

---

# Interstellar: Searching Recurrent Architecture for Knowledge Graph Embedding

---

Yongqi Zhang<sup>1,3</sup> Quanming Yao<sup>1,2</sup> Lei Chen<sup>3</sup>

<sup>1</sup>4Paradigm Inc.

<sup>2</sup>Department of Electronic Engineering, Tsinghua University

<sup>3</sup>Department of Computer Science and Engineering, HKUST  
{zhangyongqi,yaoquanming}@4paradigm.com, leichen@cse.ust.hk

## Abstract

Knowledge graph (KG) embedding is well-known in learning representations of KGs. Many models have been proposed to learn the interactions between entities and relations of the triplets. However, long-term information among multiple triplets is also important to KG. In this work, based on the relational paths, which are composed of a sequence of triplets, we define the Interstellar as a recurrent neural architecture search problem for the short-term and long-term information along the paths. First, we analyze the difficulty of using a unified model to work as the Interstellar. Then, we propose to search for recurrent architecture as the Interstellar for different KG tasks. A case study on synthetic data illustrates the importance of the defined search problem. Experiments on real datasets demonstrate the effectiveness of the searched models and the efficiency of the proposed hybrid-search algorithm.<sup>1</sup>

## 1 Introduction

Knowledge Graph (KG) [4, 43, 52] is a special kind of graph with many relational facts. It has inspired many knowledge-driven applications, such as question answering [31, 38], medical diagnosis [60], and recommendation [29]. An example of the KG is in Figure 1(a). Each relational fact in KG is represented as a triplet in the form of (*subject entity*, *relation*, *object entity*), abbreviated as (*s*, *r*, *o*). To learn from the KGs and benefit the downstream tasks, embedding based methods, which learn low-dimensional vector representations of the entities and relations, have recently developed as a promising direction to serve this purpose [8, 19, 45, 52].

Many efforts have been made on modeling the plausibility of triplets (*s*, *r*, *o*)s through learning embeddings. Representative works are triplet-based models, such as TransE [8], ComplEx [49], ConvE [14], RotatE [44], AutoSF [61], which define different embedding spaces and learn on single triplet (*s*, *r*, *o*). Even though these models perform well in capturing short-term semantic information inside the triplets in KG, they still cannot capture the information among multiple triplets.

In order to better capture the complex information in KGs, the relational path is introduced as a promising format to learn composition of relations [20, 28, 39] and long-term dependency of triplets [27, 12, 19, 51]. As in Figure 1(b), a relational path is defined as a set of *L* triplets (*s*<sub>1</sub>, *r*<sub>1</sub>, *o*<sub>1</sub>), (*s*<sub>2</sub>, *r*<sub>2</sub>, *o*<sub>2</sub>), ..., (*s*<sub>*L*</sub>, *r*<sub>*L*</sub>, *o*<sub>*L*</sub>), which are connected head-to-tail in sequence, i.e. *o*<sub>*i*</sub> = *s*<sub>*i*+1</sub>,  $\forall i = 1 \dots L - 1$ . The paths not only preserve every single triplet but also can capture the dependency among a sequence of triplets. Based on the relational paths, the triplet-based models can be compatible by working on each triplet (*s*<sub>*i*</sub>, *r*<sub>*i*</sub>, *o*<sub>*i*</sub>) separately. TransE-Comp [20] and PTransE [28] learn the composition relations on the relational paths. To capture the long-term information

<sup>1</sup>Code is available at <https://github.com/AutoML-4Paradigm/Interstellar>, and correspondence is to Q. Yao.

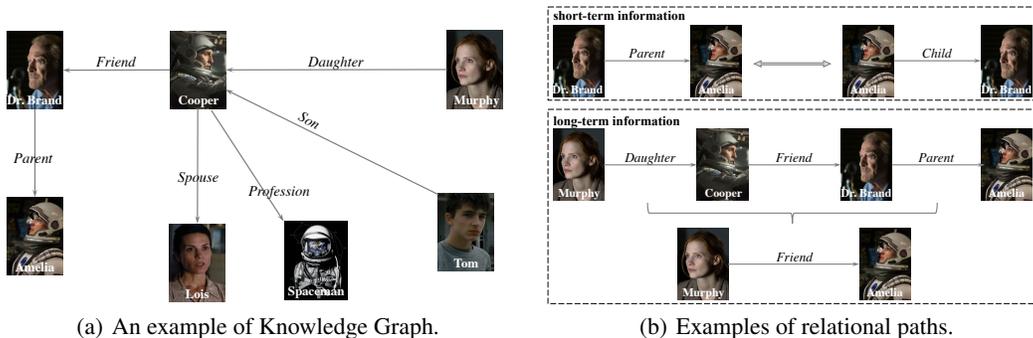


Figure 1: Short-term information is represented by a single triplet. Long-term information passes across multiple triplets. The two kinds of information in KGs can be preserved in the relational path.

in KGs, Chains [12] and RSN [19] design customized RNN to leverage all the entities and relations along path. However, the RNN models still overlook the semantics inside each triplet [19]. Another type of models leverage Graph Convolution Network (GCN) [25] to extract structural information in KGs, e.g. R-GCN, GCN-Align [53], CompGCN [50]. However, GCN-based methods do not scale well since the entire KG needs to be processed and it has large sample complexity [16].

In this paper, we observe that the relational path is an important and effective data structure that can preserve both short-term and long-term information in KG. Since the semantic patterns and the graph structures in KGs are diverse [52], how to leverage the short-term and long-term information for a specific KG task is non-trivial. Inspired by the success of neural architecture search (NAS) [15], we propose to search recurrent architectures as the *Interstellar* to learn from the relational path. The contributions of our work are summarized as follows:

1. We analyze the difficulty and importance of using the relational path to learn the short-term and long-term information in KGs. Based on the analysis, we define the *Interstellar* as a recurrent network to process the information along the relational path.
2. We formulate the above problem as a NAS problem and propose a domain-specific search space. Different from searching RNN cells, the recurrent network in our space is specifically designed for KG tasks and covers many human-designed embedding models.
3. We identify the problems of adopting stand-alone and one-shot search algorithms for our search space. This motivates us to design a hybrid-search algorithm to search efficiently.
4. We use a case study on the synthetic data set to show the reasonableness of our search space. Empirical experiments on entity alignment and link prediction tasks demonstrate the effectiveness of the searched models and the efficiency of the search algorithm.

**Notations.** We denote vectors by lowercase boldface, and matrix by uppercase boldface. A KG  $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$  is defined by the set of entities  $\mathcal{E}$ , relations  $\mathcal{R}$  and triplets  $\mathcal{S}$ . A triplet  $(s, r, o) \in \mathcal{S}$  represents a relation  $r$  that links from the subject entity  $s$  to the object entity  $o$ . The embeddings in this paper are denoted as boldface letters of indexes, e.g.  $\mathbf{s}, \mathbf{r}, \mathbf{o}$  are embeddings of  $s, r, o$ . " $\odot$ " is the element-wise multiply and " $\otimes$ " is the Hermitian product [49] in complex space.

## 2 Related Works

### 2.1 Representation Learning in Knowledge Graph (KG)

Given a single triplet  $(s, r, o)$ , TransE [8] models the relation  $r$  as a translation vector from subject entity  $s$  to object entity  $o$ , i.e., the embeddings satisfy  $\mathbf{s} + \mathbf{r} \approx \mathbf{o}$ . The following works DistMult [55], ComplEx [49], ConvE [14], RotatE [44], etc., interpret the interactions among embeddings  $\mathbf{s}, \mathbf{r}$  and  $\mathbf{o}$  in different ways. All of them learn embeddings based on single triplet.

In KGs, a relational path is a sequence of triplets. PTransE [28] and TransE-Comp [20] propose to learn the composition of relations  $(r_1, r_2, \dots, r_n)$ . In order to combine more pieces of information in KG, Chains [12] and RSN [19] are proposed to jointly learn the entities and relations along the relational path. With different connections and combinators, these models process short-term and long-term information in different ways.

Table 1: The recurrent function of existing KG embedding models. We represent the triplet/path-based models by Definition 1.  $\mathcal{N}(\cdot)$  denotes the neighbors of an entity.  $\mathbf{W} \in \mathbb{R}^{d \times d}$ 's are different weight matrices.  $\sigma$  is a non-linear activation function. ‘‘cell’’ means a RNN cell [13], like GRU [10] LSTM [46] etc. For mini-batch complexity,  $m$  is the batch size,  $d$  is the embedding dimension.

type	model	unit function	complexity	
triplet-based	TransE [8]	$\mathbf{v}_t = \mathbf{s}_t + \mathbf{r}_t, \mathbf{h}_t = 0$	$O(md)$	
	ComplEx [49]	$\mathbf{v}_t = \mathbf{s}_t \otimes \mathbf{r}_t, \mathbf{h}_t = 0$	$O(md)$	
GCN-based	R-GCN [40]	$\mathbf{s}_t = \sigma(\mathbf{s}_{t-1} + \sum_{s' \in \mathcal{N}(s)} \mathbf{W}_t^{(r)} \mathbf{s}'_{t-1})$	$O( \mathcal{E}  \mathcal{R} d)$	
	GCN-Align [53]	$\mathbf{s}_t = \sigma(\mathbf{s}_{t-1} + \sum_{s' \in \mathcal{N}(s)} \mathbf{W}_t \mathbf{s}'_{t-1})$	$O( \mathcal{E} d)$	
path-based	PTransE [28]	add	$\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{r}_t$	$O(mLd)$
		multiply	$\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \mathbf{h}_{t-1} \odot \mathbf{r}_t$	$O(mLd)$
		RNN	$\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \text{cell}(\mathbf{r}_t, \mathbf{h}_{t-1})$	$O(mLd^2)$
	Chains [12]	$\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \text{cell}(\mathbf{s}_t, \mathbf{r}_t, \mathbf{h}_{t-1})$	$O(mLd^2)$	
	RSN [19]	$\mathbf{v}_t = \mathbf{W}_1 \mathbf{s}_t + \mathbf{W}_2 \mathbf{h}_t, \mathbf{h}_t = \text{cell}(\mathbf{r}_t, \text{cell}(\mathbf{s}_t, \mathbf{h}_{t-1}))$	$O(mLd^2)$	
	Interstellar	a searched recurrent network	$O(mLd^2)$	

