Search to Pass Messages for Temporal Knowledge Graph Completion

Zhen Wang[1,2] **Haotong Du**[1,2,*] **Quanming Yao**[3,*] **Xuelong Li**[2]
[1]School of Computer Science, Northwestern Polytechnical University, China
[2]School of Artificial Intelligence, Optics and Electronics (iOPEN),
Northwestern Polytechnical University, China
[3]Department of Electronic Engineering, Tsinghua University, China
w-zhen@nwpu.edu.cn, duhaotong@mail.nwpu.edu.cn
qyaoaa@tsinghua.edu.cn, li@nwpu.edu.cn

Abstract

Completing missing facts is a fundamental task for temporal knowledge graphs (TKGs). Recently, graph neural network (GNN) based methods, which can simultaneously explore topological and temporal information, have become the state-of-the-art (SOTA) to complete TKGs. However, these studies are based on hand-designed architectures and fail to explore the diverse topological and temporal properties of TKG. To address this issue, we propose to use neural architecture search (NAS) to design data-specific message passing architecture for TKG completion. In particular, we develop a generalized framework to explore topological and temporal information in TKGs. Based on this framework, we design an expressive search space to fully capture various properties of different TKGs. Meanwhile, we adopt a search algorithm, which trains a supernet structure by sampling single path for efficient search with less cost. We further conduct extensive experiments on three benchmark datasets. The results show that the searched architectures by our method achieve the SOTA performances. Besides, the searched models can also implicitly reveal diverse properties in different TKGs. Our code is released in https://github.com/striderdu/SPA.

1 Introduction

A temporal knowledge graph (TKG) (Cai et al., 2022) is a graph-structural data with many time-sensitive relational facts. The facts can be formed as qudaruples *(subject entity, relationship, object entity, timestamp)*, denoted as (s, r, o, t), e.g., *(FIFA World Cup, is held in, Qatar, 2022)*. TKGs are used extensively in various applications that require the assistance of temporal knowledge such as temporal question answering (Saxena et al., 2021), recommendation systems (Zhao et al., 2022) and mobility prediction (Wang et al., 2021a).

Notably, similar to static KG, most TKGs are inherently incompletion, which seriously hampers their applications in downstream tasks. Therefore, a great number of works focus on TKG completion (TKGC) to infer the missing facts in TKGs. Pioneer embedding-based methods (Leblay and Chekol, 2018; Dasgupta et al., 2018; Goel et al., 2020; Lacroix et al., 2020) directly construct time-aware score functions to evaluate the plausibility of quadruple. However, embedding-based methods do not explicitly encode local graph structures in TKG, which limits their expressiveness.

Recently, based on the success of graph neural networks (GNNs), some GNN-based methods have been proposed to solve TKGC. TeMP (Wu et al., 2020), a typical GNN-based method for TKGC, discretizes a TKG into multiple static KG snapshots and generates dynamic entity representations along two dimensions: structural neighborhoods and temporal dynamics. Structural encoder extracts feature from local node neighborhoods in each snapshot through message passing and aggregation, while temporal encoder captures feature evolution over multiple time steps by sequential models. T-GAP (Jung et al., 2021), views timestamps as properties of links between entities, and proposes the temporal GNN to learn structural and temporal information on the whole graph. GNNs have been demonstrated to achieve better performance for TKGC tasks, due to their powerful expressiveness.

However, these GNN-based methods use the fixed GNN architectures to tackle different TKGs, failing to explore the diverse topological and temporal properties of TKGs, which prevents the model from fully discovering the diverse implicit patterns in different datasets. More recently, BoxTE (Messner et al., 2022) has also pointed out this problem. Therefore, it is critical to design data-specific GNN architectures for TKGC task.

Neural architecture search (NAS) (Yao et al.,

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 6160–6172

2018; Hutter et al., 2019) has achieved great success in designing data-specific architectures, of which the performances exceed the architectures crafted by human experts in various areas, e.g., computer vision (Zhang et al., 2022a), natural language processing (So et al., 2019), and graph learning (Zhang et al., 2021). More recently, in static KG completion, there are some works that adopt NAS techniques for designing the score function (Zhang et al., 2022b) or GNN architecture (Wang et al., 2021b). However, no one has made similar attempts on TKG. And designing data-specific architectures for TKGC task is non-trivial, because of the demand to simultaneously explore topological and temporal information.

