CNN. In *ICCV*, pages 1388–1397, 2017.

[45] Y. Xue, Y. Yuan, Z. Xu, and A. Sabharwal. Expanding holographic embeddings for knowledge completion. In *NeurIPS*, pages 4496–4506, 2018.

[46] 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.

[47] F. Yang, Z. Yang, and W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, pages 2319–2328, 2017.

[48] Q. Yao, X. Chen, Kwok J., Li Y, and Hsieh C. Efficient neural interaction function search for collaborative filtering. In *WWW*, 2020.

[49] Q. Yao, X. Chen, J. Kwok, and Y. Li. Searching for interaction functions in collaborative filtering. Technical report, arXiv preprint arXiv:1906.12091, 2019.

[50] Q. Yao and M. Wang. Taking human out of learning applications: A survey on automated machine learning. Technical report, Arxiv: 1810.13306, 2018.

[51] Q. Yao, J. Xu, W. Tu, and Z. Zhu. Differentiable neural architecture search via proximal iterations. Technical report, arXiv:1905.13577, 2019.

[52] F. Zhang, N. Jing Yuan, D. Lian, X. Xie, and W.-Y. Ma. Collaborative knowledge base embedding for recommender systems. In *SIGKDD*, pages 353–362, 2016.

[53] S. Zhang, Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embedding. In *NeurIPS*, 2019.

[54] Y. Zhang, Q. Yao, Y. Shao, and L. Chen. NSCaching: simple and efficient negative sampling for knowledge graph embedding. In *ICDE*, pages 614–625, 2019.

[55] B. Zoph and Q. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

APPENDIX

A. Proof of Proposition 1

Proof. We consider the four cases separately:

- *symmetric relations*: If $g(\boldsymbol{r})^\top = g(\boldsymbol{r})$ for some $\boldsymbol{r} \in \mathbb{R}^d$, then given a triplet (h, r, t), $f(\boldsymbol{t}, \boldsymbol{r}, \boldsymbol{h}) = \boldsymbol{t}^\top g(\boldsymbol{r}) \boldsymbol{h} = \left(\boldsymbol{t}^\top g(\boldsymbol{r}) \boldsymbol{h}\right)^\top = \boldsymbol{h}^\top g(\boldsymbol{r})^\top \boldsymbol{t} = \boldsymbol{h}^\top g(\boldsymbol{r}) \boldsymbol{t} = f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$, which means $g(\boldsymbol{r})$ can handle symmetric relations.
- *anti-symmetric relations*: If $g(\boldsymbol{r}')^\top = -g(\boldsymbol{r}')$ for some $\boldsymbol{r}' \in \mathbb{R}^d$, then given a triplet (h, r', t), $f(\boldsymbol{t}, \boldsymbol{r}', \boldsymbol{h}) = \boldsymbol{t}^\top g(\boldsymbol{r}') \boldsymbol{h} = \left(\boldsymbol{t}^\top g(\boldsymbol{r}') \boldsymbol{h}\right)^\top = \boldsymbol{h}^\top g(\boldsymbol{r}')^\top \boldsymbol{t} = -\boldsymbol{h}^\top g(\boldsymbol{r}') \boldsymbol{t} = -f(\boldsymbol{h}, \boldsymbol{r}', \boldsymbol{t})$, which means $g(\boldsymbol{r}')$ can handle anti-symmetric relations.
- *general asymmetric relation*: Since $\mathbf{D}_i^{\boldsymbol{r}} = \text{diag}(\boldsymbol{r}_i), i = 1 \dots 4$, then for any scalar $w \in R$, $\mathbf{D}_i^{(w\boldsymbol{r})} = \text{diag}(w\boldsymbol{r}_i) = w\mathbf{D}_i^{\boldsymbol{r}}$. This leads to $g(w\boldsymbol{r}) = wg(\boldsymbol{r})$. Similarly, we have $g(w_1\boldsymbol{r} + w_2\boldsymbol{r}') = w_1 g(\boldsymbol{r}) + w_2 g(\boldsymbol{r}')$ for scalar w_1, w_2.
  If $g(\boldsymbol{r})^\top = g(\boldsymbol{r})$ for some $\boldsymbol{r} \in \mathbb{R}^d$ and $g(\boldsymbol{r}')^\top = -g(\boldsymbol{r}')$ for another $\boldsymbol{r}' \in \mathbb{R}^d$, then for any general assymetric relation r^{asym}, let $\boldsymbol{r}^{\text{asym}} = w_1\boldsymbol{r} + w_2\boldsymbol{r}'$. We have

