Bilinear Scoring Function Search for Knowledge Graph Learning

Yongqi Zhang, *Member, IEEE*, Quanming Yao, *Member, IEEE*, and James T. Kwok, *Fellow, IEEE*

Abstract—Learning embeddings for entities and relations in knowledge graph (KG) have benefited many downstream tasks. In recent years, scoring functions, the crux of KG learning, have been human designed to measure the plausibility of triples and capture different kinds of relations in KGs. However, as relations exhibit intricate patterns that are hard to infer before training, none of them consistently perform the best on benchmark tasks. In this paper, inspired by the recent success of automated machine learning (AutoML), we search bilinear scoring functions for different KG tasks through the AutoML techniques. However, it is non-trivial to explore domain-specific information here. We first set up a search space for AutoBLM by analyzing existing scoring functions. Then, we propose a progressive algorithm (AutoBLM) and an evolutionary algorithm (AutoBLM+), which are further accelerated by filter and predictor to deal with the domain-specific properties for KG learning. Finally, we perform extensive experiments on benchmarks in KG completion, multi-hop query, and entity classification tasks. Empirical results show that the searched scoring functions are KG dependent, new to the literature, and outperform the existing scoring functions. AutoBLM+ is better than AutoBLM as the evolutionary algorithm can flexibly explore better structures in the same budget.

Index Terms—Automated machine learning, graph embedding, knowledge graph, neural architecture search

1 Introduction

The knowledge graph (KG) [1], [2], [3] is a graph in which the nodes represent entities, the edges are the relations between entities, and the facts are represented by triples of the form *(head entity, relation, tail entity)* (or (h, r, t) in short). The KG has been found useful in a lot of data mining and machine learning applications and tasks, including question answering [4], product recommendation [5], knowledge graph completion [6], [7], multi-hop query [4], [8], and entity classification [9].

In a KG, plausibility of a fact (h, r, t) is given by $f(h, r, t)$, where f is the *scoring function*. Existing f's are custom-designed by human experts, and can be categorized into the following three families: (i) translational distance models (TDMs) [10], [11], [12], [13], which model the relation embeddings as translations from the head entity embedding to the tail entity embedding; (ii) bilinear model (BLMs) [6], [7], [14], [15], [16], [17], [18], [19], which model the interaction between entities and relations by a bilinear product between the entity and relation embeddings; and (iii) neural network models (NNMs) [20], [21], [22], [23], [24], which use neural networks to capture the interaction. The scoring function can significantly impact KG learning performance [2], [3], [25]. Most TDMs are less expressive and have poor empirical performance [3], [26]. NNMs are powerful but have large numbers of parameters and may overfit the training triples. In comparison, BLMs are more advantageous in that they are easily customized to be expressive, have linear complexities w.r.t. the numbers of entities/relations/dimensions, and have state-of-the-art performance [18]. While a number of BLMs have been proposed, the best BLM is often dataset-specific.

Recently, automated machine learning (AutoML) [27], [28] has demonstrated its power in many machine learning tasks such as hyperparameter optimization (HPO) [29] and neural architecture search (NAS) [30], [31], [32]. The models discovered have better performance than those designed by humans, and the amount of human effort required is significantly reduced. Inspired by its success, we propose in this paper the use of AutoML for the design of KG-dependent scoring functions. To achieve this, one has to pay careful consideration to the three main components in an AutoML algorithm: (i) search space, which identifies important properties of the learning models to search; (ii) search algorithm, which ensures that finding a good model in this space is efficient; and (iii) evaluation method, which offers feedbacks to the search algorithm.

In this paper, we make the following contributions in achieving these goals:

- We design a search space of scoring functions, which includes all the existing BLMs. We further analyze properties of this search space, and provide conditions for a candidate scoring function to be expressive, degenerate, and equivalent to another.

- *Yongqi Zhang is with 4Paradigm Inc. Beijing 100085, China. E-mail: zhangyongqi@4paradigm.com.*
- *Quanming Yao is with the Department of Electronic Engineering, Tsinghua University, Beijing 100190, China, and also with 4Paradigm Inc. Beijing, China. E-mail: qyaoaa@connect.ust.hk.*
- *James T. Kwok is with the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, China. E-mail: jamesk@cse.ust.hk.*

Manuscript received 29 June 2021; revised 25 Feb. 2022; accepted 3 Mar. 2022. Date of publication 7 Mar. 2022; date of current version 6 Jan. 2023.
This work was supported in part by National Key Research and Development Plan under Grant 2021YFE0205700, in part by Chinese National Natural Science Foundation Projects under Grant 61961160704, and in part by the Science and Technology Development Fund of Macau Project under Grant 0070/2020/AMJ.
(Corresponding author: Quanming Yao.)
Recommended for acceptance by J. Li.
Digital Object Identifier no. 10.1109/TPAMI.2022.3157321

TABLE 1
Notations Used in the Paper

$\mathcal{E}, \mathcal{R}, \mathcal{S}$	set of entities, relations, triples
$\lvert\mathcal{E}\rvert, \lvert\mathcal{R}\rvert, \lvert\mathcal{S}\rvert$	number of entities, relations, triples
(h, r, t)	triple of head entity, relation and tail entity
$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}$	embeddings of h, r, and t
$f(h, r, t)$	scoring function for triple (h, r, t)
$\mathbb{R}^d, \mathbb{C}^d, \mathbb{H}^d$	d-dimensional real/complex/hypercomplex space
$\boldsymbol{R}_{(\boldsymbol{r})} \in \mathbb{R}^{d\times d}$	square matrix based on relation embedding $\boldsymbol{r}$
$\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\rangle$	triple product $:= \sum_{i=1}^{d} a_i b_i c_i = \boldsymbol{a}^\top \mathrm{diag}(\boldsymbol{b})\boldsymbol{c}$
$\lVert\boldsymbol{v}\rVert_1$	ℓ_1-norm of vector $\boldsymbol{v}$
$\mathrm{Re}(\boldsymbol{v})$	real part of complex vector $\boldsymbol{v} \in \mathbb{C}^d$
$\bar{\boldsymbol{v}}$	conjugate of complex vector $\boldsymbol{v} \in \mathbb{C}^d$

TABLE 2
Popular Properties in KG Relations

property	examples in WN18/FB15 k	constraint on f
symmetry	`isSimilarTo`, `spouseOf`	$f(t,r,h) = f(h,r,t)$
anti-symmetry	`ancestorOf`, `isPartOf`	$f(t,r,h) = -f(h,r,t)$
general asymmetry	`locatedIn`, `profession`	$f(t,r,h) \neq f(h,r,t)$
inverse	`hypernym`, `hyponym`	$f(t,r,h) = f(h,r,'t)$

- To explore the above search space properties and reduce the computation cost in evaluation, we design a filter to remove degenerate and equivalent structures, and a performance predictor with specifically-designed symmetry-related features (SRF) to select promising structures.
- We customize a progressive algorithm (AutoBLM) and an evolutionary algorithm (AutoBLM+) that, together with the filter and performance predictor, allow flexible exploration of new BLMs.

Extensive experiments are performed on the tasks of KG completion, multi-hop query and entity classification. The results demonstrate that the models obtained by AutoBLM and AutoBLM+ outperform the start-of-the-art human-designed scoring functions. In addition, we show that the customized progressive and evolutionary algorithms are much less expensive than popular search algorithms (random search, Bayesian optimization and reinforcement learning) in finding a good scoring function.

*Differences with the Conference Version.*Compared to the preliminary version published in ICDE 2020 [33], we have made the following important extensions:

1) *Theory*. We add new analysis to the designed search space based on bilinear models. We theoretically prove when the candidates in the search space can be expressive (Section 3.2), degenerate (Section 3.4.1) and equivalent structures (Section 3.4.2).
2) *Algorithm*. We extend the search algorithm with the evolutionary algorithm (Section 4.4), i.e., AutoBLM+. The evolutionary strategy in Algorithm 4 can explore better in the search space, and can also leverage the filter and predictor to deal with the domain-specific properties.
3) *Tasks*. We extend AutoBLM and AutoBLM+ to two new tasks, namely, multi-hop query (Section 5.2) and entity classification in (Section 5.3). We show that the search problem can be well adopted to these new scenarios, and achieve good empirical performance.
4) *Ablation Study*. We conduct more experiments on the performance (Section 5.1.2 and 5.1.3) and analysis (Section 5.1.4) of the new search algorithm, analysis on the influence of K (Section 5.1.7), and the problem of parameter sharing (Section 5.1.8) to analyze the design schemes in the search space and search algorithm.

Notations. In this paper, vectors are denoted by lowercase boldface, and matrix by uppercase boldface. The important notations are listed in Table 1.

2 Background and Related Works

2.1 Scoring Functions for Knowledge Graph (KG)

A knowledge graph (KG) can be represented by a third-order tensor $\mathbf{G} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$, in which $G_{hrt} = 1$ if the corresponding triple (h, r, t) exists in the KG, and 0 otherwise. The *scoring function* $f(h, r, t) : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$ measures plausibility of the triple (h, r, t). As introduced in Section 1, it is desirable for a scoring function to be able to represent any of the symmetric, anti-symmetric, general asymmetric and inverse KG relations in Table 2.

Definition 1 (Expressiveness [14], [34], [35]). *A scoring function is* fully expressive *if for any KG G and the corresponding tensor $\mathbf{G} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$, one can find an instantiation f of the scoring function such that $f(h, r, t) = G_{hrt}$, $\forall h, t \in \mathcal{E}, r \in \mathcal{R}$.*

Not all scoring functions are fully expressive. For example, consider a KG with two people *A*, *B*, and a relation "OlderThan". Obviously, we can have either *(A, OlderThan, B)* or *(B, OlderThan, A)*, but not both. The scoring function $f(h, r, t) = \langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}\rangle = \sum_{i=1}^{d} h_i r_i t_i$, where $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}$ are d-dimensional embeddings of h, r and t, respectively, cannot be fully expressive since $f(h, r, t) = f(t, r, h)$.

On the other hand, while expressiveness indicates the ability of f to fit a given KG, it may not generalize well when inference on different KGs. As real-world KGs can be very sparse [1], [3], a scoring function with a large amount of trainable parameters may overfit the training triples. Hence, it is also desirable that the scoring function has only a manageable number of parameters.

In the following, we review the three main types of scoring functions, namely, translational distance model (TDM), neural network model (NNM), and biLinear model (BLM). As will be seen, many TDMs (such as TransE [10] and TransH [11]) cannot model the symmetric relations well [3], [36]. Neural network models, though fully expressive, have large numbers of parameters. This not only prevents the model from generalizing well on unobserved triples in a sparse KG, but also increases the training and inference costs [18], [21], [24]. In comparison, BLMs (except DistMult) can model all relation pattens in Table 2 and are fully expressive. Besides, these models (except RESCAL and TuckER) have moderate complexities (with the number of parameters linear in $|\mathcal{E}|, |\mathcal{R}|$ and d). Therefore, we consider BLM as a better choice, and it will be our focus in this paper.

TABLE 3
The Representative BLM Scoring Functions. For Each Scoring Function We Show the Definitions, Expressiveness in Definition 1, the Ability to Model All Common Relation Patterns in Table 2 ("RP" for Short), and the Number of Parameters

scoring function	definition	expressiveness	RP	# parameters
RESCAL [6]	$\boldsymbol{h}^\top \boldsymbol{R}_{(\boldsymbol{r})} \boldsymbol{t}$	$\surd$	$\surd$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d^2)$
DistMult [7]	$\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle$	$\times$	$\times$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
ComplEx [14]/HolE [15]	$\mathrm{Re}(\boldsymbol{h} \otimes \boldsymbol{r} \otimes \bar{\boldsymbol{t}})$	$\surd$	$\surd$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
Analogy [16]	$\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle + \mathrm{Re}(\underline{\boldsymbol{h}} \otimes \underline{\boldsymbol{r}} \otimes \bar{\underline{\boldsymbol{t}}})$	$\surd$	$\surd$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
SimplE [17]/CP [18]	$\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \underline{\boldsymbol{t}} \rangle + \langle \underline{\boldsymbol{h}}, \underline{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle$	$\surd$	$\surd$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
QuatE [19]	$\boldsymbol{h} \odot \boldsymbol{r} \odot \boldsymbol{t}$	$\surd$	$\surd$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d)$
TuckER [35]	$\mathcal{W} \times_1 \boldsymbol{h} \times_2 \boldsymbol{r} \times_3 \boldsymbol{t}$	$\surd$	$\surd$	$O(\lvert\mathcal{E}\rvert d + \lvert\mathcal{R}\rvert d + d^3)$

Translational Distance Model (TDM). Inspired by analogy results in word embeddings [37], scoring functions in TDM take the relation r as a translation from h to t. The most representative TDM is TransE [10], with $f(h,r,t) = -\|\boldsymbol{t} - (\boldsymbol{h} + \boldsymbol{r})\|_1$. In order to handle one-to-many, many-to-one and many-to-many relations, TransH [11] and TransR [12] introduce additional vectors/matrices to map the entities to a relation-specific hyperplane. The more recent RotatE [13] treats the relations as rotations in a complex-valued space: $f(h,r,t) = -\|\boldsymbol{t} - \boldsymbol{h} \otimes \boldsymbol{r}\|_1$, where $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{C}^d$ and $\otimes$ is the Hermitian product [14]. As discussed in [34], most TDMs are not fully expressive. For example, TransE and TransH cannot model symmetric relations.

Neural Network Model (NNM). NNMs take the entity and relation embeddings as input, and output a probability for the triple (h,r,t) using a neural network. Earlier works are based on multilayer perceptrons [20] and neural tensor networks [38]. More recently, ConvE [21] uses the convolutional network to capture interactions among embedding dimensions. By sampling relational paths [39] from the KG, RSN [22] and Interstellar [40] use the recurrent network [41] to recurrently combine the head entity and relation with a step-wise scoring function. As the KG is a graph, R-GCN [23] and CompGCN [24] use the graph convolution network [9] to aggregate entity-relation compositions layer by layer. Representations at the final layer are then used to compute the scores. Because of the use of an additional neural network, NNM requires more parameters and has larger model complexity.