Graph convolutional network (GCN) [25] have recently been developed as a promising method to learn from graph data. As a special instance of graph, GCN has also been introduced in KG learning, e.g., R-GCN [40], GCN-Align [53], VR-GCN [58] and CompGCN [50]. However, these models are hard to scale well since the whole KG should be loaded. for each training iteration. Besides, GCN has been theoretically proved to have worse generalization guarantee than RNN in sequence learning tasks (Section 5.2 in [16]). Table 1 summarizes above works and compares them with the proposed Interstellar, which is a NAS method customized to path-based KG representation learning.

## 2.2 Neural Architecture Search (NAS)

Searching for better neural networks by NAS techniques have broken through the bottleneck in manual architecture designing [15, 21, 41, 63, 56]. To guarantee effectiveness and efficiency, the first thing we should care about is the search space. It defines what architectures can be represented in principle, like CNN or RNN. In general, the search space should be powerful but also tractable. The space of CNN has been developed from searching the macro architecture [63], to micro cells [30] and further to the larger and sophisticated cells [47]. Many promising architectures have been searched to outperform human-designed CNNs in literature [2, 54]. However, designing the search space for recurrent neural network attracts little attention. The searched architectures mainly focus on cells rather than connections among cells [37, 63].

To search efficiently, evaluation method, which provides feedback signals, and search algorithm, which guides the optimization direction, should be simultaneously considered. There are two important approaches for NAS. 1) Stand-alone methods, i.e. separately training and evaluating each model from scratch, are the most guaranteed way to compare different architectures, whereas very slow. 2) One-shot search methods, e.g., DARTS [30], ASNG [2] and NASP [57], have recently become the most popular approach that can efficiently find good architectures. Different candidates are approximately evaluated in a supernet with parameter-sharing (PS). However, PS is not always reliable, especially in complex search spaces like the macro space of CNNs and RNNs [6, 37, 41].

## 3 The Proposed Method

As in Section 2.1, the relational path is informative in representing knowledge in KGs. Since the path is a sequence of triplets with varied length, it is intuitive to use Recurrent Neural Network (RNN), which is known to have a universal approximation ability [1], to model the path as language models [46]. While RNN can capture the long-term information along steps [33, 10], it will overlook domain-specific properties like the semantics inside each triplet without a customized architecture [19]. Besides, what kind of information should we leverage varies from tasks [52]. Finally, as proved in [16], RNN has better generalization guarantee than GCN. Thus, to design a proper KG model, we define the path modeling as a NAS problem for RNN here.

### 3.1 Designing a Recurrent Search Space

To start with, we firstly define a general recurrent function (Interstellar) on the relational path.

**Definition 1** (Interstellar). An Interstellar processes the embeddings of  $\mathbf{s}_1, \mathbf{r}_1$  to  $\mathbf{s}_L, \mathbf{r}_L$  recurrently. In each recurrent step  $t$ , the Interstellar combines embeddings of  $\mathbf{s}_t, \mathbf{r}_t$  and the preceding information  $\mathbf{h}_{t-1}$  to get an output  $\mathbf{v}_t$ . The Interstellar is formulated as a recurrent function

$$[\mathbf{v}_t, \mathbf{h}_t] = f(\mathbf{s}_t, \mathbf{r}_t, \mathbf{h}_{t-1}), \quad \forall t = 1 \dots L, \quad (1)$$

where  $\mathbf{h}_t$  is the recurrent hidden state and  $\mathbf{h}_0 = \mathbf{s}_1$ . The output  $\mathbf{v}_t$  is to predict object entity  $o_t$ .

In each step  $t$ , we focus on one triplet  $(s_t, r_t, o_t)$ . To design the search space  $\mathcal{A}$ , we need to figure out what are important properties in (1). Since  $\mathbf{s}_t$  and  $\mathbf{r}_t$  are indexed from different embedding sets, we introduce two operators to make a difference, i.e.  $O_s$  for  $\mathbf{s}_t$  and  $O_r$  for  $\mathbf{r}_t$  as in Figure 2. Another operator  $O_v$  is used to model the output  $\mathbf{v}_t$  in (1). Then, the hidden state  $\mathbf{h}_t$  is defined to propagate the information across triplets. Taking Chains [12] as an example,  $O_s$  is an adding combinator for  $\mathbf{h}_{t-1}$  and  $\mathbf{s}_t$ ,  $O_r$  is an RNN cell to combine  $O_s$  and  $\mathbf{r}_t$ , and  $O_v$  directly outputs  $O_r$  as  $\mathbf{v}_t$ .

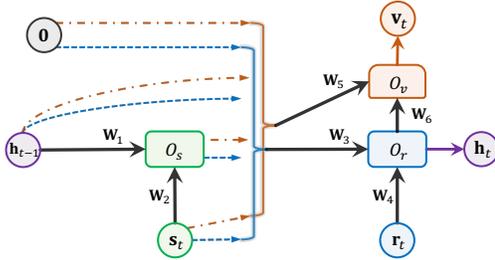


Figure 2: Search space  $\mathcal{A}$  of  $f$  for (1).

Table 2: The split search space of  $f$  in Figure 2 into macro-level  $\alpha_1$  and micro-level  $\alpha_2$ .

micro-level	activation	identity, tanh, sigmoid
$\alpha_1 \in \mathcal{A}_1$	weight matrix	$\{\mathbf{W}_i\}_{i=1}^6, \mathbf{I}$
macro-level	connections	$\mathbf{h}_{t-1}, O_s, \mathbf{0}, \mathbf{s}_t$
$\alpha_2 \in \mathcal{A}_2$	combinators	$+, \odot, \otimes, \text{gated}$

To control the information flow, we search the connections from input vectors to the outputs, i.e. the dashed lines in Figure 2. Then, the combiners, which combine two vectors into one, are important since they determine how embedding are transformed, e.g. “+” in TransE [8] and “ $\odot$ ” in DistMult [55]. As in the search space of RNN [63], we introduce activation functions *tanh* and *sigmoid* to give non-linear squashing. Each link is either a trainable weight matrix  $\mathbf{W}$  or an identity matrix  $\mathbf{I}$  to adjust the vectors. Detailed information is listed in Table 2.

Let the training and validation set be  $\mathcal{G}_{\text{tra}}$  and  $\mathcal{G}_{\text{val}}$ ,  $\mathcal{M}$  be the measurement on  $\mathcal{G}_{\text{val}}$  and  $\mathcal{L}$  be the loss on  $\mathcal{G}_{\text{tra}}$ . To meet different requirements on  $f$ , we propose to search the architecture  $\alpha$  of  $f$  as a special RNN. The network here is not a general RNN but one specific to the KG embedding tasks. The problem is defined to find an architecture  $\alpha$  such that validation performance is maximized, i.e.,

$$\alpha^* = \arg \max_{\alpha \in \mathcal{A}} \mathcal{M}(f(\mathbf{F}^*; \alpha), \mathcal{G}_{\text{val}}), \quad \text{s.t.} \quad \mathbf{F}^* = \arg \min_{\mathbf{F}} \mathcal{L}(f(\mathbf{F}; \alpha), \mathcal{G}_{\text{tra}}), \quad (2)$$

which is a bi-level optimization problem and is non-trivial to solve. First, the computation cost to get  $\mathbf{F}^*$  is generally high. Second, searching for  $\alpha \in \mathcal{A}$  is a discrete optimization problem [30] and the space is large (in Appendix A.1). Thus, how to efficiently search the architectures is a big challenge.

Compared with standard RNNs, which recurrently model each input vectors, the Interstellar models the relational path with triplets as basic unit. In this way, we can determine how to model short-term information inside each triplet and what long-term information should be passed along the triplets. This makes our search space distinctive for the KG embedding problems.

### 3.2 Proposed Search Algorithm

In Section 3.1, we have introduced the search space  $\mathcal{A}$ , which contains considerable different architectures. Therefore, how to search efficiently in  $\mathcal{A}$  is an important problem. Designing appropriate optimization algorithm for the discrete architecture parameters is a big challenge.

#### 3.2.1 Problems with Existing Algorithms

As introduced in Section 2.2, we can either choose the stand-alone approach or one-shot approach to search the network architecture. In order to search efficiently, search algorithms should be designed for specific scenarios [63, 30, 2]. When the search space is complex, e.g. the macro space of CNNs and the tree-structured RNN cells [63, 37], stand-alone approach is preferred since it can provide accurate evaluation feedback while PS scheme is not reliable [6, 41]. However, the computation

cost of evaluating an architecture under the stand-alone approach is high, preventing us to efficiently search in large spaces.