$$f(\boldsymbol{h}, \boldsymbol{r}^{\text{asym}}, \boldsymbol{t}) = \boldsymbol{h}^\top g(\boldsymbol{r}^{\text{asym}}) \boldsymbol{t} = \boldsymbol{h}^\top g(w_1\boldsymbol{r} + w_2\boldsymbol{r}') \boldsymbol{t} \quad (8)$$
$$= w_1 \boldsymbol{h}^\top g(\boldsymbol{r}) \boldsymbol{t} + w_2 \boldsymbol{h}^\top g(\boldsymbol{r}') \boldsymbol{t} = w_1 f(\boldsymbol{h}, \boldsymbol{R}, \boldsymbol{t}) + w_2 f(\boldsymbol{h}, \boldsymbol{r}', \boldsymbol{t}).$$

  Similarly, we can obtain

$$\begin{aligned} f(\boldsymbol{t}, \boldsymbol{r}^{\text{asym}}, \boldsymbol{h}) &= w_1 f(\boldsymbol{t}, \boldsymbol{r}, \boldsymbol{h}) + w_2 f(\boldsymbol{t}', \boldsymbol{r}', \boldsymbol{h}) \\ &= w_1 f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) - w_2 (\boldsymbol{h}, \boldsymbol{r}', \boldsymbol{t}). \end{aligned} \quad (9)$$

  Then, for any value of the pair $f(\boldsymbol{h}, \boldsymbol{r}^{\text{asym}}, \boldsymbol{t})$ and $f(\boldsymbol{t}, \boldsymbol{r}^{\text{asym}}, \boldsymbol{h})$, there exist appropriate scalars w_1, w_2 by solving (8) and (9) to obtain $\boldsymbol{r}^{\text{asym}} = w_1\boldsymbol{r} + w_2\boldsymbol{r}'$.
- *inverse relations*: Let r^a and r^b be two relations, and assume $\boldsymbol{r}^a = w_1\boldsymbol{r} + w_2\boldsymbol{r}'$ and $\boldsymbol{r}^b = w_3\boldsymbol{r} + w_4\boldsymbol{r}'$ given $g(\boldsymbol{r})^\top = g(\boldsymbol{r}), g(\boldsymbol{r}')^\top = -g(\boldsymbol{r}')$ based on general asymmetric property. Let $w_1 = w_3$ and $w_2 = -w_4$, then

$$\begin{aligned} f(\boldsymbol{t}, \boldsymbol{r}^a, \boldsymbol{h}) &= w_1 f(\boldsymbol{t}, \boldsymbol{r}, \boldsymbol{h}) + w_2 f(\boldsymbol{t}, \boldsymbol{r}', \boldsymbol{h}) \\ &= w_1 f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) - w_2 f(\boldsymbol{h}, \boldsymbol{r}', \boldsymbol{t}) \\ &= w_3 f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) + w_4 f(\boldsymbol{h}, \boldsymbol{r}', \boldsymbol{t}) = f(\boldsymbol{h}, \boldsymbol{r}^b, \boldsymbol{t}), \\ g(\boldsymbol{r}^a) &= w_1 g(\boldsymbol{r}) + w_2 g(\boldsymbol{r}') = w_1 g(\boldsymbol{r})^\top - w_2 g(\boldsymbol{r}')^\top \\ &= w_3 g(\boldsymbol{r})^\top + w_4 g(\boldsymbol{r}')^\top = g(w_3\boldsymbol{r} + w_4\boldsymbol{r}')^\top = g(\boldsymbol{r}^b)^\top. \end{aligned}$$

  This means r^a and r^b are a pair of inverse relations.