BiLinear Model (BLM). BLMs model the KG relation as a bilinear product between entity embeddings. For example, RESCAL [6] defines f as: $f(h,r,t) = \boldsymbol{h}^\top \boldsymbol{R}_{(\boldsymbol{r})} \boldsymbol{t}$, where $\boldsymbol{h}, \boldsymbol{t} \in \mathbb{R}^d$, and $\boldsymbol{R}_{(\boldsymbol{r})} \in \mathbb{R}^{d \times d}$. To avoid overfitting, DistMult [7] requires $\boldsymbol{R}_{(\boldsymbol{r})}$ to be diagonal, and $f(h,r,t)$ reduces to a triple product: $f(h,r,t) = \boldsymbol{h}^\top \mathrm{diag}(\boldsymbol{r}) \boldsymbol{t} = \langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle$. However, it can only model symmetric relations. To capture anti-symmetric relations, ComplEx [14] uses complex-valued embeddings $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{C}^d$ with $f(h,r,t) = \mathrm{Re}(\boldsymbol{h}^\top \mathrm{diag}(\boldsymbol{r}) \bar{\boldsymbol{t}}) = \mathrm{Re}(\boldsymbol{h} \otimes \boldsymbol{r} \otimes \bar{\boldsymbol{t}})$, where $\otimes$ is the Hermitian product in complex space [14]. HolE [15] uses the circular correlation instead of the dot product, but is shown to be equivalent to ComplEx [42].

Analogy [16] decomposes the head embedding $\boldsymbol{h}$ into a real part $\hat{\boldsymbol{h}} \in \mathbb{R}^d$ and a complex part $\underline{\boldsymbol{h}} \in \mathbb{C}^d$. Relation embedding $\boldsymbol{r}$ (resp. tail embedding $\boldsymbol{t}$) is similarly decomposed into a real part $\hat{\boldsymbol{r}}$ (resp. $\hat{\boldsymbol{t}}$) and a complex part $\underline{\boldsymbol{r}}$ (resp. $\underline{\boldsymbol{t}}$). f is then written as: $f(h,r,t) = \langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle + \mathrm{Re}(\underline{\boldsymbol{h}} \otimes \underline{\boldsymbol{r}} \otimes \bar{\underline{\boldsymbol{t}}})$, which can be regarded as a combination of DistMult and ComplEx. To simultaneously model the forward triplet (h,r,t) and its inverse $(t,r,'h)$, SimplE [17] / CP [18] similarly splits the embeddings to a forward part ($\hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \in \mathbb{R}^d$) and a backward part ($\underline{\boldsymbol{h}}, \underline{\boldsymbol{r}}, \underline{\boldsymbol{t}} \in \mathbb{R}^d$): $f(h,r,t) = \langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \underline{\boldsymbol{t}} \rangle + \langle \underline{\boldsymbol{t}}, \underline{\boldsymbol{r}}, \hat{\boldsymbol{h}} \rangle$. To allow more interactions among embedding dimensions, the recent QuatE [19] uses embeddings in the hypercomplex space ($\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{H}^d$) to model $f(h,r,t) = \boldsymbol{h} \odot \boldsymbol{r} \odot \boldsymbol{t}$ where $\odot$ is the Hamilton product. By using the Tucker decomposition [43], TuckER [35] proposes a generalized bilinear model and introduces more parameters in the core tensor $\mathcal{W} \in \mathbb{R}^{d \times d \times d}$: $f(h,r,t) = \mathcal{W} \times_1 \boldsymbol{h} \times_2 \boldsymbol{r} \times_3 \boldsymbol{t}$, where $\times_i$ is the tensor product along the ith mode. A summary of these BLMs is in Table 3.

2.2 Common Learning Tasks in KG

2.2.1 KG Completion

KG is naturally incomplete [1], and KG completion is a representative task in KG learning [3], [6], [7], [10], [14], [17], [21]. Scores on the observed triples are maximized, while those on the non-observed triplets are minimized. After training, new triples can be added to the KG by entity prediction with either a missing head entity $(?,r,t)$ or a missing tail entity $(h,r,?)$ [3]. For each kind of query, we enumerate all the entities $e \in \mathcal{E}$ and compute the corresponding scores $f(e,r,t)$ or $f(h,r,e)$. Entities with larger scores are more likely to be true facts. Most of the models in Section 2.1 can be directly used for KG completion.

2.2.2 Multi-Hop Query

In KG completion, we predict queries $(h,r,?)$ with length one, i.e., 1-hop query. In practice, there can be multi-hop queries with lengths larger than one [3], [8], [39]. For example, one may want to predict *"who is the sister of Tony's mother"*. To solve this problem, we need to solve the length-2 query problem $(?, sister \circ mother, Tony)$ with the relation composition operator $\circ$.

Given the KG $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{S}\}$, let φ_r, corresponding to the relation $r \in \mathcal{R}$, be a binary function $\mathcal{E} \times \mathcal{E} \mapsto \{True, False\}$. The multi-hop query is defined as follows.

Definition 2 (Multi-hop query [4], [8]). *The multi-hop query $(e_0, r_1 \circ r_2 \circ \dot{s} \circ r_L, e_?)$ with length $L > 1$ is defined as $\exists e_1 \dot{s} e_{L-1}, e_? : \varphi_{r_1}(e_0, e_1) \wedge \varphi_{r_2}(e_1, e_2) \wedge \dot{s} \wedge \varphi_{r_L}(e_{L-1}, e_?)$ where $\wedge$ is the conjunction operation, e_0 is the starting entity, $e_?$ is the entity to predict, and $e_1 \dot{s} e_{L-1}$ are intermediate entities that connect the conjunctions.*

Similar to KG completion, plausibility of a query $(e_0, r_1 \circ r_2 \circ \dot{s} \circ r_L, e_L)$ is measured by a scoring function [8], [39]:

$$f(e_0, r_1 \circ r_2 \circ \dot{s} \circ r_L, e_L) = \boldsymbol{e}_0^\top \boldsymbol{R}_{(r_1)} \cdots \boldsymbol{R}_{(r_L)} \boldsymbol{e}_L, \quad (1)$$

where $\boldsymbol{R}_{(r_i)}$ is a relation-specific matrix of the ith relation. The key is on how to model the composition of relations in the embedding space. Based on TransE [10], TransE-Comp [39] models the composition operator as addition, and defines the scoring function as $f(e_0, r_1 \circ \cdots \circ r_L, e_L) = -\|\boldsymbol{e}_L - (\boldsymbol{e}_0 + \boldsymbol{r}_1 + \cdots + \boldsymbol{r}_L)\|_1$. Diag-Comp [39] uses the multiplication operator in DistMult [7] to define $f(e_0, r_1 \circ \cdots \circ r_L, e_L) = \boldsymbol{e}_0^\top \boldsymbol{D}_{\boldsymbol{r}_1} \cdots \boldsymbol{D}_{\boldsymbol{r}_L} \boldsymbol{e}_L$, where $\boldsymbol{D}_{\boldsymbol{r}_i} = \text{diag}(\boldsymbol{r}_i)$. Following RESCAL [6], GQE [8] performs the composition with a product of relational matrices $\{\boldsymbol{R}_{(r_1)}, \ldots, \boldsymbol{R}_{(r_L)}\}$, as: $f(e_0, r_1 \circ \cdots \circ r_L, e_L) = \boldsymbol{e}_0^\top \boldsymbol{R}_{(r_1)} \cdots \boldsymbol{R}_{(r_L)} \boldsymbol{e}_L$. More recently, Query2box [4] models the composition of relations as a projection of box embeddings and defines an entity-to-box distance to measure the score.

2.2.3 Entity Classification

Entity classification aims at predicting the labels of the unlabeled entities. Since the labeled entities are few, a common approach is to use a graph convolutional network (GCN) [9], [44] to aggregate neighborhood information. The GCN operates on the local neighborhoods of each entity and aggregates the representations layer-by-layer as:

$$\boldsymbol{e}_i^{\ell+1} = \sigma\left(\boldsymbol{W}_0^\ell \boldsymbol{e}_i^\ell + \sum_{j:(i,r,j)\in\mathcal{S}_{\text{tra}}} \boldsymbol{W}^\ell \boldsymbol{e}_j^\ell\right),$$

where $\mathcal{S}_{\text{tra}}$ contains all the training triples, σ is the activation function, $\boldsymbol{e}_i^\ell, \boldsymbol{e}_j^\ell$ are the layer-ℓ representations of i and the neighboring entities j, respectively, and $\boldsymbol{W}_0^\ell, \boldsymbol{W}^\ell \in \mathbb{R}^{d\times d}$ are weighting matrices sharing across different entities in the ℓth layer.

GCN does not encode relations in edges. To alleviate this problem, R-GCN [23] and CompGCN [24] encode relation r and entity j together by a composition function ϕ:

$$\boldsymbol{e}_i^{\ell+1} = \sigma\left(\boldsymbol{W}_0^\ell \boldsymbol{e}_i^\ell + \sum_{(r,j):(i,r,j)\in\mathcal{S}_{\text{tra}}} \boldsymbol{W}^\ell \phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell)\right),$$

where $\boldsymbol{r}^\ell$ is the representation of relation r at the ℓth layer. The composition function $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell)$ can significantly impact performance [24]. R-GCN uses the composition operator in RESCAL [6], and defines $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{R}_{(r)}^\ell \boldsymbol{e}_j^\ell$, where $\boldsymbol{R}_{(r)}^\ell$ is a relation-specific weighting matrix in the ℓth layer. CompGCN, following TransE [10], DistMult [7] and HolE [15], defines three operators: subtraction $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{e}_j^\ell - \boldsymbol{r}^\ell$, multiplication $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{e}_j^\ell \cdot \boldsymbol{r}^\ell$ where $\cdot$ is the element-wise product, and circular correlation $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{e}_j^\ell \star \boldsymbol{r}^\ell$ where $[\boldsymbol{a}\star\boldsymbol{b}]_k = \sum_{i=1}^d a_i b_{k+i-1 \bmod d}$.

2.3 Automated Machine Learning (AutoML)

Recently, automated machine learning (AutoML) [27], [28] has demonstrated its advantages in the design of better machine learning models. AutoML is often formulated as a bi-level optimization problem [45], in which model parameters are updated from the training data in the inner loop, while hyper-parameters are tuned from the validation data in the outer loop. There are three important components in AutoML [27], [28], [46]:

1) *Search space:* This identifies important properties of the learning models to search. The search space should be large enough to cover most manually-designed models, while specific enough to ensure that the search will not be too expensive.
2) *Search algorithm:* A search algorithm is used to search for good solutions in the designed space. Unlike convex optimization problems, there is no universally efficient optimization tool.
3) *Evaluation:* Since the search aims at improving performance, evaluation is needed to offer feedbacks to the search algorithm. The evaluation procedure should be fast and the signal should be accurate.

2.3.1 Neural Architecture Search (NAS)

Recently, a variety of NAS algorithms have been developed to facilitate efficient search of deep networks [28], [30], [32]. They can generally be divided into model-based approach and sample-based approach [27]. The model-based approach builds a surrogate model for all candidates in the search space, and selects candidates with promising performance using methods such as Bayesian optimization [29], reinforcement learning [30], [47], and gradient descent [31], [48]. It requires evaluating a large number of architectures for training the surrogate model or requires a differentiable objective w.r.t. the architecture. The sample-based approach is more flexible and explores new structures in the search space by using heuristics such as progressive algorithm [49] and evolutionary algorithm [50].

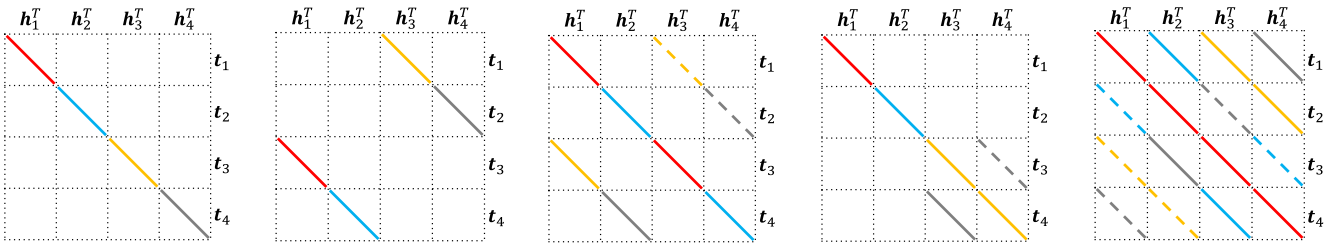
As for evaluation, parameter-sharing [31], [47], [48] allows faster architecture evaluation by combining architectures in the whole search space with the same set of parameters. However, the obtained results can be sensitive to initialization, which hinders reproducibility. On the other hand, stand-alone methods [30], [49], [50] train and evaluate the different models separately. They are slower but more reliable. To improve its efficiency, a predictor can be used to select promising architectures [49] before it is fully trained.

3 Automated Bilinear Model

In the last decade, KG learning has been improving with new scoring function designs. However, as different KGs may have different properties, it is unclear how a proper scoring function can be designed for a particular KG. This raises the question: *Can we automatically design a scoring function for a given KG*? To address this question, we first provide a unified view of BLMs, and then formulate the design of scoring function as an AutoML problem: AutoBLM ("automated bilinear model").

3.1 A Unified View of BLM

Recall from Section 2.1 that a BLM may operate in the real/complex/hypercomplex space. To write the different BLMs in the same form, we first unify them to the same representation space. The idea is to partition each of the embeddings $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}$ to $K = 4$ equal-sized chunks, as $\boldsymbol{h} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_4], \boldsymbol{r} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_4]$ and $\boldsymbol{t} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_4]$. The BLM is then written in terms of $\{\langle \boldsymbol{h}_i, \boldsymbol{r}_j, \boldsymbol{t}_k\rangle\}_{i,j,k\in\{1,\ldots,4\}}$.



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

Fig. 1. The forms of $\boldsymbol{R}_{(\boldsymbol{r})}$ for representative BLMs (best viewed in color). Different colors correspond to different parts of $[\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_4]$ (red for $\boldsymbol{r}_1$, blue for $\boldsymbol{r}_2$, yellow for $\boldsymbol{r}_3$, gray for $\boldsymbol{r}_4$). Solid lines mean positive values, while dashed lines mean negative values. The empty parts have value zero.