One-shot search algorithms are widely used in searching micro spaces, e.g. cell structures in CNNs and simplified DAGs in RNN cells [2, 37, 30]. Searching architectures on a supernet with PS in the simplified space is relatively possible. However, since the search space in our problem is more complex and the embedding status influences the evaluation, PS is not reliable (see Appendix B.2). Therefore, one-shot algorithms is not appropriate in our problem.

### 3.2.2 Hybrid-search Algorithm

Even though the two types of algorithms have their limitations, is it possible to take advantage from both of them? Back to the development from stand-alone search in NASNet [63] to one-shot search in ENAS [37], the search space is simplified from a macro space to micro cells. We are motivated to split the space  $\mathcal{A}$  into a macro part  $\mathcal{A}_1$  and a micro part  $\mathcal{A}_2$  in Table 2.  $\alpha_1 \in \mathcal{A}_1$  controls the connections and combinator, influencing information flow a lot; and  $\alpha_2 \in \mathcal{A}_2$  fine-tunes the architecture through activations and weight matrix. Besides, we empirically observe that PS for  $\alpha_2$  is reliable (in Appendix B.2). Then we propose a hybrid-search method that can be both fast and accurate in Algorithm 1. Specifically,  $\alpha_1$  and  $\alpha_2$  are sampled from a controller  $c$  — a distribution [2, 54] or a neural network [63]. The evaluation feedback of  $\alpha_1$  and  $\alpha_2$  are obtained through the stand-alone manner and one-shot manner respectively. After the searching procedure, the best architecture is sampled and we fine-tune the hyper-parameters to achieve the better performance.

---

**Algorithm 1** Proposed search recurrent architecture as the Interstellar algorithm.

---

**Require:** search space  $\mathcal{A} \equiv \mathcal{A}_1 \cup \mathcal{A}_2$  in Figure 2, controller  $c$  for sampling  $\alpha = [\alpha_2, \alpha_1]$ .

- 1: **repeat**
  - 2:   sample the *micro-level* architecture  $\alpha_2 \in \mathcal{A}_2$  by  $c$ ;
  - 3:   update the controller  $c$  for  $k_1$  steps using Algorithm 2 (in *stand-alone* manner);
  - 4:   sample the *macro-level* architecture  $\alpha_1 \in \mathcal{A}_1$  by  $c$ ;
  - 5:   update the controller  $c$  for  $k_2$  steps using Algorithm 3 (in *one-shot* manner);
  - 6: **until** termination
  - 7: Fine-tune the hyper-parameters for the best architecture  $\alpha^* = [\alpha_1^*, \alpha_2^*]$  sampled from  $c$ .
  - 8: **return**  $\alpha^*$  and the fine-tuned hyper-parameters.
- 

In the macro-level (Algorithm 2), once we get the architecture  $\alpha$ , the parameters are obtained by full model training. This ensures that the evaluation feedback for macro architectures  $\alpha_1$  is reliable. In the one-shot stage (Algorithm 3), the main difference is that, the parameters  $F$  are not initialized and different architectures are evaluated on the same set of  $F$ , i.e. by PS. This improves the efficiency of evaluating micro architecture  $\alpha_2$  without full model training.

---

**Algorithm 2** Macro-level ( $\alpha_1$ ) update

---

**Require:** controller  $c$ ,  $\alpha_2 \in \mathcal{A}_2$ , parameters  $F$ .

- 1: sample an individual  $\alpha_1$  by  $c$  to get the architecture  $\alpha = [\alpha_1, \alpha_2]$ ;
  - 2: initialize  $F$  and train to obtain  $F^*$  until converge by minimizing  $L(f(F, \alpha), \mathcal{G}_{\text{tra}})$ ;
  - 3: evaluate  $\mathcal{M}(f(F^*, \alpha), \mathcal{G}_{\text{val}})$  to update  $c$ .
  - 4: **return** the updated controller  $c$ .
- 

---

**Algorithm 3** Micro-level ( $\alpha_2$ ) update

---

**Require:** controller  $c$ ,  $\alpha_1 \in \mathcal{A}_1$ , parameters  $F$ .

- 1: sample an individual  $\alpha_2$  by  $c$  to get the architecture  $\alpha = [\alpha_1, \alpha_2]$ ;
  - 2: sample a mini-batch  $\mathcal{B}_{\text{tra}}$  from  $\mathcal{G}_{\text{tra}}$  and update  $F$  with gradient  $\nabla_F \mathcal{L}(f(F, \alpha), \mathcal{B}_{\text{tra}})$ ;
  - 3: sample a mini-batch  $\mathcal{B}_{\text{val}}$  from  $\mathcal{G}_{\text{val}}$  and evaluate  $\mathcal{M}(f(F, \alpha), \mathcal{B}_{\text{val}})$  to update  $c$ .
  - 4: **return** the updated controller  $c$ .
- 

The remaining problem is how to model and update the controller, i.e. step 3 in Algorithm 2 and step 3 in Algorithm 3. The evaluation metrics in KG tasks are usually ranking-based, e.g. Hit@ $k$ , and are non-differentiable. Instead of using the direct gradient, we turn to the derivative-free optimization methods [11], such as reinforcement learning (policy gradient in [63] and Q-learning in [5]), or Bayes optimization [7]. Inspired by the success of policy gradient in searching CNNs and RNNs [63, 37], we use policy gradient to optimize the controller  $c$ .

Following [2, 17, 54], we use stochastic relaxation for the architectures  $\alpha$ . Specifically, the parametric probability distribution  $\alpha \sim p_{\theta}(\alpha)$  is introduced on the search space  $\alpha \in \mathcal{A}$  and then the distribution parameter  $\theta$  is optimized to maximize the expectation of validation performance

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{\alpha \sim p_{\theta}(\alpha)} [\mathcal{M}(f(F^*; \alpha), \mathcal{G}_{\text{val}})]. \quad (3)$$

Then, the optimization target is transformed from (2) with the discrete  $\alpha$  into (3) with the continuous  $\theta$ . In this way,  $\theta$  is updated by  $\theta_{t+1} = \theta_t + \rho \nabla_{\theta} J(\theta_t)$ , where  $\rho$  is the step-size, and  $\nabla_{\theta} J(\theta) = \mathbb{E} [\mathcal{M}(f(F; \alpha), \mathcal{G}_{\text{val}}) \nabla_{\theta} \ln(p_{\theta_t}(\alpha))]$  is the pseudo gradient of  $\alpha$ . The key observation is that we do not need to take direct gradient w.r.t.  $\mathcal{M}$ , which is not available. Instead, we only need to get the validation performance measured by  $\mathcal{M}$ .

To further improve the efficiency, we use natural policy gradient (NPG) [35]  $\tilde{\nabla}_{\theta} J(\theta_t)$  to replace  $\nabla_{\theta} J(\theta_t)$ , where  $\tilde{\nabla}_{\theta} J(\theta_t) = [\mathbf{H}(\theta_t)]^{-1} \nabla_{\theta} J(\theta_t)$  is computed by multiplying a Fisher information matrix  $\mathbf{H}(\theta_t)$  [3]. NPG has shown to have better convergence speed [2, 3, 35] (see Appendix A.2).

## 4 Experiments

### 4.1 Experiment Setup

Following [18, 19, 20], we sample the relational paths from biased random walks (details in Appendix A.3). We use two basic tasks in KG, i.e. entity alignment and link prediction. Same as the literature [8, 19, 52, 62], we use the “filtered” ranking metrics: mean reciprocal ranking (MRR) and Hit@ $k$  ( $k = 1, 10$ ). Experiments are written in Python with PyTorch framework [36] and run on a single 2080Ti GPU. Statistics of the data set we use in this paper is in Appendix A.4. Training details of each task are given in Appendix A.5. Besides, all of the searched models are shown in Appendix C due to space limitations.

### 4.2 Understanding the Search Space

Here, we illustrate the designed search space  $\mathcal{A}$  in Section 3.1 using Countries [9] dataset, which contains 271 countries and regions, and 2 relations *neighbor* and *locatedin*. This dataset contains three tasks: S1 infers *neighbor*  $\wedge$  *locatedin*  $\rightarrow$  *locatedin* or *locatedin*  $\wedge$  *locatedin*  $\rightarrow$  *locatedin*; S2 require to infer the 2 hop relations *neighbor*  $\wedge$  *locatedin*  $\rightarrow$  *locatedin*; S3 is harder and requires modeling 3 hop relations *neighbor*  $\wedge$  *locatedin*  $\wedge$  *locatedin*  $\rightarrow$  *locatedin*.

To understand the dependency on the length of paths for various tasks, we extract four subspaces from  $\mathcal{A}$  (in Figure 2) with different connections. Specifically, (P1) represents the single hop embedding models; (P2) processes 2 successive steps; (P3) models long relational paths without intermediate entities; and (P4) includes both entities and relations along path. Then we search  $f$  in P1-P4 respectively to see how well these subspaces can tackle the tasks S1-S3.

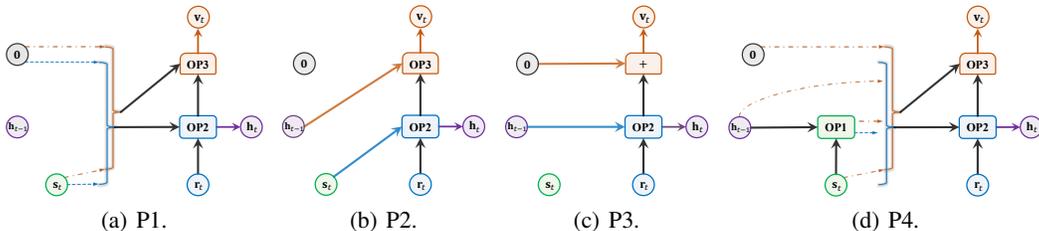


Figure 3: Four subspaces with different connection components in the recurrent search space.