Thus, we obtain the proposition. □

B. Proof of Proposition 2

Proof. (i) For each case (S1-11), the SRF is generated based on permutation and flipping signs of the 4 basic values $[v_1; v_2; v_3; v_4]$. Thus, no matter how the structure g changes due to permutation or flipping signs, they will lead to the same feature. Once the matrix $g(\mathbf{v})$ can be symmetric or anti-symmetric under one assignment Si, its corresponding feature will not change regardless of permutation or flipping signs. (ii) The SRF is generated based on the symmetric and skew-symmetric property and each dimension corresponds to a specific case of symmetric or skew-symmetric. Then the predictor can learn higher weights to the dimensions correlates the data's symmetric property well. Besides, this pattern can be easily learned through a few samples. □

C. Design of SRFs

Remark A.1 (SRF)**.** *Let the 1-dimensional degeneration of* $\mathbf{D}_1^{\boldsymbol{r}}$, $\mathbf{D}_2^{\boldsymbol{r}}$, $\mathbf{D}_3^{\boldsymbol{r}}$, $\mathbf{D}_4^{\boldsymbol{r}}$ *be scalars* v_1, v_2, v_3, v_4*. We give* $\mathbf{v} = [v_1; v_2; v_3; v_4]$ *with following assignments:*

- *Four values are non-zero: Four values are non-zero: (S1). All of them have different absolute value, like* $[1; 2; 3; 4]$*; (S2). Two have the same absolute value, and the other two have another same absolute value, like* $[1; 1; 2; 2]$*; (S3). Two of them have the same absolute value while the other two not, like* $[1; 1; 2; 3]$*; (S4). Three of them have the same absolute value while another one not, like* $[1; 1; 1; 2]$*; (S5). All have the same absolute value, like* $[1; 1; 1; 1]$.
- *Three values are non-zero: (S6). All of them have different absolute value, like* $[0; 1; 2; 3]$*; (S7). Two of them have same absolute value, like* $[0; 1; 1; 2]$*; (S8). All of them have same absolute value, like* $[0; 1; 1; 1]$.
- *Two values are non-zero: (S9). They have different absolute value, like* $[0; 0; 1; 2]$*; (S10). They have the same absolute value, like* $[0; 0; 1; 1]$.
- *Only one is non-zero: (S11).* $[0; 0; 0; 1]$.

For each (S1)-(S11), we use permutation and flipping the signs based on the given examples to check if $g(\mathbf{v})$ *can be symmetric or skew-symmetric under each case. As a result, a* $11 \times 2 = 22$ *dimensional SRF returns with little extra cost.*

The 11 cases exhaustively enumerate the possible conditions of $g(\cdot)$ being symmetric or skew-symmetric. What we care most is what kind of symmetric properties $g(\boldsymbol{r})$ can be under these cases. Some data sets may need more features being 1 if it has more symmetric, anti-symmetric and inverse relations like FB15k, but some may not like FB15k237. The process of SRF generation is given in Alg.3.

Algorithm 3 SRF generation for each (S1-11)

Input: the structure of g, SRF$_i$=[0,0] for $i = 1 \dots 11$;
1: **for** $\mathbf{v}$ in the assignment candidates of Si through permuting and flipping signs **do**
2: **if** $g(\mathbf{v}) - g(\mathbf{v})^\top = \mathbf{0}$ **then**
3: SRF$_i$[0] = 1 // symmetric
4: **end if**
5: **if** $g(\mathbf{v}) + g(\mathbf{v})^\top = \mathbf{0}$ **then**
6: SRF$_i$[1] = 1 // skew-symmetric
7: **end if**
8: **end for**
9: **return** SRF$_i$ // 2-dimensional components for each Si.

D. Details of Taking MLP as $\mathcal{G}$

To ensure quick training and testing [6], we use two fully-connected neural networks as the MLP. Specifically, to predict the tail entity, we use NN_1 to combine $\boldsymbol{h}$ and $\boldsymbol{r}$ into $\mathbf{v} = NN_1(\boldsymbol{h}, \boldsymbol{r})$. Then we use the dot product of $\mathbf{v}$ and $\boldsymbol{t}$ as the score. To test the head entity, another network NN_2 is built in similar way and final score is $\langle NN_2(\boldsymbol{t}, \boldsymbol{r}), \boldsymbol{h} \rangle$. The two networks share the same structure (128-64-64) and are trained jointly based on Alg.1.