- DistMult [7], which uses $f(h,r,t) = \langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle$. We simply split $\boldsymbol{h} \in \mathbb{R}^d$ (and analogously $\boldsymbol{r}$ and $\boldsymbol{t}$) into 4 parts as $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \boldsymbol{h}_4\}$, where $\boldsymbol{h}_i \in \mathbb{R}^{d/4}$ for $i = 1, 2, 3, 4$. Obviously,

$$\begin{aligned}&\langle \boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \rangle \\ &= \langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_2 \rangle + \langle \boldsymbol{h}_3, \boldsymbol{r}_3, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_4, \boldsymbol{r}_4, \boldsymbol{t}_4 \rangle.\end{aligned}$$

- SimplE [17] / CP [18], which uses $f(h,r,t) = \langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \underline{\boldsymbol{t}} \rangle + \langle \underline{\boldsymbol{t}}, \underline{\boldsymbol{r}}, \hat{\boldsymbol{h}} \rangle$. We split $\hat{\boldsymbol{h}} \in \mathbb{R}^d$ (and analogously $\hat{\boldsymbol{r}}$ and $\hat{\boldsymbol{t}}$) into 2 parts as $\{\boldsymbol{h}_1, \boldsymbol{h}_2\}$ (where $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{R}^{d/2}$), and similarly $\underline{\boldsymbol{h}}$ as $\{\boldsymbol{h}_3, \boldsymbol{h}_4\}$ (and analogously $\underline{\boldsymbol{r}}$ and $\underline{\boldsymbol{t}}$). Then,

$$\begin{aligned}&\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \underline{\boldsymbol{t}} \rangle + \langle \underline{\boldsymbol{t}}, \underline{\boldsymbol{r}}, \hat{\boldsymbol{h}} \rangle \\ &= \langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_4 \rangle + \langle \boldsymbol{h}_3, \boldsymbol{r}_3, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_4, \boldsymbol{r}_4, \boldsymbol{t}_2 \rangle.\end{aligned}$$

- ComplEx [14] / HolE [15], which uses $f(h,r,t) = \mathrm{Re}(\boldsymbol{h} \otimes \boldsymbol{r} \otimes \bar{\boldsymbol{t}})$, where $\boldsymbol{h}, \boldsymbol{r}, \bar{\boldsymbol{t}}$ are complex-valued. Recall that any complex vector $\boldsymbol{v} \in \mathbb{C}^d$ is of the form $\boldsymbol{v}_r + i\boldsymbol{v}_i$, where $\boldsymbol{v}_r \in \mathbb{R}^d$ is the real part and $\boldsymbol{v}_i \in \mathbb{R}^d$ is the imaginary part. Thus,

$$\begin{aligned}\mathrm{Re}(\boldsymbol{h} \otimes \boldsymbol{r} \otimes \bar{\boldsymbol{t}}) &= \langle \boldsymbol{h}_r, \boldsymbol{r}_r, \boldsymbol{t}_r \rangle + \langle \boldsymbol{h}_i, \boldsymbol{r}_r, \boldsymbol{t}_i \rangle \\ &\quad + \langle \boldsymbol{h}_r, \boldsymbol{r}_i, \boldsymbol{t}_i \rangle - \langle \boldsymbol{h}_i, \boldsymbol{r}_i, \boldsymbol{t}_r \rangle. \end{aligned} \tag{2}$$

  We split $\boldsymbol{h}_r \in \mathbb{R}^d$ (and analogously $\boldsymbol{r}_r$ and $\boldsymbol{t}_r$) into 2 parts $\{\boldsymbol{h}_1, \boldsymbol{h}_2\}$ (where $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{R}^{d/2}$), and similarly $\boldsymbol{h}_i = \{\boldsymbol{h}_3, \boldsymbol{h}_4\}$ (and analogously $\boldsymbol{r}_i$ and $\boldsymbol{t}_i$). Then,

$$\begin{aligned}&\mathrm{Re}(\boldsymbol{h} \otimes \boldsymbol{r} \otimes \bar{\boldsymbol{t}}) \\ &= (\langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_2 \rangle) + (\langle \boldsymbol{h}_3, \boldsymbol{r}_1, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_4, \boldsymbol{r}_2, \boldsymbol{t}_4 \rangle) \\ &\quad + (\langle \boldsymbol{h}_1, \boldsymbol{r}_3, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_4, \boldsymbol{t}_4 \rangle) - (\langle \boldsymbol{h}_3, \boldsymbol{r}_3, \boldsymbol{t}_1 \rangle - \langle \boldsymbol{h}_4, \boldsymbol{r}_4, \boldsymbol{t}_2 \rangle).\end{aligned}$$

- Analogy [16], which uses $f(h,r,t) = \langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle + \mathrm{Re}(\underline{\boldsymbol{h}} \otimes \underline{\boldsymbol{r}} \otimes \bar{\underline{\boldsymbol{t}}})$. We split $\hat{\boldsymbol{h}} \in \mathbb{R}^d$ (and analogously $\hat{\boldsymbol{r}}$ and $\hat{\boldsymbol{t}}$) into 2 parts $\{\boldsymbol{h}_1, \boldsymbol{h}_2\}$ (where $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{R}^{d/2}$), and similarly $\underline{\boldsymbol{h}} \in \mathbb{C}^{d/2}$ (and analogously $\underline{\boldsymbol{r}}$ and $\underline{\boldsymbol{t}}$) into 2 parts $\{\boldsymbol{h}_3, \boldsymbol{h}_4\}$ (where $\boldsymbol{h}_3, \boldsymbol{h}_4 \in \mathbb{R}^{d/2}$). Then,

$$\begin{aligned}&\langle \hat{\boldsymbol{h}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{t}} \rangle + \mathrm{Re}(\underline{\boldsymbol{h}} \otimes \underline{\boldsymbol{r}} \otimes \bar{\underline{\boldsymbol{t}}}) \\ &= \langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_2 \rangle + \langle \boldsymbol{h}_3, \boldsymbol{r}_3, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_3, \boldsymbol{r}_4, \boldsymbol{t}_4 \rangle \\ &\quad + \langle \boldsymbol{h}_4, \boldsymbol{r}_3, \boldsymbol{t}_4 \rangle - \langle \boldsymbol{h}_4, \boldsymbol{r}_4, \boldsymbol{t}_3 \rangle.\end{aligned}$$

- QuatE [19], which uses $f(h,r,t) = \boldsymbol{h} \odot \boldsymbol{r} \odot \boldsymbol{t}$. Recall that any hypercomplex vector $\boldsymbol{v} \in \mathbb{H}^d$ is of the form $\boldsymbol{v}_1 + i\boldsymbol{v}_2 + j\boldsymbol{v}_3 + k\boldsymbol{v}_4$, where $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4 \in \mathbb{R}^d$. Thus,

$$\begin{aligned}&\boldsymbol{h} \odot \boldsymbol{r} \odot \boldsymbol{t} \\ &= \langle \boldsymbol{h}_1, \boldsymbol{r}_1, \boldsymbol{t}_1 \rangle - \langle \boldsymbol{h}_1, \boldsymbol{r}_2, \boldsymbol{t}_2 \rangle - \langle \boldsymbol{h}_1, \boldsymbol{r}_3, \boldsymbol{t}_3 \rangle - \langle \boldsymbol{h}_1, \boldsymbol{r}_4, \boldsymbol{t}_4 \rangle \\ &\quad + \langle \boldsymbol{h}_2, \boldsymbol{r}_2, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_1, \boldsymbol{t}_2 \rangle + \langle \boldsymbol{h}_2, \boldsymbol{r}_4, \boldsymbol{t}_3 \rangle - \langle \boldsymbol{h}_2, \boldsymbol{r}_3, \boldsymbol{t}_4 \rangle \\ &\quad + \langle \boldsymbol{h}_3, \boldsymbol{r}_3, \boldsymbol{t}_1 \rangle - \langle \boldsymbol{h}_3, \boldsymbol{r}_4, \boldsymbol{t}_2 \rangle + \langle \boldsymbol{h}_3, \boldsymbol{r}_1, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_3, \boldsymbol{r}_2, \boldsymbol{t}_4 \rangle \\ &\quad + \langle \boldsymbol{h}_4, \boldsymbol{r}_4, \boldsymbol{t}_1 \rangle + \langle \boldsymbol{h}_4, \boldsymbol{r}_3, \boldsymbol{t}_2 \rangle - \langle \boldsymbol{h}_4, \boldsymbol{r}_2, \boldsymbol{t}_3 \rangle + \langle \boldsymbol{h}_4, \boldsymbol{r}_1, \boldsymbol{t}_4 \rangle.\end{aligned}$$

As $\langle \boldsymbol{h}_i, \boldsymbol{r}_k, \boldsymbol{t}_j \rangle = \boldsymbol{h}_i^\top \mathrm{diag}(\boldsymbol{r}_k) \boldsymbol{t}_j$, all the above BLMs can be written in the form of a bilinear function

$$\boldsymbol{h}^\top \boldsymbol{R}_{(\boldsymbol{r})} \boldsymbol{t}, \tag{3}$$

where[1] $\boldsymbol{h} = [\boldsymbol{h}_1^\top, \ldots, \boldsymbol{h}_4^\top]^\top, \boldsymbol{t} = [\boldsymbol{t}_1; \dot{\boldsymbol{s}}; \boldsymbol{t}_4] \in \mathbb{R}^d$, and $\boldsymbol{R}_{(\boldsymbol{r})} \in \mathbb{R}^{d \times d}$ is a matrix with 4×4 blocks, each block being either $\boldsymbol{0}, \pm\mathrm{diag}(\boldsymbol{r}_1), \dot{\boldsymbol{s}}$, or $\pm\mathrm{diag}(\boldsymbol{r}_4)$. Fig. 1 shows graphically the $\boldsymbol{R}_{(\boldsymbol{r})}$ for the BLMs considered.

3.2 Unified Bilinear Model

Using the above unified representation, the design of BLM becomes designing $\boldsymbol{R}_{(\boldsymbol{r})}$ in (3).

Definition 3 (Unified BiLinear Model). *The desired scoring function is of the form*

$$f_{\boldsymbol{A}}(h,r,t) = \sum_{i,j=1}^{K} sign(A_{ij}) \left\langle \boldsymbol{h}_i, \boldsymbol{r}_{|A_{ij}|}, \boldsymbol{t}_j \right\rangle, \tag{4}$$

where

$$\boldsymbol{A} \in \{0, \pm 1, \ldots, \pm K\}^{K \times K} \tag{5}$$

is called the structure matrix. *Here, we define* $\boldsymbol{r}_0 \equiv \boldsymbol{0}$, *and* $sign(0) = 0$.

It can be easily seen that this covers all the BLMs in Section 3.1 when $K = 4$. Let $g_K(\boldsymbol{A}, \boldsymbol{r})$ be a matrix with $K \times K$ blocks, with its (i,j)-th block:

$$[g_K(\boldsymbol{A}, \boldsymbol{r})]_{ij} = \mathrm{sign}(A_{ij}) \cdot \mathrm{diag}(\boldsymbol{r}_{|A_{ij}|}). \tag{6}$$

The form in (4) can be written more compactly as

$$f_{\boldsymbol{A}}(h,r,t) = \boldsymbol{h}^\top g_K(\boldsymbol{A}, \boldsymbol{r}) \boldsymbol{t}. \tag{7}$$

A graphical illustration is shown in Fig. 2.

The following Proposition gives a necessary and sufficient condition for the BLM with scoring function in (7) to be fully expressive. The proof is in Appendix A.1.

1. With a slight abuse of notations, we still use d to denote the dimensionality after this transformation.

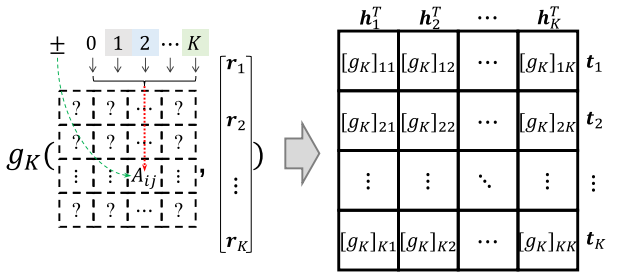


Fig. 2. A graphical illustration of the proposed form of $f_{\boldsymbol{A}}(h,r,t)$.

Proposition 1. *Let*

$$\mathcal{C} \equiv \{\boldsymbol{r} \in \mathbb{R}^K \mid \boldsymbol{r} \neq \boldsymbol{0}, \\ r[i] \in \{0, \pm 1, \ldots, \pm K\}, i = 1, \ldots, K\}. \quad (8)$$

Given an $\boldsymbol{A}$ in (5), the bilinear model with scoring function (7) is fully expressive if

1) *$\exists \hat{\boldsymbol{r}} \in \mathcal{C}$ such that $g_K(\boldsymbol{A}, \hat{\boldsymbol{r}})$ is symmetric (i.e., $g_K(\boldsymbol{A}, \hat{\boldsymbol{r}})^\top = g_K(\boldsymbol{A}, \hat{\boldsymbol{r}})$), and*
2) *$\exists \check{\boldsymbol{r}} \in \mathcal{C}$ such that $g_K(\boldsymbol{A}, \check{\boldsymbol{r}})$ is skew-symmtric (i.e., $g_K(\boldsymbol{A}, \check{\boldsymbol{r}})^\top = -g_K(\boldsymbol{A}, \check{\boldsymbol{r}})$).*

Table 4 shows examples of $\hat{\boldsymbol{r}}$ and $\check{\boldsymbol{r}}$ for the existing BLMs (ComplEx, HolE, Analogy, SimplE, CP, and QuatE), thus justifying that they are fully expressive.

3.3 Searching for BLMs

Using the family of unified BLMs in Definition 3 as the search space $\mathcal{A}$ for structure matrix $\boldsymbol{A}$, the search for a good data-specific BLM can be formulated as the following AutoML problem.