In this work, we propose a novel method which tries to Search to PAss messages(SPA), to automatically design data-specific architectures for TKGC. Firstly, we design a generalized framework to simultaneously explore topological and temporal information in TKGs. From this, we define a novel and expressive search space, in which different combinations of operations can capture various patterns of different TKGs. To enable efficient search on top of the search space, we adopt a flexible and effective search algorithm, which trains a supernet by sampling single path uniformly, thus greatly reducing the GPU memory cost. To demonstrate the effectiveness of SPA, we conduct extensive experiments on three benchmark datasets of TKGC. Experimental results show that SPA can consistently achieve state-of-the-art performance by designing data-specific architectures. Further empirical results verify the searched models provide implicitly properties expression for different TKGs.

2 The Proposed Method

As mentioned in the introduction, the GNN-based method for TKGC should be data-specific. Generally, TKGs contain both topological and temporal information. Thus, to design a proper model, we first define a framework which can model topological patterns and temporal contexts jointly. Then, we introduce our novel search space. Finally, we describe our search objective and search algorithm.

2.1 The Generalized Framework

To search for data-specific and well-performing architectures based on GNN, we need to define a framework which has the ability to model topological and temporal information in TKG. Following some existing works (Taheri et al., 2019; Sankar et al., 2020; Manessi et al., 2020; Wu et al., 2020; Gao et al., 2022; Wang et al., 2022), we firstly discretize a TKG into multiple static KG snapshots along the time, and utilize GNNs and sequential models to generate dynamic entity representations. The main advantages of this approach include simplicity as well as enabling the use of a wealth of GNN and sequential model techniques. A large number of works on temporal graphs also achieve competitive results with such this approach consisting of combinations of GNNs and recurrent architectures, whereby the former digest graph information and the latter handle dynamism.

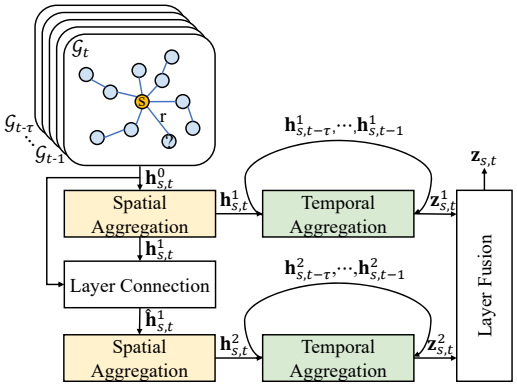


Figure 1: An illustration of the 2-layer framework. The temporal aggregation module is placed after each spatial aggregation module in each layer, and the layer fusion module is utilized to incorporate the intermediate feature representations produced by temporal aggregation module. The layer connection module is used to help the feature reuse for each spatial aggregation.

Based on this motivation, we develop a generalized framework that mainly consists of four key modules for learning expressive dynamic entity representation, including **spatial aggregation**, **temporal aggregation**, **layer connection**, and **layer fusion**. In Figure 1, we use a 2-layer architecture as an illustrative example of the generalized framework. More detailed descriptions of the four modules are as follows:

1. **Spatial Aggregation** at the i-th layer is conducted to aggregate information from the neighbors of s in static snapshot $\mathcal{G}_t$ and results in the intermediate representation of entity s, as follows,

$$\mathbf{h}^1_{s,t} = \mathcal{O}_{\text{SA}}(\mathcal{G}_t, \mathbf{h}^0_s), \tag{1}$$

where $\mathbf{h}^0_s \in \mathbf{H}$ is the initialized embedding of entity s, $\mathbf{H}$ is the representation matrix containing embeddings of entities and relations in TKG.