For each task, we randomly generate 100 models for each subspace and record the model with the best *area under curve of precision recall* (AUC-PR) on validation set. This procedure is repeated 5 times to evaluate the testing performance in Table 3. We can see that there is no single subspace performing well on all tasks. For easy tasks S1 and S2, short-term information is more important. Incorporating entities along path like P4 is bad for learning 2 hop relationships. For the harder task S3, P3 and P4 outperform the others since it can model long-term information in more steps.

Table 3: Performance on Countries dataset.

	S1	S2	S3
P1	0.998±0.001	0.997±0.002	0.933±0.031
P2	<b>1.000±0.000</b>	0.999±0.001	0.952±0.023
P3	0.992±0.001	<b>1.000±0.000</b>	0.961±0.016
P4	0.977±0.028	0.984±0.010	<b>0.964±0.015</b>
<b>Interstellar</b>	<b>1.000±0.000</b>	<b>1.000±0.000</b>	<b>0.968±0.007</b>

Besides, we evaluate the best model searched in the whole space  $\mathcal{A}$  for S1-S3. As in the last line of Table 3, the model searched by Interstellar achieves good performance on the hard task S3. Interstellar prefers different candidates (see Appendix C.2) for S1-S3 over the same search space.

This verifies our analysis that it is difficult to use a unified model that can adapt to the short-term and long term information for different KG tasks.

### 4.3 Comparison with State-of-the-art KG Embedding Methods

**Entity Alignment.** The entity alignment task aims to align entities in different KGs referring the same instance. In this task, long-term information is important since we need to propagate the alignment across triplets [19, 45, 62]. We use four cross-lingual and cross-database subset from DBpedia and Wikidata generated by [19], i.e. DBP-WD, DBP-YG, EN-FR, EN-DE. For fair comparison, we follow the same path sampling scheme and the data set splits in [19].

Table 4: Performance comparison on entity alignment task.  $H@k$  is short for  $\text{Hit}@k$ . The results of TransD [22], BootEA [45], IPTransE [62], GCN-Align [53] and RSN [19] are copied from [19].

models	DBP-WD			DBP-YG			EN-FR			EN-DE			
	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	
triplet	TransE	18.5	42.1	0.27	9.2	24.8	0.15	16.2	39.0	0.24	20.7	44.7	0.29
	TransD*	27.7	57.2	0.37	17.3	41.6	0.26	21.1	47.9	0.30	24.4	50.0	0.33
	BootEA*	32.3	63.1	0.42	31.3	62.5	0.42	31.3	62.9	0.42	44.2	70.1	0.53
GCN	GCN-Align	17.7	37.8	0.25	19.3	41.5	0.27	15.5	34.5	0.22	25.3	46.4	0.22
	VR-GCN	19.4	55.5	0.32	20.9	55.7	0.32	16.0	50.8	0.27	24.4	61.2	0.36
	R-GCN	8.6	31.4	0.16	13.3	42.4	0.23	7.3	31.2	0.15	18.4	44.8	0.27
path	PTransE	16.7	40.2	0.25	7.4	14.7	0.10	7.3	19.7	0.12	27.0	51.8	0.35
	IPTransE*	23.1	51.7	0.33	22.7	50.0	0.32	25.5	55.7	0.36	31.3	59.2	0.41
	Chains	32.2	60.0	0.42	35.3	64.0	0.45	31.4	60.1	0.41	41.3	68.9	0.51
	RSN*	38.8	65.7	0.49	40.0	67.5	0.50	34.7	63.1	0.44	48.7	72.0	0.57
	<b>Interstellar</b>	<b>40.7</b>	<b>71.2</b>	<b>0.51</b>	<b>40.2</b>	<b>72.0</b>	<b>0.51</b>	<b>35.5</b>	<b>67.9</b>	<b>0.46</b>	<b>50.1</b>	<b>75.6</b>	<b>0.59</b>

Table 4 compares the testing performance of the models searched by Interstellar and human-designed ones on the *Normal* version datasets [19] (the *Dense* version [19] in Appendix B.1). In general, the path-based models are better than the GCN-based and triplets-based models by modeling long-term dependencies. BootEA [45] and IPTransE [62] win over TransE [8] and PTransE [28] respectively by iteratively aligning discovered entity pairs. Chains [12] and RSN [19] outperform graph-based models and the other path-based models by explicitly processing both entities and relations along path. In comparison, Interstellar is able to search and balance the short-term and long-term information adaptively, thus gains the best performance. We plot the learning curve on DBP-WD of some triplet, graph and path-based models in Figure 4(a) to verify the effectiveness of the relational paths.

**Link Prediction.** In this task, an incomplete KG is given and the target is to predict the missing entities in unknown links [52]. We use three famous benchmark datasets, WN18-RR [14] and FB15k-237 [48], which are more realistic than their superset WN18 and FB15k [8], and YAGO3-10 [32], a much larger dataset.

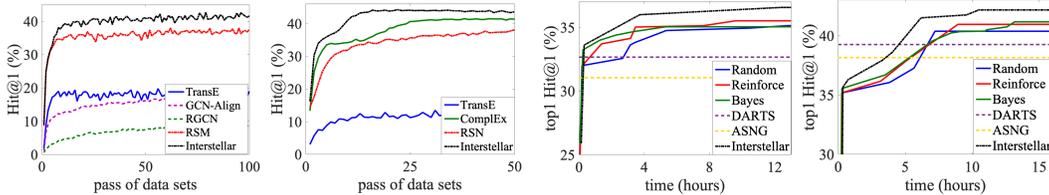
Table 5: Link prediction results.

models	WN18-RR			FB15k-237			YAGO3-10		
	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR
TransE	12.5	44.5	0.18	17.3	37.9	0.24	10.3	27.9	0.16
CompLex	41.4	49.0	0.44	22.7	49.5	0.31	40.5	62.8	0.48
RotatE*	43.6	54.2	0.47	<b>23.3</b>	50.4	<b>0.32</b>	40.2	63.1	0.48
R-GCN	-	-	-	15.1	41.7	0.24	-	-	-
PTransE	27.2	46.4	0.34	20.3	45.1	0.29	12.2	32.3	0.19
RSN	38.0	44.8	0.40	19.2	41.8	0.27	16.4	37.3	0.24
Interstellar	<b>44.0</b>	<b>54.8</b>	<b>0.48</b>	<b>23.3</b>	<b>50.8</b>	<b>0.32</b>	<b>42.4</b>	<b>66.4</b>	<b>0.51</b>

We search architectures with dimension 64 to save time and compare the models with dimension 256. Results of R-GCN on WN18-RR and YAGO3-10 are not available due to out-of-memory issue. As shown in Table 5, PTransE outperforms TransE by modeling compositional relations, but worse than CompLex and RotatE since the adding operation is inferior to  $\otimes$  when modeling the interaction between entities and relations [49]. RSN is worse than CompLex/RotatE since it pays more attention to long-term information rather than the inside semantics. Interstellar outperforms the path-based methods PTransE and RSN by searching architectures that model fewer steps (see Appendix C.4). And it works comparable with the triplet-based models, i.e. CompLex and RotatE, which are specially designed for this task. We show the learning curve of TransE, CompLex, RSN and Interstellar on WN18-RR in Figure 4(b).

#### 4.4 Comparison with Existing NAS Algorithms

In this part, we compare the proposed Hybrid-search algorithm in Interstellar with the other NAS methods. First, we compare the proposed algorithm with stand-alone NAS methods. Once an architecture is sampled, the parameters are initialized and trained into converge to give reliable feedback. Random search (*Random*), Reinforcement learning (*Reinforce*) [63] and Bayes optimization (*Bayes*) [7] are chosen as the baseline algorithms. As shown in Figure 5 (entity alignment on DBP-WD and link prediction on WN18-RR), the Hybrid algorithm in Interstellar is more efficient since it takes advantage of the one-shot approach to search the micro architectures in  $\mathcal{A}_2$ .



(a) Entity alignment.

(b) Link prediction.

(a) Entity alignment.

(b) Link prediction.

Figure 4: Single model learning curve.

Figure 5: Compare with NAS methods.

Then, we compare with one-shot NAS methods with PS on the entire space. DARTS [30] and ASNG [2] are chosen as the baseline models. For DARTS, the gradient is obtained from loss on training set since validation metric is not differentiable. We show the performance of the best architecture found by the two one-shot algorithms. As shown in the dashed lines, the architectures found by one-shot approach are much worse than that by the stand-alone approaches. The reason is that PS is not reliable in our complex recurrent space (more experiments in Appendix B.2). In comparison, Interstellar is able to search reliably and efficiently by taking advantage of both the stand-alone approach and the one-shot approach.