Definition 4 (Bilinear Model Search (AutoBLM)). *Let $F(\boldsymbol{P}; \boldsymbol{A})$ be a KG embedding model (where $\boldsymbol{P}$ includes the entity embedding matrix $\boldsymbol{E}$ and relation embedding matrix $\boldsymbol{R}$, and $\boldsymbol{A}$ is the structure matrix) and $M(F, \mathcal{S})$ be the performance measurement of F on triples $\mathcal{S}$ (the higher the better). The AutoBLM problem is formulated as:*

$$\boldsymbol{A}^* \in Arg\max_{\boldsymbol{A} \in \mathcal{A}} M(F(\boldsymbol{P}^*; \boldsymbol{A}), \mathcal{S}_{\text{val}}) \quad (9)$$

$$\text{s.t. } \boldsymbol{P}^* = \arg\max_{\boldsymbol{P}} M(F(\boldsymbol{P}; \boldsymbol{A}), \mathcal{S}_{\text{tra}}), \quad (10)$$

where

$$\mathcal{A} = \{\boldsymbol{A} = [A_{ij}] \in \mathbb{R}^{K \times K} \\ \mid A_{ij} \in \{0, \pm 1, \ldots, \pm K\} \ \forall i, j = 1, \ldots, K\}, \quad (11)$$

contains all the possible choices of $\boldsymbol{A}$, $\mathcal{S}_{\text{tra}}$ is the training set, and $\mathcal{S}_{\text{val}}$ is the validation set.

As a bi-level optimization problem, we first train the model to obtain $\boldsymbol{P}^*$ (converged model parameters) on the training set $\mathcal{S}_{\text{tra}}$ by (10), and then search for a better $\boldsymbol{A}$ (and consequently a better relation matrix $g_K(\boldsymbol{A}, \boldsymbol{r})$) based on its performance M on the validation set $\mathcal{S}_{\text{val}}$ in (9). Note that the objectives in (9) and (10) are non-convex, and the search space is large (with $(2K+1)^{K^2}$ candidates, as can be seen from (5)). Thus, solving (10) can be expensive and challenging.

TABLE 4
Example $\hat{\boldsymbol{r}}$ (Resp. $\check{\boldsymbol{r}}$) for the Two Conditions in Proposition 1

model	$\hat{\boldsymbol{r}}$	$\check{\boldsymbol{r}}$
ComplEx/HolE	[1,2,0,0]	[0,0,3,4]
Analogy	[1,2,3,0]	[0,0,0,4]
SimplE/CP	[1,2,1,2]	$[1,2,-1,-2]$
QuatE	[1,0,0,0]	[0,2,3,4]

3.4 Degenerate and Equivalent Structures

In this section, we introduce properties specific to the proposed search space $\mathcal{A}$. A careful exploitation of these would be key to an efficient search.

3.4.1 Degenerate Structures

Obviously, not all structure matrices in (5) are equally good. For example, if all the nonzero blocks in $g_K(\boldsymbol{A}, \boldsymbol{r})$ are in the first column, $f_{\boldsymbol{A}}$ will be zero for all head embeddings with $\boldsymbol{h}_1 = \boldsymbol{0}$. These structures should be avoided.

Definition 5 (Degenerate structure). *Matrix $\boldsymbol{A}$ is degenerate if (i) there exists $\boldsymbol{h} \neq \boldsymbol{0}$ such that $\boldsymbol{h}^\top g_K(\boldsymbol{A}, \boldsymbol{r})\boldsymbol{t} = 0, \forall \boldsymbol{r}, \boldsymbol{t}$; or (ii) there exists $\boldsymbol{r} \neq \boldsymbol{0}$ such that $\boldsymbol{h}^\top g_K(\boldsymbol{A}, \boldsymbol{r})\boldsymbol{t} = 0, \forall \boldsymbol{h}, \boldsymbol{t}$.*

With a degenerate $\boldsymbol{A}$, the triple (h, r, t) is always non-plausible for every nonzero head embedding $\boldsymbol{h}$ or relation embedding $\boldsymbol{r}$, which limits expressiveness of the scoring function. The following Proposition shows that it is easy to check whether $\boldsymbol{A}$ is degenerate. Its proof is in Appendix A.2.

Proposition 2. *$\boldsymbol{A}$ is not degenerate if and only if $rank(\boldsymbol{A}) = K$ and $\{1, \ldots, K\} \subset \{|A_{ij}| : i, j = 1, \ldots, K\}$.*

Since K is very small (which is equal to 4 here), the above conditions are inexpensive to check. Hence, we can efficiently filter out degenerate $\boldsymbol{A}$'s and avoid wasting time in training and evaluating these structures.

3.4.2 Equivalence

In general, two different $\boldsymbol{A}$'s can have the same performance (as measured by F in Definition 4). This is captured in the following notion of equivalence. If a group of $\boldsymbol{A}$'s are equivalent, we only need to evaluate one of them.

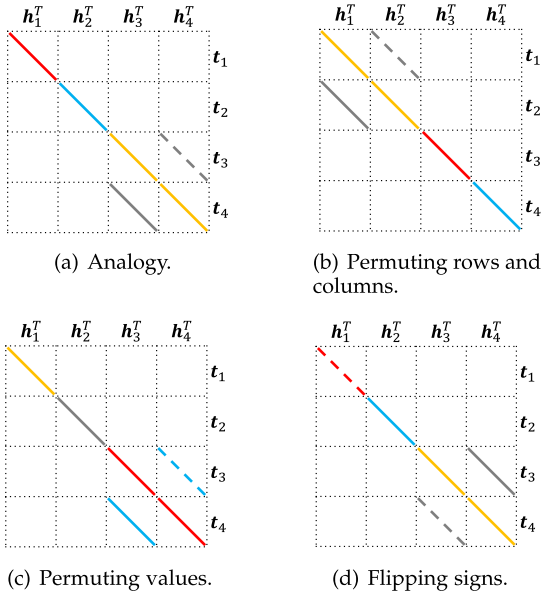
Definition 6 (Equivalence). *Let $\boldsymbol{P}^* = \arg\max_{\boldsymbol{P}} M(F(\boldsymbol{P}; \boldsymbol{A}), \mathcal{S})$ and $\boldsymbol{P}'^* = \arg\max_{\boldsymbol{P}'} M(F(\boldsymbol{P}'; \boldsymbol{A}'), \mathcal{S})$. If $\boldsymbol{A} \neq \boldsymbol{A}'$ but $M(F(\boldsymbol{P}^*; \boldsymbol{A}), \mathcal{S}) = M\big(F(\boldsymbol{P}'^*; \boldsymbol{A}'), \mathcal{S}\big)$ for all $\mathcal{S}$, then $\boldsymbol{A}$ is equivalent to $\boldsymbol{A}'$ (denoted $\boldsymbol{A} \equiv \boldsymbol{A}'$).*

The following Proposition shows several conditions for two structures to be equivalent. Its proof is in Appendix A.3. Examples are shown in Fig. 3.

Proposition 3. *Given an $\boldsymbol{A}$ in (5), construct $\boldsymbol{\Phi}_{\boldsymbol{A}} \in \mathbb{R}^{K \times K^2}$ such that $[\boldsymbol{\Phi}_{\boldsymbol{A}}]_{|A_{ij}|,(i-1)K+j} = sign(A_{ij})$ if $|A_{ij}| \in \{1, \ldots, K\}$, and 0 otherwise.[2] Two structure matrices $\boldsymbol{A}$ and $\boldsymbol{A}'$ are equivalent if any one of the following conditions is satisfied.*

(i) Permuting rows and columns: There exists a permutation matrix $\boldsymbol{\Pi} \in \{0, 1\}^{K \times K}$ such that $\boldsymbol{A}' = \boldsymbol{\Pi}^\top \boldsymbol{A} \boldsymbol{\Pi}$.

2. Intuitively, in $\boldsymbol{\Phi}_{\boldsymbol{A}}$, the indexes of nonzero values in its $|A_{ij}|$-th row indicate positions of elements in $\boldsymbol{A}$ whose absolute values are $|A_{ij}|$.



(a) Analogy. (b) Permuting rows and columns. (c) Permuting values. (d) Flipping signs.

Fig. 3. Illustration of $R_{(r)}$ for Analogy (Fig. 3a) and three example equivalent structures based on the conditions in Proposition 3. Fig. 3b permutes the index [1,2,3,4] of rows and columns in A to [3,4,1,2]; Fig. 3c permutes the values [1,2,3,4] in A to [3,4,1,2]; Fig. 3d flips the signs of values [1,2,3,4] in A to $[-1, 2, -3, 4]$.

(ii) *Permuting values: There exists a permutation matrix* $\mathbf{\Pi} \in \{0,1\}^{K\times K}$ *such that* $\mathbf{\Phi}_{\boldsymbol{A}'} = \mathbf{\Pi}\mathbf{\Phi}_{\boldsymbol{A}}$;

(iii) *Flipping signs: There exists a sign vector* $\boldsymbol{s} \in \{\pm 1\}^K$ *such that* $[\mathbf{\Phi}_{\boldsymbol{A}'}]_{i,\cdot} = s_i \cdot [\mathbf{\Phi}_{\boldsymbol{A}}]_{i,\cdot}, \forall i = 1, \ldots, K$.

There are $K!$ possible permutation matrices for conditions (i) and (ii), and 2^K possible sign vectors for condition (iii). Hence, one has to check a total of $(K!)^2 2^K$ combinations.

4 Search Algorithm

In this section, we design efficient algorithms to search for the structure matrix A in (5). As discussed in Section 2.3.1, the model-based approach requires a proper surrogate model for such a complex space. Thus, we will focus on the sample-based approach, particularly on the progressive algorithm and evolutionary algorithm. To search efficiently, one needs to (i) ensure that each new A is neither degenerate nor equivalent to an already-explored structure; and (ii) the scoring function $f_{\boldsymbol{A}}(h, r, t)$ obtained from the new A is likely to have high performance. These can be achieved by designing an efficient filter (Section 4.1) and performance predictor (Section 4.2). Then, we introduce two search algorithms: progressive search (Section 4.3) and evolutionary algorithm (Section 4.4).

4.1 Filtering Degenerate and Equivalent Structures

Algorithm 1 shows the filtering procedure. First, step 2 removes degenerate structure matrices by using the conditions in Proposition 2. Step 3 then generates a set of $(K!)^2 2^K$ structures that are equivalent to A (Proposition 3). A is filtered out if any of its equivalent structures appears in the set $\mathcal{H}$ containing structure matrices that have already been explored. As K is small, this filtering cost is very low compared with the cost of model training in (10).

Algorithm 1. Filtering degenerate and equivalent structure matrices. The output is "False" if the input structure matrix A is to be filtered out.

Input: $\boldsymbol{A}$: a $K \times K$ structure matrix, $\mathcal{H}$: a set of structures.
1: **initialization:** $\mathcal{Q}(\boldsymbol{A}, \mathcal{H}) = \text{True}$.
2: **if** $\det(\boldsymbol{A}) = 0$ or $\{1, \ldots, K\} \not\subset \{|A_{ij}| : i, j = 1, \ldots, K\}$, **then** $\mathcal{Q}(\boldsymbol{A}, \mathcal{H}) = \text{False}$.
3: generate a set of equivalent structures $\{\boldsymbol{A}' : \boldsymbol{A}' \equiv \boldsymbol{A}\}$ by enumerating permutation matrices $\boldsymbol{P}$'s and sign vectors $\boldsymbol{s}$'s.
4: **for** $\boldsymbol{A}'$ in $\{\boldsymbol{A}' : \boldsymbol{A}' \equiv \boldsymbol{A}\}$ **do**
5: **if** $\boldsymbol{A}' \in \mathcal{H}$, **then** $\mathcal{Q}(\boldsymbol{A}, \mathcal{H}) = \text{False}$, and exit the loop.
6: **end for**
7: **return** $\mathcal{Q}(\boldsymbol{A}, \mathcal{H})$.

4.2 Performance Predictor

After collecting N structures in $\mathcal{H}$, we construct a predictor $\mathcal{P}$ to estimate the goodness of each A. As mentioned in Section 2.3, search efficiency depends heavily on how to evaluate the candidate models.

A highly efficient approach is parameter sharing, as is popularly used in one-shot neural architecture search (NAS) [31], [47]. However, parameter sharing can be problematic when used to predict the performance of scoring functions. Consider the following two A's: (i) A_1 is a 4×4 matrix of all $+1$'s, and so $f_{\boldsymbol{A}_1}(h, r, t) = \sum_{i,j=1}^{4} \langle \boldsymbol{h}_i, \boldsymbol{r}_1, \boldsymbol{t}_j \rangle$, and (ii) A_2 is a 4×4 matrix of all -1's, and so $f_{\boldsymbol{A}_2}(h, r, t) = -\sum_{i,j=1}^{4} \langle \boldsymbol{h}_i, \boldsymbol{r}_1, \boldsymbol{t}_j \rangle = -f_{\boldsymbol{A}_1}(h, r, t)$. When parameter sharing is used, it is likely that the performance predictor will output different scores for A_1 and A_2. However, from Proposition 3, by setting $\boldsymbol{s} = [-1, -1, -1, -1]$ in condition (iii), we have $\boldsymbol{A}_1 \equiv \boldsymbol{A}_2$ and thus they indeed have the same performance. This problem will also be empirically demonstrated in Section 5.1.8. Hence, instead, we train and evaluate the models separately as in the stand-alone NAS evaluation [30], [49].

Algorithm 2. Construction of the symmetry-related feature (SRF) vectors.

Input: structure matrix $\boldsymbol{A}$.
1: **initialization:** $\boldsymbol{\alpha}, \boldsymbol{\beta} := \boldsymbol{0}$.
2: **for** $\boldsymbol{r} \in \mathcal{C}$ **do**
3: **if** $\boldsymbol{r} \neq \boldsymbol{0}$ **then**
4: $x = \big|\{i : r_i = 0\}\big|$;
5: $y = \big|\{j > 0 : r_i = j \text{ or } r_i = -j\}\big|$;
// *for symmetric case*
6: **if** $g_K(\boldsymbol{A}, \boldsymbol{r}) - g_K(\boldsymbol{A}, \boldsymbol{r})^\top = \boldsymbol{0}$ **then** $\alpha_{(x,y)} = 1$;
// *for skew-symmetric case*
7: **if** $g_K(\boldsymbol{A}, \boldsymbol{r}) + g_K(\boldsymbol{A}, \boldsymbol{r})^\top = \boldsymbol{0}$ **then** $\beta_{(x,y)} = 1$;
8: **end if**
9: **end for**
10: **return** $[\text{vec}(\boldsymbol{\alpha}); \text{vec}(\boldsymbol{\beta})]$.