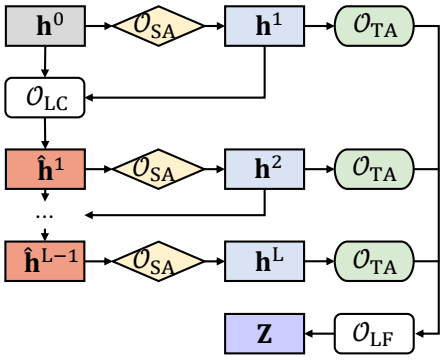


Figure 2: The illustration of search space of SPA.

Module name	Operations
Spatial Aggregation ($\mathcal{O}_{\text{SA}}$)	RGCN, RGAT, COMPGCN
Temporal Aggregation ($\mathcal{O}_{\text{TA}}$)	GRU, SA, IDENTITY
Layer Connection ($\mathcal{O}_{\text{LC}}$)	LC_SKIP, LC_SUM, LC_CONCAT
Layer Fusion ($\mathcal{O}_{\text{LF}}$)	LF_MAX, LF_CONCAT, LF_SKIP, LF_MEAN

Table 1: The operations used in our search space.

2. **Temporal Aggregation** at the i-th layer generates temporal feature $\mathbf{z}_{s,t}^i$ based historical feature sequences $\mathbf{h}_{s,t-\tau}^i, \cdots, \mathbf{h}_{s,t-1}^i$ behind, as follows,

$$\mathbf{z}_t^1 = \mathcal{O}_{\text{TA}}(\mathbf{h}_{s,t-\tau}^1, \cdots, \mathbf{h}_{s,t-1}^1, \mathbf{h}_{s,t}^1), \quad (2)$$

where τ is a hyper-parameter, stands for the number of input KG snapshots to the model.

3. **Layer Connection** combines $\mathbf{h}_{s,t}^{i-1}$ with $\mathbf{h}_{s,t}^i$ to form a new representation $\hat{\mathbf{h}}_{s,t}^i$, as follows,

$$\hat{\mathbf{h}}_{s,t}^1 = \mathcal{O}_{\text{LC}}(\mathbf{h}_s^0, \mathbf{h}_{s,t}^1). \quad (3)$$

4. **Layer Fusion** generates the final representation of entity $\mathbf{z}_{s,t}$ by fusing temporal features from temporal aggregation module in different layer, as follows,

$$\mathbf{z}_{s,t} = \mathcal{O}_{\text{LF}}(\mathbf{z}_{s,t}^1, \mathbf{z}_{s,t}^2). \quad (4)$$

Based on this generalized framework, we can search the specific form of each operation to obtain data-specific architecture. An effective search space can be naturally designed by including human-designed operations, the details of which are given in Table 1.

2.2 Search Space

Based on above framework, we design one novel search space with a set of candidate operations as shown in Table 1. In the following, we will describe the details of these operations.

Spatial Aggregation. We choose three widely used multi-relational GNNs as alternative spatial aggregation module: RGCN (Schlichtkrull et al., 2018), RGAT (Busbridge et al., 2019), CompGCN (Vashishth et al., 2020), which denoted as RGCN, RGAT, COMPGCN.

Temporal Aggregation. For the temporal aggregation module, we consider two sequential models to learn temporal patterns: GRU (Cho et al., 2014) , Self-Attention (SA) (Vaswani et al., 2017). Besides, we incorporate the operation IDENTITY, which means using the results of spatial aggregation directly, i.e., $\mathbf{z}_{s,t}^i = \mathbf{h}_{s,t}^i$, rather than learning dynamic feature between snapshots.

Layer Connection. It has been well proven in many literatures (Li et al., 2021) that the use of skip connections between spatial aggregation modules can help alleviate over-smoothing and the vanishing gradient issue, and improve the performance of the model. In our search space, we add three different skip connection operations to encourage various feature reuse, i.e., LC_SKIP, LC_SUM, LC_CONCAT.

Layer Fusion. In static graph learning, some studies (Xu et al., 2018a) focus on obtaining more expressive structure-aware representation by selectively fusing the intermediate representation of spatial aggregation. We borrow this idea to temporal graph learning and provide four fusion operations to integrate the representations of the intermediate temporal aggregation layers with the average, maximum, concatenation and skip, denoted as LF_MEAN, LF_MAX, LF_CONCAT and LF_SKIP, respectively. The search for various fusion operations allows the model to learn to adapt to different dynamic subgraph structures.