#### 4.5 Searching Time Analysis

We show the clock time of Interstellar on entity alignment and link prediction tasks in Table 6. Since the datasets used in entity alignment task have similar scales (see Appendix A.4), we show them in the same column. For each task/dataset, we show the computation cost of the macro-level and micro-level in Interstellar for 50 iterations between step 2-5 in Algorithm 1 (20 for YAGO3-10 dataset); and the fine-tuning procedure after searching for 50 groups of hyper-parameters, i.e. learning rate, decay rate, dropout rate, L2 penalty and batch-size (details in Appendix A.5). As shown, the entity alignment tasks take about 15-25 hours, while link prediction tasks need about one or more days due to the larger data size. The cost of the search process is at the same scale with that of the fine-tuning time, which shows the search process is not expensive.

Table 6: Comparison of searching and fine-tuning time (in hours) in Algorithm 1.

procedure		entity alignment		link prediction		
		Normal	Dense	WN18-RR	FB15k-237	YAGO3-10
search	macro-level (line 2-3)	9.9±1.5	14.9±0.3	11.7±1.9	23.2±3.4	91.6±8.7
	micro-level (line 4-5)	4.2±0.2	7.5±0.6	6.3±0.9	5.6±0.4	10.4±1.3
fine-tune (line 7)		11.6±1.6	16.2±2.1	44.3±2.3	67.6±4.5	> 200

## 5 Conclusion

In this paper, we propose a new NAS method, Interstellar, to search RNN for learning from the relational paths, which contain short-term and long-term information in KGs. By designing a specific search space based on the important properties in relational path, Interstellar can adaptively search promising architectures for different KG tasks. Furthermore, we propose a hybrid-search algorithm that is more efficient compared with the other the state-of-art NAS algorithms. The experimental results verifies the effectiveness and efficiency of Interstellar on various KG embedding benchmarks. In future work, we can combine Interstellar with AutoSF [61] to give further improvement on the embedding learning problems. Taking advantage of data similarity to improve the search efficiency on new datasets is another extension direction.

## Broader impact

Most of the attention on KG embedding learning has been focused on the triplet-based models. In this work, we emphasize the benefits and importance of using relational paths to learn from KGs. And we propose the path-interstellar as a recurrent neural architecture search problem. This is the first work applying neural architecture search (NAS) methods on KG tasks.

In order to search efficiently, we propose a novel hybrid-search algorithm. This algorithm addresses the limitations of stand-alone and one-shot search methods. More importantly, the hybrid-search algorithm is not specific to the problem here. It is also possible to be applied to the other domains with more complex search space [63, 47, 42].

One limitation of this work is that, the Interstellar is currently limited on the KG embedding tasks. Extending to reasoning tasks like DRUM [39] is an interesting direction.

## Acknowledgment

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

## References

- [1]
- [2] Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, and K. Nishida. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*, pages 171–180, 2019.
- [3] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.
- [4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *Semantic Web*, pages 722–735, 2007.
- [5] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.
- [6] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In *ICML*, pages 549–558, 2018.
- [7] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NeurIPS*, pages 2546–2554, 2011.
- [8] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, pages 2787–2795, 2013.
- [9] G. Bouchard, S. Singh, and T. Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *AAAI Spring Symposium Series*, 2015.
- [10] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Gated feedback recurrent neural networks. In *ICML*, pages 2067–2075, 2015.
- [11] A. Conn, K. Scheinberg, and L. Vicente. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.
- [12] R. Das, A. Neelakantan, D. Belanger, and A. McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *ACL*, pages 132–141, 2017.
- [13] B. Dasgupta and E. Sontag. Sample complexity for learning recurrent perceptron mappings. In *NIPS*, pages 204–210, 1996.
- [14] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2D knowledge graph embeddings. In *AAAI*, 2017.

- [15] T. Elsken, Jan H. Metzen, and F. Hutter. Neural architecture search: A survey. *JMLR*, 20(55):1–21, 2019.
- [16] V. Garg, S. Jegelka, and T. Jaakkola. Generalization and representational limits of graph neural networks. In *ICML*, 2020.
- [17] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *TPAMI*, (6):721–741, 1984.
- [18] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.
- [19] L. Guo, Z. Sun, and W. Hu. Learning to exploit long-term relational dependencies in knowledge graphs. In *ICML*, 2019.
- [20] K. Guu, J. Miller, and P. Liang. Traversing knowledge graphs in vector space. In *EMNLP*, pages 318–327, 2015.
- [21] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.
- [22] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, volume 1, pages 687–696, 2015.
- [23] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NeurIPS*, pages 2016–2025, 2018.
- [24] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.
- [25] T. N Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016.
- [26] T. Lacroix, N. Usunier, and G. Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, 2018.
- [27] N. Lao, T. Mitchell, and W. Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, pages 529–539, 2011.
- [28] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu. Modeling relation paths for representation learning of knowledge bases. Technical report, arXiv:1506.00379, 2015.
- [29] C. Liu, L. Li, X. Yao, and L. Tang. A survey of recommendation algorithms based on knowledge graph embedding. In *CSEI*, pages 168–171, 2019.
- [30] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.
- [31] D. Lukovnikov, A. Fischer, J. Lehmann, and S. Auer. Neural network-based question answering over knowledge graphs on word and character level. In *WWW*, pages 1211–1220, 2017.
- [32] F. Mahdisoltani, J. Biega, and F. M Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2013.
- [33] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur.
- [34] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. *JMLR*, 18(1):564–628, 2017.
- [35] R. Pascanu and Y. Bengio. Revisiting natural gradient for deep networks. Technical report, arXiv:1301.3584, 2013.
- [36] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *ICLR*, 2017.
- [37] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *ICML*, pages 4095–4104, 2018.
- [38] H. Ren, W. Hu, and J. Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *ICLR*, 2020.
- [39] A. Sadeghian, M. Armandpour, P. Ding, and D. Wang. DRUM: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, pages 15321–15331, 2019.
- [40] M. Schlichtkrull, T. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, pages 593–607, 2018.

- [41] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. In *ICLR*, 2020.
- [42] D. So, Q. Le, and C. Liang. The evolved transformer. In *ICML*, pages 5877–5886, 2019.
- [43] R. Socher, D. Chen, C. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, pages 926–934, 2013.
- [44] Z. Sun, Z. Deng, J. Nie, and J. Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019.
- [45] Z. Sun, W. Hu, Q. Zhang, and Y. Qu. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*, pages 4396–4402, 2018.
- [46] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *CISCA*, 2012.
- [47] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114, 2019.
- [48] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In *Workshop on CVSMC*, pages 57–66, 2015.
- [49] T. Trouillon, C. Dance, E. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. Knowledge graph completion via complex tensor factorization. *JMLR*, 18(1):4735–4772, 2017.
- [50] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. Composition-based multi-relational graph convolutional networks. *ICLR*, 2020.
- [51] H. Wang, H. Ren, and J. Leskovec. Entity context and relational paths for knowledge graph completion. Technical report, arXiv:1810.13306, 2020.
- [52] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 29(12):2724–2743, 2017.
- [53] Z. Wang, Q. Lv, X. Lan, and Y. Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *EMNLP*, pages 349–357, 2018.
- [54] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019.
- [55] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.
- [56] Q. Yao and M. Wang. Taking human out of learning applications: A survey on automated machine learning. Technical report, arXiv:1810.13306, 2018.
- [57] Q. Yao, J. Xu, W. Tu, and Z. Zhu. Efficient neural architecture search via proximal iterations. In *AAAI*, 2020.
- [58] R. Ye, X. Li, Y. Fang, H. Zang, and M. Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In *IJCAI*, pages 4135–4141, 2019.
- [59] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.
- [60] C. Zhang, Y. Li, N. Du, W. Fan, and P. Yu. On the generative discovery of structured medical knowledge. In *KDD*, pages 2720–2728, 2018.
- [61] Y. Zhang, Q. Yao, W. Dai, and L. Chen. AutoSF: Searching scoring functions for knowledge graph embedding. In *ICDE*, pages 433–444, 2020.
- [62] H. Zhu, R. Xie, Z. Liu, and M. Sun. Iterative entity alignment via joint knowledge embeddings. In *IJCAI*, pages 4258–4264, 2017.
- [63] B. Zoph and Q. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

## A Supplementary Details

### A.1 Details of the search space

**Operators** Given two input vectors  $\mathbf{a}$  and  $\mathbf{b}$  with  $d$ -dimensions and  $d$  is an even number. Two basic combinators are (a). adding (+):  $\mathbf{o}_i = \mathbf{a}_i + \mathbf{b}_i$ ; and (b).multiplying ( $\odot$ ):  $\mathbf{o}_i = \mathbf{a}_i \cdot \mathbf{b}_i$ . In order to cover ComplEx [49] in the search space, we use Hermitian product ( $\otimes$ ) as in [49]

$$\mathbf{o}_i = \begin{cases} \mathbf{a}_i \cdot \mathbf{b}_i - \mathbf{a}_{i+d/2} \cdot \mathbf{b}_{i+d/2} & \text{if } i < d/2 \\ \mathbf{a}_{i-d/2} \cdot \mathbf{b}_i - \mathbf{a}_i \cdot \mathbf{b}_{i-d/2} & \text{otherwise} \end{cases}$$

As for the gated unit, we define it as  $\mathbf{o}_i = \mathbf{g}_i \cdot \mathbf{a}_i + (1 - \mathbf{g}_i) \cdot \mathbf{b}_i$ , where the gate function  $\mathbf{g} = \text{sigmoid}(\mathbf{W}_a \mathbf{a} + \mathbf{W}_b \mathbf{b})$  with trainable parameters  $\mathbf{W}_a, \mathbf{W}_b \in \mathbb{R}^{d \times d}$ , to imitate the gated recurrent function.