Recall from Section 2.1 that it is desirable for the scoring function to be fully expressive. Proposition 1 shows that this requires looking for a $\hat{\boldsymbol{r}} \in \mathcal{C}$ such that $g_K(\boldsymbol{A}, \hat{\boldsymbol{r}})$ is symmetric and a $\check{\boldsymbol{r}} \in \mathcal{C}$ such that $g_K(\boldsymbol{A}, \check{\boldsymbol{r}})$ is skew-symmetric. This motivates us to examine each of the $(2K+1)^K - 1 \boldsymbol{r}$'s in $\mathcal{C}$ (defined in (8)) and see whether it leads to a symmetric or skew-symmetric $g_K(\boldsymbol{A}, \boldsymbol{r})$. However, directly using all these $(2K+1)^K - 1$ choices as features to a performance predictor can be computationally expensive. Instead, empirically

we find that the following two features can be used to group the scoring functions: (i) number of zeros in $\boldsymbol{r}$: $|\{i \in \{1,\ldots,K\} : r_i = 0\}|$; and (ii) number of nonzero absolute values in $\boldsymbol{r}$: $|\{j > 0 : r_i = j \text{ or } r_i = -j,\ i \in \{1,\ldots,K\}\}|$. The possible choices is reduced to $K(K+1)/2$ (groups of scoring functions). We keep two symmetry-related feature (SRF) as $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. If $g_K(\boldsymbol{A},\boldsymbol{r})$ is symmetric (resp. skew-symmetric) for any $\boldsymbol{r}$ in $\mathcal{C}$, the entry in $\boldsymbol{\alpha}$ (resp. $\boldsymbol{\beta}$) corresponding to $\boldsymbol{r}$ is set to 1. The construction process is also shown in Algorithm 2. Finally, the SRF vector is composed with $\mathrm{vec}(\boldsymbol{\alpha})$ and $\mathrm{vec}(\boldsymbol{\beta})$, which vectorize the values in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and fed as input to a two-layer MLP for performance prediction.

Algorithm 3. Progressive search algorithm (AutoBLM).

Input: I: number of top structures; N: number of generated structures; P: number of structures selected by $\mathcal{P}$; b_0: number of nonzero elements in initial structures; filter $\mathcal{Q}$ and performance predictor $\mathcal{P}$.

1: **initialization:** $b := b_0$, create a candidate set $\mathcal{H}^b = \emptyset$;
2: **for**each $\boldsymbol{A}^{(b)} \in \{\boldsymbol{A}^{(b)}\}$ **do**
3: **if** $\mathcal{Q}(\boldsymbol{A}^{(b)}, \mathcal{H}^b)$ from Algorithm 1 is true **then** $\mathcal{H}^b \leftarrow \mathcal{H}^b \cup \{\boldsymbol{A}^{(b)}\}$;
4: **if** $|\mathcal{H}^b| = I$, break loop;
5: **end for**
6: *train* and *evaluate* all $\boldsymbol{A}^{(b)}$'s in $\mathcal{H}^b$;
7: add $\boldsymbol{A}^{(b)}$'s to $\mathcal{T}^b$ and record the performance in $\mathcal{Y}^b$;
8: update predictor $\mathcal{P}$ with records in $(\mathcal{T}^b, \mathcal{Y}^b)$.
9: **repeat**
10: $b := b + 1$;
11: $\mathcal{H}^b = \emptyset$;
12: **repeat**
13: randomly select a top-I structure $\boldsymbol{A}^{b-1} \in \mathcal{T}^{b-1}$;
14: randomly generate $i_b, j_b, k_b \in \{1,\ldots,K\}$, $s_b \in \{\pm 1\}$, and form $\boldsymbol{A}^{(b)}$ with $f_{\boldsymbol{A}^{(b)}} \leftarrow f_{\boldsymbol{A}^{b-1}} + s_b\langle \boldsymbol{h}_{i_b}, \boldsymbol{r}_{k_b}, \boldsymbol{t}_{j_b}\rangle$;
15: **if** $\mathcal{Q}(\boldsymbol{A}^{(b)}, \mathcal{H}^b \cup \mathcal{T}^b)$ from Algorithm 1 is true **then** $\mathcal{H}^b \leftarrow \mathcal{H}^b \cup \{\boldsymbol{A}^{(b)}\}$;
16: **until**$\left|\mathcal{H}^b\right| = N$
17: select top-P $\boldsymbol{A}^{(b)}$'s in $\mathcal{H}^b$ based on the *predictor* $\mathcal{P}$;
18: *train* embeddings and *evaluate* the performance of $\boldsymbol{A}^{(b)}$'s;
19: add $\boldsymbol{A}^{(b)}$'s in $\mathcal{T}^b$ and record the performance in $\mathcal{Y}^b$;
20: update the predictor (the following commented out) $\mathcal{P}$ with $(\mathcal{T} = \mathcal{T}^{b_0} \cup \dot{s} \cup \mathcal{T}^b, \mathcal{Y} = \mathcal{Y}^{b_0} \cup \dot{s} \cup \mathcal{Y}^b)$;
21: **until**budget is exhausted or $b = K^2$;
22: select the top-I structures in $\mathcal{T}$ based on performance in $\mathcal{Y}$ to form the set $\mathcal{I}$.
23: **return** $\mathcal{I}$.

4.3 Progressive Algorithm

To explore the search space $\mathcal{A}$ in (11), the simplest approach is by direct sampling. However, it can be expensive as the space is large. Note from (4) that the complexity of $f_{\boldsymbol{A}}(h,r,t)$ is controlled by the number of nonzero elements in $\boldsymbol{A}$. Inspired by [49], we propose in this section a progressive algorithm that starts with $\boldsymbol{A}$'s having only a few nonzero elements, and then gradually expands the search space by allowing more nonzeros.

The procedure, which is called AutoBLM, is in Algorithm 3. Let $\boldsymbol{A}^{(b)}$ be an $\boldsymbol{A}$ with b nonzero elements, and the corresponding BLM be $f_{\boldsymbol{A}^{(b)}}$. In step 1, we initialize b to some b_0 and create an empty candidate set $\mathcal{H}^b$. As $\boldsymbol{A}$'s with fewer than K nonzero elements are degenerate (Proposition 2), we use $b_0 = K$. We first sample positions of b_0 nonzero elements, and then randomly assign them values in $\{\pm 1, \pm 2, \ldots, \pm K\}$. The other entries are set to zero.

Steps 2-5 filter away degenerate and equivalent structures. The number of nonzero elements b is then increased by 1 (step 10). For each such b, steps 12-16 greedily select a top-performing structure (evaluated based on the mean reciprocal rank (MRR) [3] performance on $\mathcal{S}_{\mathrm{val}}$) in $\mathcal{T}^{b-1}$, and generate N candidates. All the candidates are checked by the filter $\mathcal{Q}$ (Section 4.1) to avoid degenerate or equivalent solutions. Next, the predictor $\mathcal{P}$ in Section 4.2 selects the top-P $\boldsymbol{A}^{(b)}$'s, which are then trained and evaluated in step 18. The training data for $\mathcal{P}$ is collected with the recorded structures and performance in $(\mathcal{T},\mathcal{Y})$ at step 20. Finally, the top-I structures in $\mathcal{T}$ evaluated by the corresponding performance in $\mathcal{Y}$ are returned.

Algorithm 4. Evolutionary search algorithm. (AutoBLM+).

Input: I: number of top structures; N: number of generated structures; P: number of structures selected by $\mathcal{P}$; b_0: number of nonzero elements in initial structures; filter $\mathcal{Q}$, and performance predictor $\mathcal{P}$.

1: **initialization:** $\mathcal{I} = \emptyset$;
2: **for**each $\boldsymbol{A} \in \{\boldsymbol{A}^{(b_0)}\}$**do**
3: **if** $\mathcal{Q}(\boldsymbol{A}, \mathcal{I})$ from Algorithm 1 is true **then** $\mathcal{I} \leftarrow \mathcal{I} \cup \{\boldsymbol{A}\}$;
4: **if** $|\mathcal{I}| = I$, break loop;
5: **end for**
6: *train* and *evaluate* all $\boldsymbol{A}$'s in $\mathcal{I}$;
7: add $\boldsymbol{A}$'s to $\mathcal{T}$ and record the performance in $\mathcal{Y}$;
8: **repeat**
9: update predictor $\mathcal{P}$ with records in $(\mathcal{T},\mathcal{Y})$.
10: **repeat**
11: $\mathcal{H} = \emptyset$;
12: **mutation:** sample $\boldsymbol{A} \in \mathcal{I}$ and mutate to $\boldsymbol{A}_{\mathrm{new}}$; **or**
13: **crossover:** sample $\boldsymbol{A}_{(a)}, \boldsymbol{A}_{(b)} \in \mathcal{I}$, and use crossover to generate $\boldsymbol{A}_{\mathrm{new}}$;
14: **if** $\mathcal{Q}(\boldsymbol{A}_{\mathrm{new}}, \mathcal{H} \cup \mathcal{T})$ is true by Algorithm 1, **then** $\mathcal{H} \leftarrow \mathcal{H} \cup \{\boldsymbol{A}_{\mathrm{new}}\}$;
15: **until**$|\mathcal{H}| = N$;
16: select top-P structures $\boldsymbol{A}$ in $\mathcal{H}$ based on the the *predictor* $\mathcal{P}$;
17: **for** each top-P structure $\boldsymbol{A}$**do**
18: *train* embeddings and *evaluate* the performance of $\boldsymbol{A}$;
19: **survive:** update $\mathcal{I}$ with $\boldsymbol{A}$ if $\boldsymbol{A}$ is better than the worst structure in $\mathcal{I}$;
20: **end for**
21: add $\boldsymbol{A}$'s in $\mathcal{T}$ and record the performance in $\mathcal{Y}$;
22: **until**budget is exhausted;
23: **return** $\mathcal{I}$.

4.4 Evolutionary Algorithm

While progressive search can be efficient, it may not fully explore the search space and can lead to sub-optimal solutions [51]. The progressive search can only generate structures from fewer non-zero elements to more ones. Thus, it can not visit and adjust the structures with fewer nonzero elements when b is increased. To address these problems, we consider in this section the use of evolutionary algorithms [52].

The procedure, which is called AutoBLM+, is in Algorithm 4. As in Algorithm 3, we start with structures having $b_0 = K$ nonzero elements. Steps 2-7 initializes a set $\mathcal{I}$ of I non-degenerate and non-equivalent structures. The main difference with Algorithm 3 is in steps 9-16, in which new structures are generated by mutation and crossover. For a given structure $\boldsymbol{A}$, mutation changes the value of each entry to another one in $\{0, \pm 1, \ldots, \pm K\}$ with a small probability p_m. For crossover, given two structures $\boldsymbol{A}_{(a)}$ and $\boldsymbol{A}_{(b)}$, each entry of the new structure has equal probabilities to be selected from the corresponding entries in $\boldsymbol{A}_{(a)}$ or $\boldsymbol{A}_{(b)}$. After mutation or crossover, we check if the newly generated $\boldsymbol{A}_{\text{new}}$ has to be filtered out. After N structures are collected, we use the performance predictor $\mathcal{P}$ in Section 4.2 to select the top-P structures. These are then trained and evaluated for actual performance. Finally, structures in $\mathcal{I}$ with performance worse than the newly evaluated ones are replaced (step 19).

5 Experiments

In this section, experiments are performed on a number of KG tasks. Algorithm 5 shows the general procedure for each task. First, we find a good hyper-parameter setting to train and evaluate different structures (steps 2-6). Based on the observation that the performance ranking of scoring functions is consistent across different d's (details are in Appendix C), we set d to a smaller value (64) to reduce model training time. The search algorithm is then used to obtain the set $\mathcal{I}$ of top-I structures (step 8). Finally, the hyper-parameters are fine-tuned with a larger d, and the best structure selected (steps 10-14). Experiments are run on a RTX 2080Ti GPU with 11 GB memory. All algorithms are implemented in python [53].

Algorithm 5. Experimental procedure for each KG task. Here, *HP* denotes the hyper-parameters $\{\eta, \lambda, m, d\}$.

1: // **stage 1**: *configure hyper-parameters for scoring function search*.
2: **for** $i = 1, \ldots, 10$ **do**
3: fix $d = 64$, randomly select $\eta_i \in [0, 1]$, $\lambda_i \in [10^{-5}, 10^{-1}]$ and $m_i \in \{256, 512, 1024\}$;
4: train *SimplE* with $HP_i = \{\eta_i, \lambda_i, m_i, d\}$, and evaluate the validation MRR;
5: **end for**
6: select the best hyper-parameter setting $\bar{HP} \in \{HP_i\}_{i=1}^{10}$;
7: // **stage 2**: *search scoring function*
8: using hyper-parameter setting $\bar{HP}$, obtain the set $\mathcal{I}$ of top-I structures from Algorithm 3 or Algorithm 4;
9: // **stage 3**: *fine-tune the obtained scoring function*
10: **for** $j = 1, \ldots, 50$ ($j = 1, \ldots, 10$ for YAGO3-10) **do**
11: randomly select a structure $\boldsymbol{A}_j \in \mathcal{I}$;
12: randomly select $\eta_j \in [0, 1]$, $\lambda_j \in [10^{-5}, 10^{-1}]$, $m_j \in \{256, 512, 1024\}$, and $d_j \in \{256, 512, 1024, 2048\}$;
13: train the KG learning model with structure $\boldsymbol{A}_j$ and hyper-parameter setting $HP_j = \{\eta_j, \lambda_j, m_j, d_j\}$
14: **end for**
15: select the best structure $\{\boldsymbol{A}^*, HP^*\} \in \{\boldsymbol{A}_j, HP_j\}_{j=1}^{50}$.

5.1 Knowledge Graph (KG) Completion

In this section, we perform experiments on KG completion as introduced in Section 2.2.1. we use the full multi-class log-loss [18], which is more robust and has better performance than negative sampling [18], [33].