An example of the L-layer search space is shown in Figure 2. With so many candidate architectures in the search space, SPA can use efficient search algorithm to obtain data-specific architectures beyond existing human-designed ones.

2.3 Search Objective

Let the training and validation set be $\mathcal{D}_{\text{tra}}$ and $\mathcal{D}_{\text{val}}$, $\mathcal{N}(\mathbf{W}_{\Theta,\mathbf{H}}; \boldsymbol{\alpha})$ be a TKGC model (where $\mathbf{W}_{\Theta,\mathbf{H}}$

represents model parameters containing model weights Θ and TKG embedding $\mathbf{H}$, and $\boldsymbol{\alpha}$ is the model architecture), $\mathcal{M}$ be the measurement on $\mathcal{D}_{\text{val}}$ and $\mathcal{L}$ be the loss on $\mathcal{D}_{\text{tra}}$. The problem is defined to find an architecture $\boldsymbol{\alpha}$ such that validation performance is maximized, i.e.,

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}\in\mathcal{A}} \ \mathcal{M}(\mathcal{N}(\mathbf{W}^*_{\Theta,\mathbf{H}}; \boldsymbol{\alpha}), \mathcal{D}_{\text{val}}), \tag{5}$$

$$\text{s.t. } \mathbf{W}^*_{\Theta,\mathbf{H}} = \arg\min_{\mathbf{W}} \mathcal{L}(\mathcal{N}(\mathbf{W}_{\Theta,\mathbf{H}}; \boldsymbol{\alpha}), \mathcal{D}_{\text{tra}}), \tag{6}$$

which is a bi-level optimization problem and is non-trivial to solve. Because the computation cost to get the optimal parameters $\mathbf{W}^*_{\Theta,\mathbf{H}}$ is generally high. And the search space is large. Thus, how to efficiently search the architectures is a big challenge.

To perform TKGC task, we use score function to measure the plausibility of each candidate quadruple (s, r, o, t). Since our proposed framework can generate time-aware entity embeddings, we only need static score function.

Specifically, the score function for quadruple is defined as follows in SPA:

$$\phi(s, r, o, t) = f(\boldsymbol{z}_{s,t}, \boldsymbol{h}_r, \boldsymbol{z}_{o,t}), \tag{7}$$

where $\boldsymbol{z}_{s,t}$ and $\boldsymbol{z}_{o,t}$ are time-aware representaions for subject and object entities, while $\boldsymbol{z}_r$ is a learned embedding of the relation r. In this work, we use ComplEx (Trouillon et al., 2016) as the score function, which is known to perform well on static KGC benchmarks.

For the loss function, following the setting of TeMP, we employ the cross-entropy loss for parameter learning. More details about loss function is in the Appendix A.1.

2.4 Search Algorithm

Based on the proposed framework and the search space, the search algorithm is used to search operations from the corresponding operation set.

Inspired by recent advances in NAS, we propose to solve Equation (5), (6) using one-shot NAS paradigm, which greatly improves the efficiency of performance estimation by training only one supernet.

There are two types of methods in one-shot NAS: the single-stage method and the two-stage method. The first one combines supernet training and search in a *single stage*. Representative methods include DARTS (Liu et al., 2019), SNAS (Xie et al., 2019), etc. The single-stage approach requires that the validation metrics be differentiable to allow supernet training and architecture search to be jointly optimized by gradient-based methods, which is inappropriate for our task as its metric(i.e. MRR) is non-differentiable. And the correlation between the validation loss and the validation metric is unclear. Using the validation loss to update the architecture parameters may mislead the search algorithm to find a sub-optimal architecture.

Moreover, the single-stage approach requires training the whole supernet, which demands tremendous GPU memory as the proposed search space contains spatial encoder and temporal encoder. Hence, we adopt the two-stage approach, which decouples supernet training and architecture search.

In this paper, we adopt SPOS (Guo et al., 2020), a typical two-stage method, as it can consume the GPU memory less and fully train each candidate operation. Algorithm 1 delineates the full procedure.