**Space size** Based on the details in Table 2, the inputs of  $O_r$  and  $O_v$  are chosen from  $\mathbf{h}_{t-1}, O_s, \mathbf{0}, \mathbf{s}_t$ . Then the three operators  $O_s, O_r, O_v$  select one combinator from +,  $\odot$ ,  $\otimes$  and gated. Thus, we have  $4^2 \times 4^3 = 1024$  architectures in the macro space  $\mathcal{A}_1$ . For the micro space, we only use activation functions after  $O_s, O_r$  since we empirically observe that activation function on  $O_v$  is bad for the embedding spaces. In this way, the size of micro space  $\mathcal{A}_2$  is  $3^2 \times 2^6 = 576$ . Totally, there should be  $6 \times 10^5$  candidates with the combination of macro-level space and micro-level space.

### A.2 Details of the search algorithm

**Compare with existing NAS problem.** In Section 3.2, we have already talked about the problems of existing search algorithms and propose a hybrid-search algorithm that can search fast and accurate. Here, we give an overview of the different NAS methods in Table 7.

Table 7: Comparison of state-of-the-art NAS methods for general RNN with the proposed Interstellar.

	Existing NAS Algorithms		Interstellar
	Stand-alone approach	One-shot approach	
space	complex structures	micro cells	KG-specific
algorithm	reinforcement learning [63], Bayes optimization [7, 23]	direct gradient descent [30], stochastic relaxation [2, 54]	natural policy gradient
evaluation	full model training	parameter-sharing	hybrid

**Implementation of  $\mathcal{M}(f(\mathbf{F}^*, \alpha), \mathcal{G}_{\text{val}})$ .** For both tasks, the training samples are composed of relational paths. But the validation data format is different. For the entity alignment task, the validation data is a set of entity pairs, i.e.  $\mathcal{S}_{\text{val}} = \{(e_1, e_2) | e_1 \in \mathcal{E}_1, e_2 \in \mathcal{E}_2\}$ . The performance is measured by the cosine similarity of the embeddings  $\mathbf{e}_1$  and  $\mathbf{e}_2$ . Thus, the architecture  $\alpha$  is not available in  $\mathcal{M}(f(\mathbf{F}^*, \alpha), \mathcal{G}_{\text{val}})$ . Instead, we use the negative loss function on a training mini-batch  $-\mathcal{L}(f(\mathbf{F}^*, \alpha), \mathcal{B}_{\text{tra}})$  as an alternative of  $\mathcal{M}(f(\mathbf{F}^*, \alpha), \mathcal{G}_{\text{val}})$ . For the link prediction task, the validation data is set of single triplets, i.e. 1-step paths. Different from the entity alignment task, the path is available for validation measurement here. Therefore, once we sample a mini-batch  $\mathcal{B}_{\text{val}}$  from  $\mathcal{G}_{\text{val}}$ , we can either use the loss function  $-\mathcal{L}(f(\mathbf{F}^*, \alpha), \mathcal{B}_{\text{val}})$  or the direct evaluation metric (MRR or Hit@ $k$ )  $\mathcal{M}(f(\mathbf{F}^*, \alpha), \mathcal{B}_{\text{val}})$  on the mini-batch  $\mathcal{B}_{\text{val}}$ .

**Implementation of natural policy gradient.** Recall that the optimization problem of architectures is changed from optimizing  $\alpha$  by  $\alpha^* = \arg \max_{\alpha \in \mathcal{A}} \mathcal{M}(f(\mathbf{F}^*; \alpha), \mathcal{G}_{\text{val}})$  to optimizing the distribution parameter  $\theta$  by  $\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{\alpha \sim p_{\theta}(\alpha)} [\mathcal{M}(f(\mathbf{F}^*; \alpha), \mathcal{G}_{\text{val}})]$ . As discussed in Section 3.2.2, we use the natural policy gradient to solve  $\theta^* = \arg \max_{\theta} J(\theta)$ . To begin with, we need to define the distribution  $p_{\theta}$ . Same as [2, 34], we refer to the exponential family  $h(\alpha) \cdot \exp(\eta(\theta)^\top T(\alpha) - A(\theta))$ , where  $h(\alpha)$ ,  $T(\alpha)$  and  $A(\theta)$  are known functions depending on the target distribution. The benefit of using exponential family is that the inverse Fisher information matrix  $\mathbf{H}^{-1}(\theta)$  is easily obtained. For simplicity, we set  $h(\alpha) = 1$  and choose the expectation

parameters  $\theta = \mathbb{E}_{p_\theta}[T(\alpha)]$  as in [19]. Then the gradient reduces to  $\nabla_\theta \ln(p_\theta(\alpha)) = T(\alpha) - \theta$ , and the inverse Fisher information matrix  $\mathbf{H}^{-1}(\theta) = \mathbb{E}_{p_\theta}[(T(\alpha) - \theta)(T(\alpha) - \theta)^\top]$  is computed with the finite approximation. The iteration is,

$$\theta_{t+1} = \theta_t + \rho \frac{1}{m} \sum_{i=1}^m \left[ \left( T(\alpha^{(i)}) - \theta_t \right) \left( T(\alpha^{(i)}) - \theta_t \right)^\top \cdot \mathcal{M} \left( f(\mathbf{F}^{(i)*}; \alpha^{(i)}), \mathcal{G}_{\text{val}} \right) \right],$$

where  $\alpha^{(i)}$ 's are sampled from  $p_{\theta_t}$ . In this paper, we set the finite sample  $m$  as 2 to save time. Since the architecture parameters are categorical, we model  $p_\theta$  as categorical distributions. For a component with  $K$  categories, we use a  $K$  dimensional vector  $\theta$  to model the distribution  $p_\theta$ . The probability of sampling the  $i$ -th category is  $\theta_i$  and  $\theta_K = 1 - \sum_{j=1}^{K-1} \theta_j$ . Note that, the NG is only used to update the first  $K - 1$  dimension of  $\theta$ , and  $T(\alpha_i)$  is a one-hot vector (without dimension  $K$ ) of the category  $\alpha_i$ . For more details, please refer to Section 2.4 in [19] and the published code.

### A.3 Training details

Following [18, 19], we sample paths from biased random walks. Take Figure 6 as an example and the path walked from  $e_0$  to  $e_1$  just now. In conventional random walk, all the neighbors of  $e_1$  have the same probability to be sampled as the next step. In biased random walk, we sample the neighbor that can go deeper or go to another KG with larger probability. For single KG, we use one parameter  $\alpha \in (0.5, 1)$  to give the depth bias. Specifically, the neighbors that are two steps away from the previous entity  $e_0$ , like  $e_2$  and  $e_4$  in Figure 6, have the bias of  $\alpha$ . And the bias of others, i.e.  $e_0, e_3$ , are  $1 - \alpha$ . Since  $\alpha > 0.5$ , the path is more likely to go deeper. Similarly, we have another parameter  $\beta \in (0.5, 1)$  for two KGs to give cross-KG bias. Assume  $e_0, e_1$ , and  $e_3$  belongs to the same KG  $\mathcal{G}_1$  and  $e_2, e_4$  are parts of  $\mathcal{G}_2$ . Then, the next step is encouraged to jump to entities in another KG, namely to the entities  $e_2$  and  $e_4$  in Figure 6. Since the aligned pairs are usually rare in the training set, encouraging cross-KG walk can learn more about the aligned entities in the two KGs.

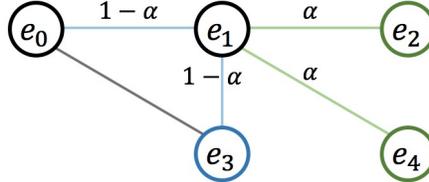


Figure 6: An example of the biased random walks.

Since the KGs are usually large, we sample two paths for each triplet in  $\mathcal{S}_{tra}$ . The length of paths is 7 for entity alignment task on two KGs and 3 for link prediction task on single KG. The paths are fixed for each dataset to keep a fair comparison among different models. The parameters used for sampling path are summarized in Table 8.

Table 8: Parameter used for sampling path.

parameters	$\alpha$	$\beta$	length
entity alignment	0.9	0.9	7
link prediction	0.7	-	3

Once the paths are sampled, we start training based on the relational path. The loss for a path with length  $L$ , i.e. containing  $L$  triplets, is given in given as

$$\mathcal{L}_{tra} = \sum_{t=1}^L \left\{ -\mathbf{v}_t \cdot \mathbf{o}_t + \log \left( \sum_{o_i \in \mathcal{E}} \exp(\mathbf{v}_t \cdot \mathbf{o}_i) \right) \right\} \quad (4)$$

In the recurrent step  $t$ , we focus on one triplet  $(s_t, r_t, o_t)$ . The subject entity embedding  $\mathbf{s}_t$  and relation embedding  $\mathbf{r}_t$  are processed along with the proceeding information  $\mathbf{h}_{t-1}$  to get the output  $\mathbf{v}_t$ . The output  $\mathbf{v}_t$  is encouraged to approach the object entity embedding  $\mathbf{o}_t$ . Thus, the objective in (4) can be regarded as a multi-class log-loss [26] where the object  $o_t$  is the true label.

Besides, the training of the parameters  $F$  is based on mini-batch gradient descent. Adam [24] is used as the optimization method for updating model parameters.

#### A.4 Data statistics

The datasets used in entity alignment task are the customized version by [19]. For the normal version data, nodes are sampled to approximate the degree distribution of original KGs. This makes the datasets in this version more realistic. For the dense version data, the entities with low degrees are randomly removed in the original KGs. This makes the data more similar to those used by existing methods [45, 53]. We give the statistics in Table 9.