TABLE 5
Statistics of the KG Completion Datasets

data set	#entity	#relation	number of samples		
			training	validation	testing
WN18 [10]	40,943	18	141,442	5,000	5,000
FB15k [10]	14,951	1,345	484,142	50,000	59,071
WN18RR [21]	40,943	11	86,835	3,034	3,134
FB15k237 [56]	14,541	237	272,115	17,535	20,466
YAGO3-10 [60]	123,188	37	1,079,040	5,000	5,000
ogbl-biokg	94 k	51	4,763 k	163 k	163 k
ogbl-wikikg2	2500 k	535	16,109 k	429 k	598 k

5.1.1 Setup

Datasets. Experiments are performed on the following popular benchmark datasets: WN18, FB15 k, WN18RR, FB15k237, YAGO3-10, ogbl-biokg and ogbl-wikikg2 (Table 5). WN18 and FB15 k are introduced in [10]. WN18 is a subset of the lexical database WordNet [54], while FB15 k is a subset of the Freebase KG [55] for human knowledge. WN18RR [21] and FB15k237 [56] are obtained by removing the near-duplicates and inverse-duplicate relations from WN18 and FB15 k. YAGO3-10 is created by [21], and is a subset of the semantic KG YAGO [57], which unifies WordNet and Wikipedia. The ogbl-biokg and ogbl-wikikg2 datasets are from the open graph benchmark (OGB) [58], which contains realistic and large-scale datasets for graph learning. The ogbl-biokg dataset is a biological KG describing interactions among proteins, drugs, side effects and functions. The ogbl-wikikg2 dataset is extracted from the Wikidata knowledge base [59] describing relations among entities in Wikipedia.

Baselines. For AutoBLM and AutoBLM+, we select the structure for evaluation from the set returned by Algorithm 3 or 4 based on the MRR performance on the validation set.

For WN18, FB15 k, WN18RR, FB15k237, YAGO3-10, AutoBLM and AutoBLM+ are compared with the following popular human-designed KG embedding models[3]: (i) TDM, including TransH [11], RotatE [13] and PairE [61]; (ii) NNM, including ConvE [21], RSN [22] and CompGCN [24]; (iii) BLM, including TuckER [35], Quat [19], DistMult [7], ComplEx [14], HolE [15], Analogy [16] SimplE [17], and CP [18]. We do not compare with NASE [62] as its code is not publicly available.

For ogbl-biokg and ogbl-wikikg2 [58], we compare with the models reported in the OGB leaderboard[4], namely, TransE [10], RotatE, PairE, DistMult, and ComplEx.

Performance Measures. The learned $f_{\boldsymbol{A}}(h, r, t)$ is evaluated in the context of link prediction. Following [3], [7], [14], [16], [17], [21], for each triple (h, r, t), we first take $(?, r, t)$ as the query and obtain the filtered rank on the head

$$\text{rank}_h = \left| \left\{ e \in \mathcal{E} : \begin{array}{c} (f(e,r,t) \geq f(h,r,t)) \wedge \\ ((e,r,t) \notin \mathcal{S}_{\text{tra}} \cup \mathcal{S}_{\text{val}} \cup \mathcal{S}_{\text{tst}}) \end{array} \right\} \right| + 1, \tag{12}$$

3. Obtained from https://github.com/thunlp/OpenKE and https://github.com/Sujit-O/pykg2vec
4. https://ogb.stanford.edu/docs/leader_linkprop/

TABLE 6
Testing Performance of MRR, H@1 and H@10 on KG Completion. The Best Model is Highlighted in Bold and the Second Best is Underlined. "–" Means That Results are Not Reported in Those Papers or Their Code on That data/metric is Not Available. CompGCN Uses the Entire KG in Each Iteration and So Runs Out of Memory on the Larger Data Sets of WN18, FB15 k and YAGO3-10

		WN18			FB15 k			WN18RR			FB15k237			YAGO3-10		
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)	TransH	0.521	—	94.5	0.452	—	76.6	0.186	—	45.1	0.233	—	40.1	—	—	—
	RotatE	0.949	94.4	95.9	0.797	74.6	88.4	0.476	42.8	57.1	0.338	24.1	53.3	0.488	39.6	66.3
	PairE	—	—	—	0.811	76.5	89.6	—	—	—	0.351	25.6	54.4	—	—	—
(NNM)	ConvE	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.
	RSN	0.94	92.2	95.3	—	—	—	—	—	—	0.28	20.2	45.3	—	—	—
	Interstellar	—	—	—	—	—	—	0.48	44.0	54.8	0.32	23.3	50.8	0.51	42.4	66.4
	CompGCN	—	—	—	—	—	—	0.479	44.3	54.6	0.355	26.4	53.5	—	—	—
(BLM)	TuckER	**0.953**	**94.9**	95.8	0.795	74.1	89.2	0.470	44.3	52.6	0.358	26.6	54.4	—	—	—
	DistMult	0.821	71.7	95.2	0.775	71.4	87.2	0.443	40.4	50.7	0.352	25.9	54.6	0.552	47.1	68.9
	SimplE/CP	0.950	94.5	95.9	0.826	79.4	90.1	0.462	42.4	55.1	0.350	26.0	54.4	0.565	49.1	71.0
	HolE/ComplEx	0.951	94.5	95.7	0.831	79.6	90.5	0.471	43.0	55.1	0.345	25.3	54.1	0.563	49.0	70.7
	Analogy	0.950	94.6	95.7	0.816	78.0	89.8	0.467	42.9	55.4	0.348	25.6	54.7	0.557	48.5	70.4
	QuatE	0.950	94.5	95.9	0.782	71.1	90.0	0.488	43.8	**58.2**	0.348	24.8	55.0	0.556	47.4	70.4
AutoBLM		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**
AutoBLM+		0.952	94.7	**96.1**	**0.861**	**83.2**	**91.3**	**0.492**	**45.2**	56.7	**0.364**	**27.0**	**55.3**	**0.577**	**50.2**	**71.5**

where $\mathcal{S}_{\rm tra}, \mathcal{S}_{\rm val}, \mathcal{S}_{\rm tst}$ are the training, validation, and test sets, respectively. Next we take $(h, r, ?)$ as the query and obtain the filtered rank on the tail

$$\text{rank}_t = \left| \left\{ e \in \mathcal{E} : \begin{array}{c} (f(h,r,e) \geq f(h,r,t)) \wedge \\ ((h,r,e) \notin \mathcal{S}_{\rm tra} \cup \mathcal{S}_{\rm val} \cup \mathcal{S}_{\rm tst}) \end{array} \right\} \right| + 1. \tag{13}$$

The following metrics are computed from both the head and tail ranks on all triples: (i) Mean reciprocal ranking (MRR):

$$\text{MRR} = \frac{1}{2|\mathcal{S}|} \sum_{(h,r,t) \in \mathcal{S}} \left(\frac{1}{\text{rank}_h} + \frac{1}{\text{rank}_t} \right);$$

and (ii) H@k: ratio of ranks no larger than k, i.e.,

$$\text{H@}k = \frac{1}{2|\mathcal{S}|} \sum_{(h,r,t) \in \mathcal{S}} (\mathbb{I}(\text{rank}_h \leq k) + \mathbb{I}(\text{rank}_t \leq k)),$$

where $\mathbb{I}(a) = 1$ if a is true, otherwise 0. The larger the MRR or H@k, the better is the embedding. Other metrics for the completion task [63], [64] can also be adopted here.

For ogbl-biokg and ogbl-wikikg2 [58], we only use the MRR as the H@k is not reported by the baselines in the OGB leaderboard.

Hyper-parameters. The search algorithms have the following hyper-parameters: (i) N: number of candidates generated after filtering; (ii) P: number of scoring functions selected by the predictor; (iii) I: number of top structures selected in Algorithm 3 (step 13), or the number of structures survived in $\mathcal{I}$ in Algorithm 4; and (iv) b_0: number of nonzero elements in the initial set. Unless otherwise specified, we use $N = 128$, $P = 8$, $I = 8$ and $b_0 = K$. For the evolutionary algorithm, the mutation and crossover operations are selected with equal probabilities. When mutation is selected, the value of each entry has a mutation probability of $p_m = 2/K^2$. A budget is used to terminate the algorithm. This is set to 256 structures on WN18, FB15 k, WN18RR, FB15k-237, 128 on YAGO3-10, 64 on ogbl-biokg, and 32 on ogbl-wikikg2.

We follow [14], [18] to use Adagrad [65] as optimizer. The Adagrad hyper-parameters are selected from the following ranges: learning rate η in $[0, 1]$, ℓ_2-penalty λ in $[10^{-5}, 10^{-1}]$, batch size m in $\{256, 512, 1024\}$, and dimension d in $\{64, 256, 512, 1024, 2048\}$.

5.1.2 Results on WN18, FB15k, WN18RR, FB15k237, YAGO3-10

Performance. Table 6 shows the testing results on WN18, FB15 k, WN18RR, FB15k237, and YAGO3-10. As can be seen, there is no clear winner among the baselines. On the other hand, AutoBLM performs consistently well. It outperforms the baselines on FB15 k, WN18RR, FB15k237 and YAGO3-10, and is the first runner-up on WN18. AutoBLM+ further improves AutoBLM on FB15 k, WN18RR, FB15k237 and YAGO3-10.

Learning curves. Fig. 4 shows the learning curves of representative models in each type of scoring functions, including: RotatE in TDM; ConvE and CompGCN in NNM; and DistMult, SimplE/CP, ComplEx/HolE, Analogy, QuatE and the proposed AutoBLM/AutoBLM+ in BLM. As can be seen, NNMs are much slower and inferior than BLMs. On the other hand, AutoBLM+ has better performance and comparable time as the other BLMs.

Data-dependent BLM structure. Fig. 5 shows the BLMs obtained by AutoBLM and AutoBLM+. As can be seen, they are different from the human-designed BLMs in Fig. 1 and are also different from each other. To demonstrate that these data-dependent structures also have different accuracies on the same dataset, we take the BLM obtained by AutoBLM (or AutoBLM+) on a source dataset and then evaluate it on a different target dataset. Table 7 shows the testing MRRs obtained (the trends for H@1 and H@10 are similar). As can be seen, the different BLMs perform differently on the same dataset, again confirming the need for data-dependent structures.

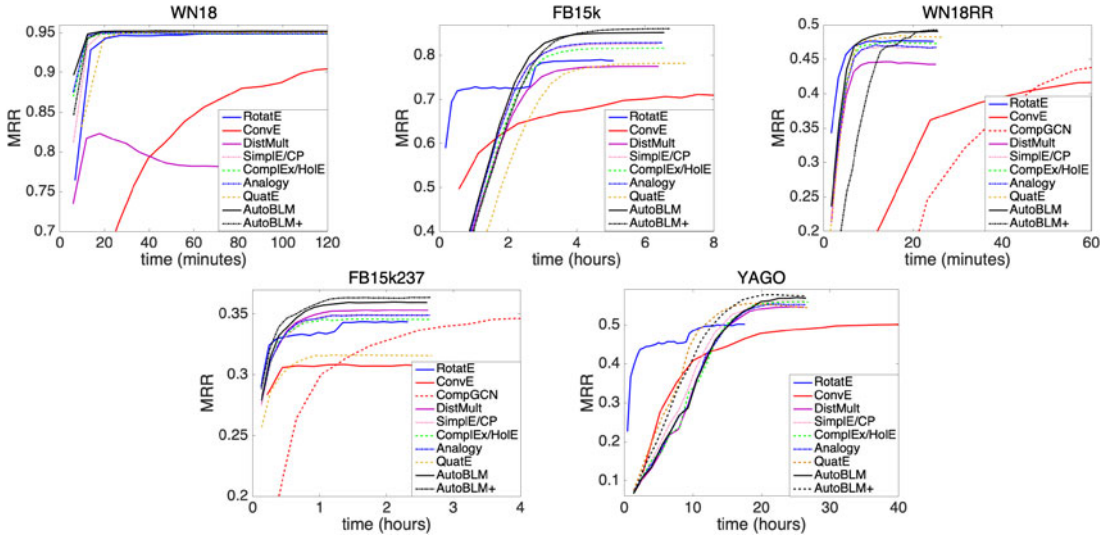
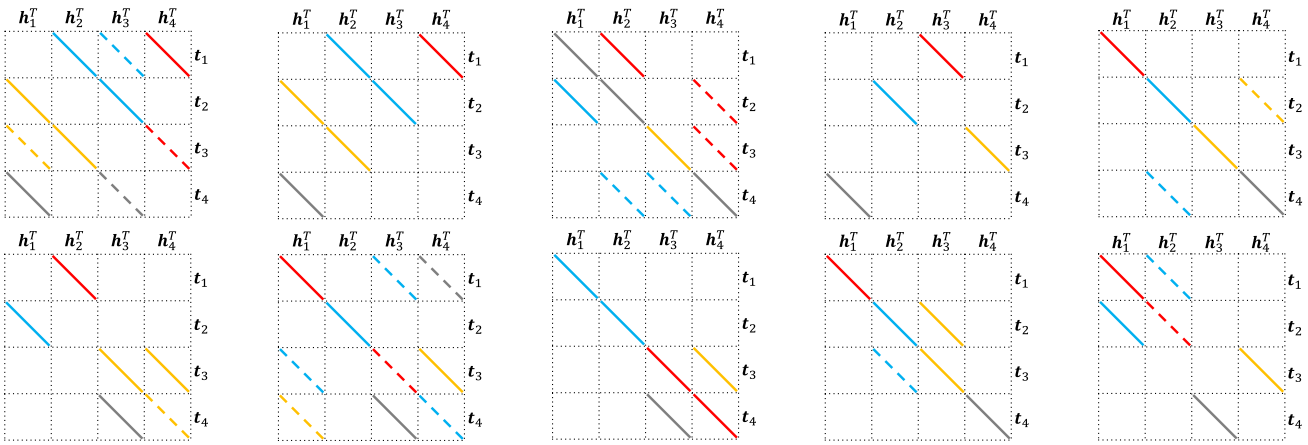


Fig. 4. Convergence of the testing MRR versus running time on the KG completion task.



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

Fig. 5. Graphical illustration of the BLMs obtained by AutoBLM (top) and AutoBLM+ (bottom) on the KG completion task (Section 5.1.2). Different colors correspond to different parts of $[r_1$(red), r_2(blue), r_3(yellow), r_4(gray)]. Solid lines mean positive values, while dashed lines mean negative values. The empty parts have value zero.

5.1.3 Results on Ogbl-Biokg and ogbl-Wikikg2

Table 8 shows the testing MRRs of the baselines (as reported in the OGB leaderboard), the BLMs obtained by AutoBLM and AutoBLM+. As can be seen, AutoBLM and AutoBLM+ achieve significant gains on the testing MRR on both datasets, even though fewer model parameters are needed for AutoBLM+. The searched structures are provided in Figure 13 in Appendix D.

5.1.4 Ablation Study 1: Search Algorithm Selection

First, we study the following search algorithm choices.

(i) Random, which samples each element of $\boldsymbol{A}$ independently and uniformly from $\{0, \pm 1, \ldots, \pm K\}$;
(ii) Bayes, which selects each element of $\boldsymbol{A}$ from $\{0, \pm 1, \ldots, \pm K\}$ by performing hyperparameter optimization using the Tree Parzen estimator [66] and Gaussian mixture model (GMM);
(iii) Reinforce, which generates the K^2 elements in $\boldsymbol{A}$ by using a LSTM [41] recurrently as in NAS-Net [30]. The LSTM is optimized with REINFORCE [67];
(iv) AutoBLM (no Filter, no Predictor, b_0=1) with initial $b_0 = 1$;
(v) AutoBLM+ (no Filter, no Predictor, b_0=1) with initial $b_0 = 1$.

For a fair comparison, we do not use the filter and performance predictor in the proposed AutoBLM and AutoBLM+ here. All structures selected by each of the above algorithms are trained and evaluated with the same hyper-parameter settings in step 6 of Algorithm 5. Each algorithm evaluates a total of 256 structures.

Fig. 6 shows the mean validation MRR of the top $I = 8$ structures w.r.t. clock time during the search process. As can be seen, AutoBLM (no Filter, no Predictor, b_0=1) and AutoBLM+ (no Filter, no Predictor, b_0=1) outperform the rest at the later stages. They have poor initial performance as they start with structures having few nonzero elements, which can be degenerate. This will be further demonstrated in the next section.

5.1.5 Ablation Study 2: Effectiveness of the Filter

Structures with more nonzero elements are more likely to satisfy the two conditions in Proposition 2, and thus less

TABLE 7
Testing MRR on Applying the BLMs Obtained From a Source Dataset (row) to a Target Dataset (column). Bold Numbers Indicate the Best Performance Each Dataset for the Models Searched by AutoBLM and AutoBLM+ Respectively

		WN18	FB15 k	WN18RR	FB15k237	YAGO3-10
AutoBLM	WN18	**0.952**	0.841	0.473	0.349	0.561
	FB15 k	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**
AutoBLM+	WN18	**0.952**	0.848	0.482	0.350	0.564
	FB15 k	0.951	**0.861**	0.479	0.352	0.563
	WN18RR	0.947	0.841	**0.492**	0.347	0.551
	FB15k237	0.860	0.821	0.463	**0.364**	0.546
	YAGO3-10	0.951	0.833	0.469	0.345	**0.577**

TABLE 8
Testing MRR and Number of Parameters on Ogbl-Biokg and ogbl-Wikikg2. the Best Performance is Indicated in Boldface

	ogbl-biokg		ogbl-wikikg2	
model	MRR	# params	MRR	# params
TransE	0.745	188 M	0.426	1251 M
RotatE	0.799	188 M	0.433	1250 M
PairE	0.816	188 M	0.521	500 M
DistMult	0.804	188 M	0.373	1250 M
ComplEx	0.810	188 M	0.403	1250 M
AutoBLM	0.828	188 M	0.532	500 M
AutoBLM+	**0.831**	94 M	**0.546**	500 M

likely to be degenerate. Hence, the filter is expected to be particularly useful when there are few nonzero elements in the structure. In this experiment, we demonstrate this by comparing AutoBLM/AutoBLM+ with and without the filter. The performance predictor is always enabled.

Fig. 7 shows the mean validation MRR of the top $I = 8$ structures w.r.t. clock time. As expected, when the filter is not used, using a larger b_0 will be more likely to have non-degenerate structures and thus better performance, especially at the initial stages. When the filter is used, the performance of both b_0 settings are improved. In particular, with $b_0 = 4$, the initial search space is simpler and leads to better performance.

5.1.6 Ablation Study 3: Performance Predictor

In this experiment, we compare the following AutoBLM/AutoBLM+ variants: (i) AutoBLM (no-predictor) and AutoBLM+ (no-predictor), which simply randomly select P structures for evaluation (in step 17 of Algorithm 3 and step 16 of Algorithm 4, respectively); (ii) AutoBLM (Predictor +SRF) and AutoBLM+ (Predictor+SRF), using the proposed SRF (in Section 4.2) as input features to the performance predictor; and (iii) AutoBLM (Predictor+1hot) and AutoBLM+ (Predictor+1hot), which map each of the K^2 entries in $\boldsymbol{A}$ (with values in $\{0, \pm 1, \ldots, \pm K\}$) to a simple $(2K + 1)$-dimensional one-hot vector, and then use these as features to the performance predictor. The resultant feature vector is thus $K^2(2K + 1)$-dimensional, which is much longer than the $K(K + 1)$-dimensional SRF representation.

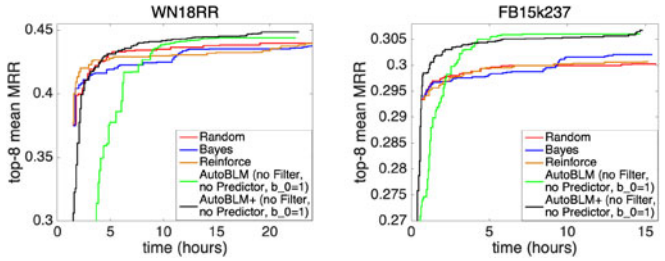


Fig. 6. Comparison of different search algorithms.

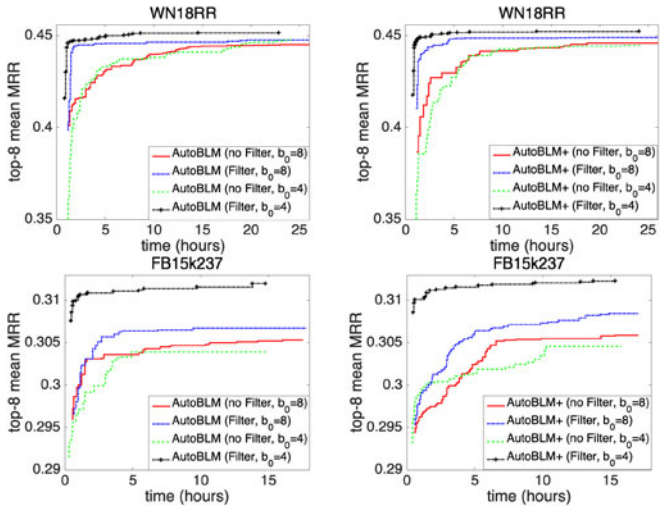


Fig. 7. Comparison of the effect of filter.

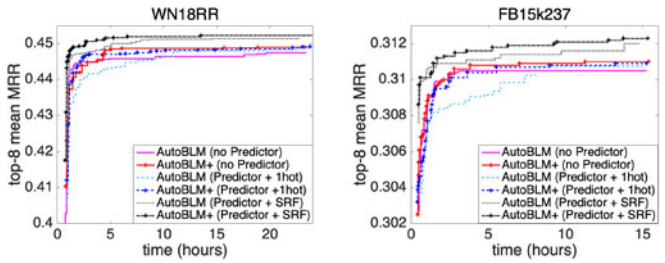


Fig. 8. Effectiveness of the performance predictor.

Fig. 8 shows the mean validation MRR of the top $I = 8$ structures w.r.t. clock time. As can be seen, the use of performance predictor improves the results over AutoBLM (no-Predictor) and AutoBLM+ (no-Predictor). The SRF features also perform better than the one-hot features, as the one-hot features are higher-dimensional and more difficult to learn. Besides, we observe that AutoBLM+ performs better than AutoBLM, as it can more flexibly explore the search space. Thus, in the remaining ablation studies, we will only focus on AutoBLM+.

5.1.7 Ablation Study 4: Varying K

As K increases, the search space, which has a size of $(2K + 1)^{K^2}$ (Section 3.3), increases dramatically. Moreover, the SRF also needs to enumerate a lot more (K^{2K+1}) vectors in $\mathcal{C}$. In this experiment, we demonstrate the dependence on K by running AutoBLM+ with $K = 3, 4, 5$. To ensure that d is divisible by K, we set $d = 60$. Fig. 9 shows the top-8 mean MRR performance on the validation set of the searched models versus clock time. As can be seen, the best performance

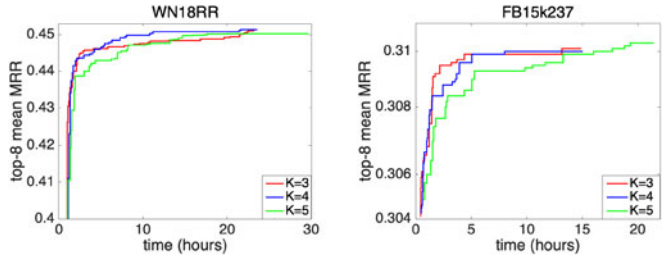


Fig. 9. Comparison of different K values.

attained by different K's are similar. However, $K = 5$ runs slower.

Table 9 shows the running time of the filter, performance predictor (with SRF features), training and evaluation in Algorithm 4 with different K's. As can be seen, the costs of filter and performance predictor increase a lot with K, while the model training and evaluation time are relatively stable for different K's.

TABLE 9
Running Time (In Minutes) of Different Components in Algorithm 4

dataset	K	filter	performance predictor	train	evaluate
WN18RR	3	0.04	1	1217	152
	4	1.4	23	1231	156
	5	90	276	1252	161
FB15k237	3	0.04	1	714	178
	4	1.5	22	721	181
	5	91	283	728	186

5.1.8 Ablation Study 5: Analysis of Parameter Sharing

As mentioned in Section 4.2, parameter sharing may not reliably predict the model performance. To demonstrate this, we empirically compare the parameter-sharing approach, which shares parameter $\boldsymbol{P} = \{\boldsymbol{E}, \boldsymbol{R}\}$ (where $\boldsymbol{E} \in \mathbb{R}^{d \times |\mathcal{E}|}$ is the entity embedding matrix and $\boldsymbol{R} \in \mathbb{R}^{d \times |\mathcal{R}|}$ is the relation embedding matrix in Section 2.1) and the stand-alone approach, which trains each model separately. For parameter sharing, we randomly sample a $\boldsymbol{A}$ in each training iteration from the set of top candidate structures ($\mathcal{H}^b$ in Algorithm 3 or $\mathcal{H}$ in Algorithm 4), and then update parameter $\boldsymbol{P}$. After one training epoch, the sampled structures are evaluated. After 500 training epochs, the top-100 $\boldsymbol{A}$'s are output. For the stand-alone approach, the 100 $\boldsymbol{A}$'s are separately trained and evaluated.

Fig. 10 shows the MRR estimated by parameter-sharing versus the true MRR obtained by individual model training. As can be seen, structures that have high estimated MRRs (by parameter sharing) do not truly have high MRRs. Indeed, the Spearman's rank correlation coefficient[5] between the two sets of MRRs is negative (-0.2686 on WN18RR and -0.2451 on FB15k237). This demonstrates that the one-shot approach, though faster, cannot find good structures.

5.2 Multi-Hop Query

In this section, we perform experiment on multi-hop query as introduced in Section 2.2.2. The entity and relation embeddings are optimized by maximizing the scores on positive queries and minimizing the scores on negative queries, which are generated by replacing e_L with an incorrect entity. On evaluation, we rank the scores of queries $(e_0, r_1 \circ r_2 \circ \dot{s} \circ r_L, e_L)$ of all $e_L \in \mathcal{E}$ to obtain the ranking of ground truth entities.

5.2.1 Setup

Following [4], we use the FB15 k and FB15k237 datasets in Table 5. Evaluation is based on two-hop (2p) and three-hop (3p) queries. Interested readers are referred to [4] for a more detailed description on query generation. For FB15 k, there are 273,710 queries in the training set, 8,000 non-overlapping queries in the validation and testing sets. For FB15k237, there are 143,689 training queries, and 5,000 queries for validation and testing. The setting of the search algorithms' hyper-parameters are the same as in Section 5.1. For the learning hyper-parameters, we search the dimension $d \in \{32, 64\}$, and the other hyper-parameters are the same as those in Section 5.1. We use the MRR performance on the validation set to search for structures as well as hyper-parameters. For performance evaluation, we follow [4], [8], and use the testing Hit@3 and MRR.

We compare with the following baselines: (i) TransE-Comp [39] (based on TransE); (ii) Diag-Comp [39] (based on DistMult); (iii) GQE [8], which uses a $d \times d$ trainable matrix $\boldsymbol{R}_{(r)}$ for composition, and can be regarded as a composition based on RESCAL [6]; and (iv) Q2B [4], which is a recently proposed box embedding method.

5.2.2 Results

Results are shown in Table 10. As can be seen, among the baselines, TransE-Comp, Diag-Comp and GQE are inferior to Q2B. This shows that the general scoring functions cannot be directly applied to model the complex interactions in multi-hop queries. On the other hand, AutoBLM and AutoBLM+ have better performance as they can adapt to the different tasks with different matrices $g_K(\boldsymbol{A}, \boldsymbol{r})$. The obtained structures can be found in Appendix D.

5.3 Entity Classification

In this section, we perform experiment on entity classification as introduced in Section 2.2.3.

5.3.1 Setup

After aggregation for L layers, representation $\boldsymbol{e}^L$ at the last layer is transformed by a multi-layer perception (MLP) to

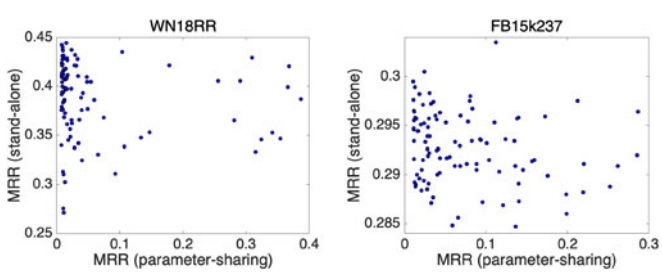


Fig. 10. MRRs of structures as estimated by the parameter-sharing approach and stand-alone approach.