Table 9: Statistics of the datasets we used for entity alignment. Each single KG contains 15,000 entities. There are 4,500 aligned entity pairs in the training sets, and 11,500 pairs for evaluation. The first 10% pairs among the 11,500 are used for validation and the left for testing. We use “#” short for “number of”.

version	data source	DBP-WD		DBP-YG		EN-FR		EN-DE	
		DBpedia	Wikidata	DBpedia	YAGO3	English	French	English	German
Normal	# relations	253	144	219	30	211	177	225	118
	# triplets	38,421	40,159	33,571	34,660	36,508	33,532	38,281	37,069
Dense	# relations	220	135	206	30	217	174	207	117
	# triplets	68,598	75,465	71,257	97,131	71,929	66,760	56,983	59,848

The datasets used for the link prediction task are used in many baseline models [14, 44, 19]. The WN18-RR and FB15k-237 datasets are created so that the link leakage problem [14, 49] can be solved. Thus, they are more realistic than their super-set WN18 and FB15k [8].

Table 10: Statistics of the datasets used for link prediction.

Dataset	#entity	#relations	#train	#valid	#test
WN18-RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466
YAGO3-10	123,188	37	1,079,040	5,000	5,000

#### A.5 Hyper-parameters and Training Details

During searching, the meta hyper-parameters  $k_1$  is one step and  $k_2$  is one epoch based on the efficiency consideration.

In order to find the hyper-parameters during search procedure, we use RSN [19] as a standard baseline to tune the hyper-parameters. We use the *HyperOpt* package with tree parzen estimator [7] to search the learning rate  $\eta$ , L2 penalty  $\lambda$ , decay rate  $u$ , batch size  $m$ , as well as a dropout rate  $p$ . The decay rate is applied to the learning rate each epoch and the dropout rate is applied to the input embeddings. The tuning ranges are given in Table 11.

Table 11: Searching range of hyper-parameters

hyper-param	ranges
$\eta$	$[10^{-5}, 10^{-3}]$
$\lambda$	$[10^{-5}, 10^{-2}]$
$u$	$[0.98, 1]$
$m$	$\{128, 256, 512, 1024, 2048\}$
$p$	$[0, 0.6]$

The embedding dimension for entity alignment task is 256, and for link prediction is 64 during searching. After the hyper-parameter tuning, the proposed Interstellar starts searching with this hyper-parameter setting. For all the tasks and datasets, we search and evaluate 100 architectures. Among them, we select the best architecture indicated by the Hit@1 performance on validation set, i.e. the architecture with top 1 performance in the curve of Interstellar in Figure 5. After searching, the embedding size on link prediction is increased to be 256 and we fine-tune the other hyper-parameters in Table 11 again for the searched architectures. Finally, the performance on testing data is reported in Table 4. The time cost of searching and fine-tuning is given in Table 6 in Section 4.5.

## B Supplementary Experiments

### B.1 Entity alignment on the Dense version

Table 12 compares the testing performance of the models searched by Interstellar and human-designed ones on the *Dense* version datasets [19]. In this version, there is no much gap between the triplet-based and GCN-based models since more data can be used for learning the triplet-based ones. In comparison, the path-based models are better by traversing across two KGs. Among them, Interstellar performs the best on DBP-WD, EN-FR and EN-DE and is comparable with RSN on DBP-YG dataset. Besides, the searched architectures are different between the Normal version and Dense version, as will be illustrated Appendix C.3.

Table 12: Performance comparison on entity alignment tasks on dense dataset.  $H@k$  is short for Hit@ $k$ . The results of TransD [22], BootEA [45], IPTransE [62], GCN-Align [53] and RSN [19] are copied from [19].

models		DBP-WD			DBP-YG			EN-FR			EN-DE		
		H@1	H@10	MRR									
triplet	TransE	54.9	75.5	0.63	33.8	68.5	0.46	22.3	39.8	0.28	30.2	60.2	0.40
	TransD*	60.5	86.3	0.69	62.1	85.2	0.70	54.9	86.0	0.66	57.9	81.6	0.66
	BootEA*	67.8	91.2	0.76	68.2	89.8	0.76	64.8	91.9	0.74	66.5	87.1	0.73
GCN	GCN-Align	43.1	71.3	0.53	31.3	57.5	0.40	37.3	70.9	0.49	32.1	55.2	0.40
	VR-GCN	39.1	73.1	0.51	28.5	59.3	0.39	35.3	69.6	0.47	31.5	61.1	0.41
	R-GCN	24.1	61.4	0.36	31.6	57.3	0.40	25.7	59.0	0.37	28.6	52.4	0.37
path	PTransE	61.6	81.7	0.69	62.9	83.2	0.70	38.4	80.2	0.52	37.7	65.8	0.48
	IPTransE*	43.5	74.5	0.54	23.6	51.3	0.33	42.9	78.3	0.55	34.0	63.2	0.44
	Chains	64.9	88.3	0.73	71.3	91.8	0.79	70.6	92.6	0.78	70.7	86.4	0.76
	RSN*	76.3	92.4	0.83	<b>82.6</b>	95.8	<b>0.87</b>	75.6	92.5	0.82	73.9	89.0	0.79
<b>Interstellar</b>		<b>77.9</b>	<b>94.1</b>	<b>0.84</b>	81.8	<b>96.2</b>	<b>0.87</b>	<b>77.3</b>	<b>94.7</b>	<b>0.84</b>	<b>75.3</b>	<b>90.3</b>	<b>0.81</b>

### B.2 Problem under parameter-sharing

In this part, we empirically show the problem under parameter-sharing in the full search space  $\mathcal{A}$  and how the micro-level space  $\mathcal{A}_2$  solves this problem. Following [6, 59], we train the supernet in Figure 2 with parameters-sharing and 1) randomly sample 100 architectures from  $\mathcal{A}$  (one-shot); 2) re-train these sampled architectures from scratch (standard-alone). If the top-performed architectures selected by one-shot approach also perform well by the stand-alone approach, then parameter-sharing works here and vice the visa. Based on the discussion in Appendix A.2, we use the loss on validation mini-batch as the measurement in the one-shot approach for entity alignment task. And we use the metric, i.e. Hit@1, on validation mini-batch as the measurement in the stand-alone approach for link prediction task. Figure 7 shows such a comparison in the full space  $\mathcal{A}$ . As shown in the upper-right corner of each figure, the top-performed architectures in one-shot approach do not necessarily perform the best in stand-alone approach. Therefore, searching architectures by one-shot approach in the full search space does not work since parameter-sharing does not work here.

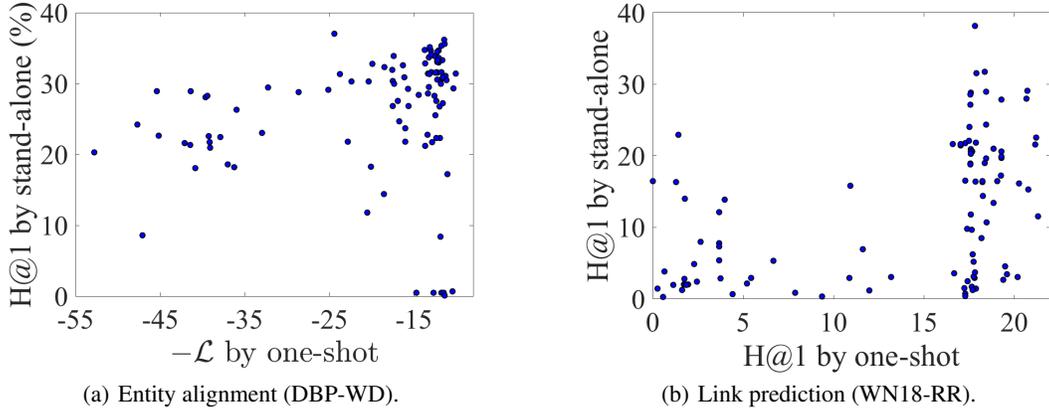


Figure 7: Comparison of models trained with one-shot model and stand-alone model on the full search space.

In order to take advantage of the efficiency in one-shot approach, we split the search space into a macro-level space  $\mathcal{A}_1$  and micro-level space  $\mathcal{A}_2$ . To show how parameter-sharing works in the micro-level space, we use the best macro-level architecture  $\alpha_1^*$  during macro-level searching, and sample 100 architectures  $\alpha_2 \in \mathcal{A}_2$  to get the architecture  $\alpha = [\alpha_1^*, \alpha_2]$ . As shown in Figure 8, the top performed architectures in one-shot approach also perform the best in stand-alone approach. This observation verifies that 1) parameter-sharing is more likely to work in a simpler search space as discussed in [6, 37, 59]; 2) the split of the macro and micro level space in our problem is reasonable; 3) searching architectures by the Hybrid-search algorithm takes the advantage of the stand-alone approach and the one-shot approach.