5. https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

TABLE 10
Testing Performance of H@3 and MRR on Multi-Hop Query Task. Results of ⋆'s are Copied from [4]

	FB15 K				FB15K237			
	2p		3p		2p		3p	
	H@3	MRR	H@3	MRR	H@3	MRR	H@3	MRR
TransE-Comp [39]	27.3	. 264	15.8	. 153	19.4	. 177	14.0	. 134
Diag-Comp [39]	32.2	. 309	27.5	. 266	19.1	. 187	15.5	. 147
GQE [8]⋆	34.6	. 320	25.0	. 222	21.3	. 193	15.5	. 145
Q2B [4]⋆	41.3	. 373	30.3	. 274	24.0	. 225	18.6	. 173
AutoBLM	41.5	. 402	29.1	. 283	23.6	. 232	18.2	. 180
AutoBLM+	**43.2**	**. 415**	**30.7**	**. 293**	**24.9**	**. 248**	**19.9**	**. 196**

$\boldsymbol{e}_i^o = MLP(\boldsymbol{e}_i^L) \in \mathbb{R}^C$, where d is the intermediate layer dimension, and is the number of classes. The parameters, including embeddings of entities, relations, $\boldsymbol{W}_0^\ell$, $\boldsymbol{W}^\ell$'s and the MLP, are optimized by minimizing the cross-entropy loss on the labeled entities: $\mathcal{L} = -\sum_{i\in\mathcal{B}}\sum_{c=1}^{C} y_{ic} \ln e_{ic}^o$, where $\mathcal{B}$ is the set of labeled entities, $y_{ic} \in \{0, 1\}$ indicates whether the ith entity belongs to class c, and e_{ic}^o is the cth dimension of $\boldsymbol{e}_i^o$.

Three graph datasets are used (Table 11): AIFB, an affiliation graph; MUTAG, a bioinformatics graph; and BGS, a geological graph. More details can be found in [68]. All entities do not have attributes. The entities' and relations' trainable embeddings are used as input to the GCN.

The following five models are compared: (i) GCN [9], with $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{e}_j^\ell$, does not leverage relations of the edges; (ii) R-GCN [23], with $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{R}_{(r)}^\ell \boldsymbol{e}_j^\ell$; (iii) CompGCN [24] with $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = \boldsymbol{e}_j^\ell$ (-/*/⋆) $\boldsymbol{r}^\ell$, in which the operator (subtraction/multiplication/circular correlation as discussed in Section 2.2.3) is chosen based on 5-fold cross-validation; (iv) AutoBLM; and (v) AutoBLM+. oth AutoBLM and AutoBLM + use the searched structure $\boldsymbol{A}$ to form $\phi(\boldsymbol{e}_j^\ell, \boldsymbol{r}^\ell) = g_K(\boldsymbol{A}, \boldsymbol{r}^\ell)\boldsymbol{e}_j^\ell$.

Setting of the hyper-parameters are the same as in Section 5.1. As for the learning hyper-parameters, we search the embedding dimension d from $\{12, 20, 32, 48\}$, learning rate from $[10^{-5}, 10^{-1}]$ with Adam as the optimizer [69]. For the GCN structure, the hidden size is the same as the embedding dimension, the dropout rate for each layer is from [0,0.5]. We search for 50 hyper-parameter settings for each dataset based on the 5-fold classification accuracy.

For performance evaluation, we use the testing accuracy. Each model runs 5 times, and then the average testing accuracy reported.

5.3.2 Results

Table 12 shows the average testing accuracies. Among the baselines, R-GCN is slightly better than CompGCN on the AIFB dataset, but worse on the other two sparser datasets.

TABLE 11
Data Sets Used in Entity Classification. Sparsity is Computed as $\#\text{edges}/(\#\text{entity}^2 \cdot \#\text{relation})$

dataset	#entity	#relation	#edges	#train	#test	#classes	sparsity
AIFB	8,285	45	29,043	140	36	4	9.4e-6
MUTAG	23,644	23	74,227	272	68	2	5.7e-6
BGS	333,845	103	916,199	117	29	2	8.0e-8

TABLE 12
Classification Accuracies (In %) on Entity Classification Task. Values Marked "*" are Copied from [24]

dataset	AIFB	MUTAG	BGS
GCN	86.67	68.83	73.79
R-GCN	92.78	74.12	82.97
CompGCN	90.6*	85.3*	84.14
AutoBLM	95.55	85.00	84.83
AutoBLM+	**96.66**	**85.88**	**86.17**

By searching the composition operators, AutoBLM and AutoBLM+ outperform all the baseline methods. AutoBLM + is better than AutoBLM since it can find better structures with the same budget by the evolutionary algorithm. The structures obtained are in Appendix D.

6 Conclusion

In this paper, we propose AutoBLM and AutoBLM+, the algorithms to automatically design and discover better scoring functions for KG learning. By analyzing the limitations of existing scoring functions, we setup the problem as searching relational matrix for BLMs. In AutoBLM, we use a progressive search algorithm which is enhanced by a filter and a predictor with domain-specific knowledge, to search in such a space. Due to the limitation of progressive search, we further design an evolutionary algorithm, enhanced by the same filter and predictor, called AutoBLM+. AutoBLM and AutoBLM+ can efficiently design scoring functions that outperform existing ones on tasks including KG completion, multi-hop query and entity classification from the large search space. Comparing AutoBLM with AutoBLM+, AutoBLM+ can design better scoring functions with the same budget.

Acknowledgments

The code is public available at https://github.com/AutoML-Research/AutoSF.

References

[1] A. Singhal, "Introducing the knowledge graph: Things, not strings," *Official Google blog*, vol. 48, no. 1–4, p. 2, 2012.
[2] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proc. IEEE*, vol. 104, no. 1, pp. 11–33, 2016.
[3] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017.
[4] H. Ren, W. Hu, and J. Leskovec, "Query2box: Reasoning over knowledge graphs in vector space using box embeddings," in *Proc. Int. Conf. Learn. Representations*, 2020.
[5] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 353–362.
[6] M. Nickel, V. Tresp, and H. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proc. Int. Conf. Mach. Learn.*, 2011, vol. 11, pp. 809–816.
[7] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," in *Proc. Int. Conf. Learn. Representations*, 2015.
[8] W. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec, "Embedding logical queries on knowledge graphs," in *Proc. Neural Inf. Process. Syst.*, 2018, pp. 2026–2037.
[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[10] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Proc. Neural Inf. Process. Syst.*, 2013, pp. 2787–2795.

[11] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. AAAI Conf. Artif. Intell.*, 2014, vol. 14, pp. 1112–1119.

[12] M. Fan, Q. Zhou, E. Chang, and T. F. Zheng, "Transition-based knowledge graph embedding with relational mapping properties," in *Proc. 28th Pacific Asia Conf. Lang., Inf. Comput.*, 2014, pp. 328–337.

[13] Z. Sun, Z. Deng, J. Nie, and J. Tang, "RotatE: Knowledge graph embedding by relational rotation in complex space," in *Proc. Int. Conf. Learn. Representations*, 2019.

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

[15] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1955–1961.

[16] H. Liu, Y. Wu, and Y. Yang, "Analogical inference for multi-relational embeddings," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2168–2178.

[17] M. Kazemi and D. Poole, "SimplE embedding for link prediction in knowledge graphs," in *Neural Inf. Process. Syst.*, 2018, pp. 4289–4300.

[18] T. Lacroix, N. Usunier, and G. Obozinski, "Canonical tensor decomposition for knowledge base completion," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2863–2872.

[19] S. Zhang, Y. Tay, L. Yao, and Q. Liu, "Quaternion knowledge graph embedding," *Neural Inf. Process. Syst.*, 2019, pp. 2735–2745.

[20] X. Dong *et al.*, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proc. 20th ACM SIGKDD Int. Conf. Know. Discov. Data Mining*, 2014, pp. 601–610.

[21] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2D knowledge graph embeddings," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1811–1818.

[22] L. Guo, Z. Sun, and W. Hu, "Learning to exploit long-term relational dependencies in knowledge graphs," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2505–2514.

[23] M. Schlichtkrull *et al.*, "Modeling relational data with graph convolutional networks," in *Proc. Euro. Semantic web Conf.*, 2018, pp. 593–607.

[24] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, "Composition-based multi-relational graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2020.

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

[26] Y. Wang, R. Daniel, G. Rainer, B. Samuel, and M. Christian, "On evaluating embedding models for knowledge base completion," in *Proc. 4th Workshop Representation Learn. NLP*, 2019, pp. 104–112.

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

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

[29] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970.

[30] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017.

[31] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.

[32] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *JMLR*, vol. 20, no. 55, pp. 1–21, 2019.

[33] Y. Zhang, Q. Yao, W. Dai, and L. Chen, "AutoSF: Searching scoring functions for knowledge graph embedding," in *Proc. 36th Int. Conf. Data Eng.*, 2020, pp. 433–444.

[34] Y. Wang, R. Gemulla, and H. Li, "On multi-relational link prediction with bilinear models," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 4227–4234.

[35] I. Balažević, C. Allen, and T. M. Hospedales, "Tucker: Tensor factorization for knowledge graph completion," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2019, pp. 5185–5194.

[36] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 2, pp. 494–514, Feb. 2022.

[37] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[38] R. Socher, D. Chen, C. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Proc. Neural Inf. Process. Syst.*, 2013, pp. 926–934.

[39] K. Guu, J. Miller, and P. Liang, "Traversing knowledge graphs in vector space," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 318–327.

[40] Y. Zhang, Q. Yao, and L. Chen, "Interstellar: Searching recurrent architecture for knowledge graph embedding," *Neural Inf. Process. Syst.*, vol. 33, pp. 10030–10040, 2020.

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

[42] K. Hayashi and M. Shimbo, "On the equivalence of holographic and complex embeddings for link prediction," in *Proc. Assoc. Comput. Linguistics*, 2017, vol. 2, pp. 554–559.

[43] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[44] J. Gilmer, S. S. Schoenholz, P. F. Riley, R. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

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

[46] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 549–558.

[47] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.

[48] Q. Yao, J. Xu, W. Tu, and Z. Zhu, "Efficient neural architecture search via proximal iterations," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 6664–6671.

[49] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. IEEE Eur. Conf. Comput. Vis.*, 2018, pp. 19–34.

[50] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, pp. 4780–4789.

[51] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2231–2242, Oct. 2004.

[52] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. London, U.K.: Oxford Univ. Press, 1996.

[53] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. Int. Conf. Learn. Representations*, 2017.

[54] G. A. Miller, "WordNet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[55] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1247–1250.

[56] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proc. 3rd Workshop Continuous Vector Space Models Compositionality*, 2015, pp. 57–66.

[57] F. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 697–706.

[58] W. Hu *et al.*, "Open graph benchmark: Datasets for machine learning on graphs," *Neural Inf. Process. Syst.*, vol. 33, pp. 22 118–22 133, 2020.

[59] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[60] F. Mahdisoltani, J. Biega, and F. M. Suchanek, "Yago3: A knowledge base from multilingual wikipedias," in *Proc. 7th Biennial Conf. Innov. Data Syst. Res.*, 2013.

[61] L. Chao, J. He, T. Wang, and W. Chu, "Pairre: Knowledge graph embeddings via paired relation vectors," in *Proc. Assoc. Comput. Linguistics*, 2021, pp. 4360–4369.

[62] X. Kou, B. Luo, H. Hu, and Y. Zhang, "Nase: Learning knowledge graph embedding for link prediction via neural architecture search," in *Proc. 29th ACM Int. Conf. Inf. Know. Manage.*, 2020, pp. 2089–2092.

[63] Y. Wang, D. Ruffinelli, R. Gemulla, S. Broscheit, and C. Meilicke, "On evaluating embedding models for knowledge base completion," in *Proc. RepL4NLP-2019*, 2019, pp. 104–112.

[64] P. Tabacof and L. Costabello, "Probability calibration for knowledge graph embedding models," in *Proc. Int. Conf. Learn. Representations*, 2019.

[65] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.
[66] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.
[67] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn. J.*, vol. 8, no. 3-4, pp. 229–256, 1992.
[68] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *Proc. Int. Semantic Web Conf.*, 2016, pp. 498–514.
[69] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.
[70] R. A. Horn and C. R. Johnson, *Matrix Analysis*bibpunc. Cambridge, U.K.: Cambridge Univ. Press, 2012.
[71] A. Rossi, D. Firmani, A. Matinata, P. Merialdo, and D. Barbosa, "Knowledge graph embedding for link prediction: A comparative analysis," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 2, pp. 1–49, 2021.

Yongqi Zhang (Member, IEEE) received the bachelor's degree from Shanghai Jiao Tong University in 2015, and the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology in 2020. He is currently a senior researcher with 4Paradigm. He has authored or coauthored five top-tier conference/journal papers as first-author, including NeurIPS, ACL, WebConf, ICDE, and VLDB-J. His research interests include knowledge graph embedding, automated machine learning, and graph learning. He was a program committee for AAAI 2020–2022, IJCAI 2020–2022, CIKM 2021, KDD 2022, and ICML 2022, and a reviewer of TKDE and NEUNET.

Quanming Yao (Member, IEEE) is currently a tenure-track assistant professor with the Department of Electronic Engineering, Tsinghua University. Before that, he spent three years from a researcher to a senior scientist with 4Paradigm INC, where he set up and led the company's machine learning research team. His current research interests include automated machine learning (AutoML) and neural architecture search (NAS). He was the recipient of the Wunwen Jun Prize of Excellence Youth of Artificial Intelligence (issued by CAAI), runner up of Ph.D. Research Excellence Award (School of Engineering, HKUST), and winner of Google Fellowship (in machine learning). He was an area chair of ICLR 2022, IJCAI 2021, and ACML 2021, a senior program committee of IJCAI 2020 and AAAI 2020–2021, and a guest editor of IEEE TPAMI AutoML special issue in 2019.

James T. Kwok (Fellow, IEEE) received the PhD degree in computer science from The Hong Kong University of Science and Technology in 1996. He is currently a professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include machine learning, deep learning, and artificial intelligence. He was the recipient of the IEEE Outstanding 2004 Paper Award and Second Class Award in Natural Sciences by the Ministry of Education, China, in 2008. He is an associate editor of *IEEE Transactions on Neural Networks and Learning Systems*, *Neural Networks*, *Neurocomputing*, *Artificial Intelligence Journal*, *International Journal of Data Science and Analytics*, an editorial board member of Machine Learning, board member, and the vice president for Publications of Asia Pacific Neural Network Society. He was/is also a senior area chairs or area chairs of main machine learning or AI conferences, including NIPS, ICML, ICLR, IJCAI, AAAI, and ECML.