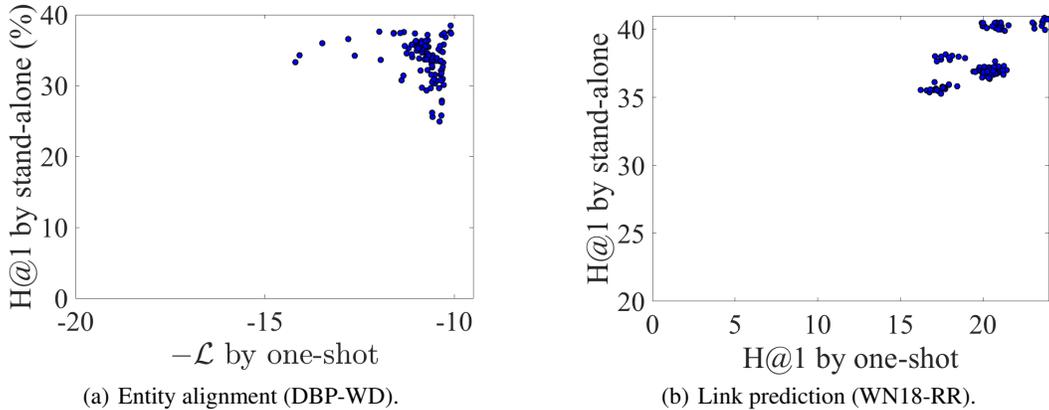


Figure 8: Comparison of models trained with one-shot model and stand-alone model on the micro-level search space.

### B.3 Influence of path length

In this part, we show the performance of the best architecture on each datasets with varied maximum length of the relational path in Figure 9. For entity alignment task, we vary the length from 1 to 10 for DBP-WD datasets (normal version); for link prediction task, we vary the length from 1 to 7 for WN18-RR datasets. We plot them in the same figure to make better comparison. As shown, when path length is 1, the model performs very bad. However, it has little influence on WN18-RR. This verifies that long-term information is important for the entity alignment task while short-term information is important for link prediction task. For the other length, the performance is quite stable.

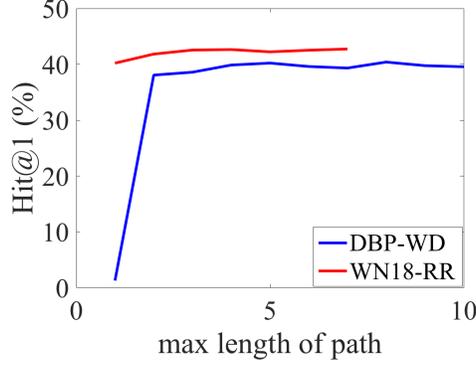


Figure 9: The influence of path length.

## C Searched Models

### C.1 Architectures in literature

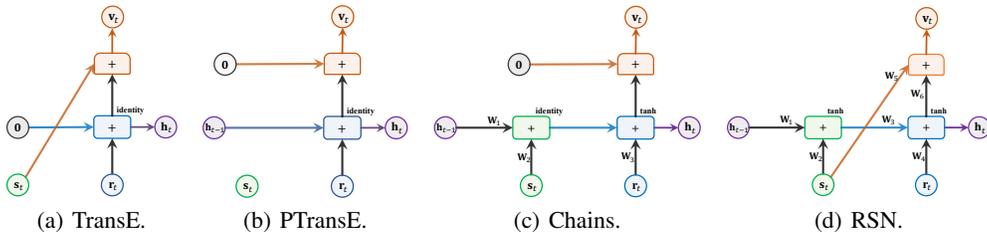


Figure 10: Graphical illustration of TransE, PTransE, Chains, RSN. TransE is a triplet-based model thus it is non-recurrent. PTransE recurrently processes the relational path without the intermediate entities. Chains simultaneously combines the entities and relations along the path, and RSN designs a customized variant of RNN. Each edge is an identity mapping, except an explicit mark of the weight matrix  $W_i$ . Activation functions are shown in the upper right corner of the operators.

### C.2 Countries

We give the graphical illustration of architectures searched in Countries dataset in Figure 11. The search procedure is conducted in the whole search space rather than the four patterns P1-P4 in Figure 3. We can see that in Figure 11, (a) belongs to P1, (b) belongs to P2 and (c) belongs to P4. These results further verify that Interstellar can adaptively search architectures for specific KG tasks and datasets.

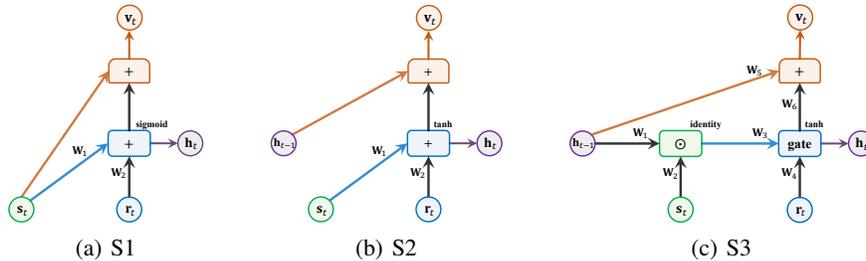


Figure 11: Graphical representation of the searched  $f$  on countries dataset.

### C.3 Entity Alignment

The searched models by Interstellar, which consistently perform the best on each dataset, are graphically illustrated in Figure 12. As shown, all of the searched recurrent networks processing recurrent information in  $h_{t-1}$ , subject entity embedding  $s_t$  and relation embedding  $r_t$  together. They have different connections, different composition operators and different activation functions, even though the searching starts with the same random seed 1234.

More interestingly, the searched architectures in DBP-WD and DBP-YG can model multiple triplets simultaneously. While the architectures in EN-FR and EN-DE models two adjacent triplets each time. The reason is that, in the first two datasets, the KGs are from different sources, thus the distribution is various [19]. In comparison, EN-FR and EN-DE are two cross-lingual datasets, which should have more similar distributions in each KG. Therefore, we need to model the longer term information on DBP-WD and DBP-YG. In comparison, modeling the shorter term information like two triplets together is better for EN-FR and EN-DE. In this way, the model can focus more on learning short-term semantic in a range of two triplets.

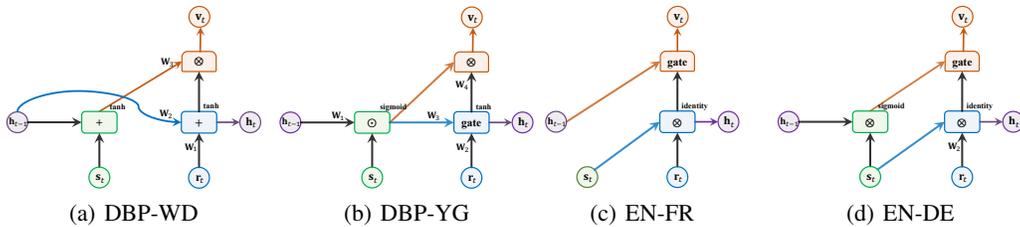


Figure 12: Graphical representation of the searched recurrent network  $f$  on each datasets in entity alignment task (Normal version).

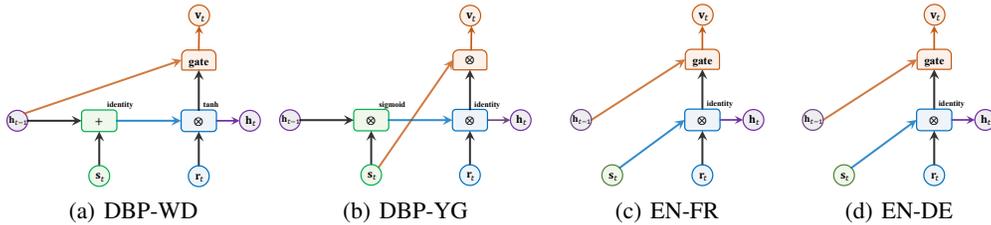


Figure 13: Graphical representation of the searched recurrent network  $f$  on each datasets in entity alignment task (Dense version).

### C.4 Link Prediction

The best architectures searched in link prediction tasks are given in Figure 14. In order to illustrate the models we searched, we make a statistics of the distance, i.e. the shortest path distance when regarding the KG as a simple undirected graph, between two entities in the validation and testing set in Table 13. As shown, most of the triplets have distance less than 3. Besides, as indicated by the performance on the two datasets, we infer that triplets far away from 3-hop are very challenging to model. At least in this task, triplets less or equal than 3 hops are the main focus for different models. This also explains why RSN, which processes long relational path, does not perform well in the link prediction task. The searched models in Figure 14 do not directly consider long-term structures. Instead, the architecture on WN18-RR models one triplet, and the architecture on FB15k-237 focuses on modeling two consecutive triplets. These are consistent with the statistics in Table 13, where WN18-RR has more one-hop and FB15k-237 has more two-hop triplets.

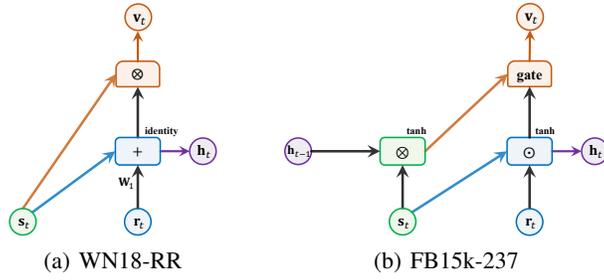


Figure 14: Graphical representation of the searched  $f$  on each datasets in link prediction task.

Table 13: Percentage of the  $n$ -hop triplets in validation and testing datasets.

Datasets		Hops			
		$\leq 1$	2	3	$\geq 4$
WN18-RR	validation	35.5%	8.8%	22.2%	33.5%
	testing	35.0%	9.3%	21.4%	34.3%
FB15k-237	validation	0%	73.2%	26.1%	0.7%
	testing	0%	73.4%	26.8%	0.8%