Supernet Training. For the solution of Equation (6), following SPOS, we construct a supernet structure that each candidate architecture is a single path. In each step of optimization, as shown

Algorithm 1 SPA - Search to PAss messages

Require: Training dataset $\mathcal{D}_{\text{tra}}$, validation dataset $\mathcal{D}_{\text{val}}$, the epoch T_1 for train supernet, the epoch T_2 for search architecture, the search space $\mathcal{A}$.
Ensure: The searched architecture.

```
Random initialize the parameter of supernet W.
while t < T_1 do
    for each minibatch B ∈ D_train do
        Random sample α from A.
        Calculate the training loss L_tra for α.
        Update weight subset W_{Θ,H}(α) with L_tra.
    end for
end while
while t < T_2 do
    Random sample α from A.
    Inherit weight subset W*_{Θ,H}(α) from W*_{Θ,H}.
    Calculate the validation performance for α in D_val.
end while
return The searched architecture with the highest validation performance.
```

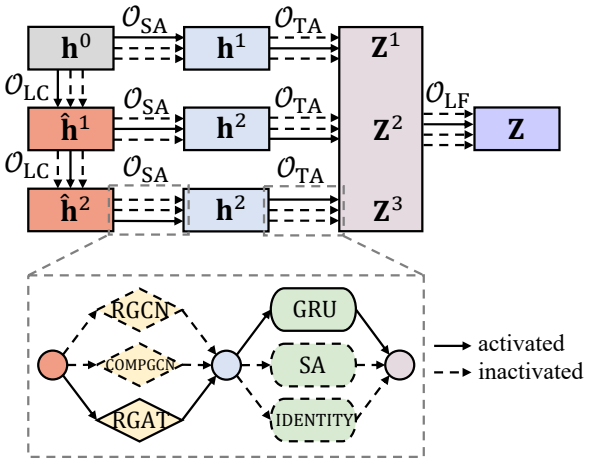


Figure 3: The illustration of the single path supernet. In the training stage, the weights of the solid line part (RGAT, GRU) are activated and updated, the dotted portions are masked and inactivated.

in Figure 3. an architecture $\boldsymbol{\alpha}$ (one path, i.e., the solid part of the figure) is sampled from the search space in a uniformly distributed manner. It guarantees equal expectations of the number of times each architecture is sampled, thus all architectures (and their weights) are trained fully and equally. And then, only the weights corresponding $\boldsymbol{\alpha}$ are activated and updated. So the GPU memory usage is efficient.

Architecture Search. After getting the trained optimal weights of supernet $\mathbf{W}^*_{\Theta,\mathbf{H}}$, to solve the problem in Equation (5), we leverage random search to find well-performing architecture $\boldsymbol{\alpha}$. This is simple but effective for our search space.

Finally, architecture with the highest validation performance(i.e. validation MRR) in all iterations will be returned.

3 Experiments

3.1 Experimental Settings

Datasets. We perform evaluation on three widely used TKG completion datasets, including ICEWS14 (García-Durán et al., 2018), ICEWS05-15 (García-Durán et al., 2018) and GDELT (Leetaru and Schrodt, 2013). ICEWS14 and ICEWS05-15 are two subsets of *Integrated Crisis Early Warning System* (ICEWS) database with different time spans. GDELT is a subset of *Global Database of Events, Language, and Tone* (GDELT), which contains facts facts from April 1, 2015 to March 31, 2016. The detailed dataset statistics is presented in the Appendix A.2.

Evaluation Metrics. We follow (Bordes et al., 2013) to use the filtered ranking-based metrics, i.e., mean reciprocal ranking (MRR) and Hit@1/3/10 for evaluation. For both metrics, the larger value indicates the better performance.

Baseline Methods. We compare SPA with two types of baselines: human-designed methods and NAS methods.

For human-designed methods, we take TransE (Bordes et al., 2013), Distmult (Yang et al., 2015), ComplEx (Trouillon et al., 2016) and SimplE (Kazemi and Poole, 2018) to represent static KG completion methods, and TTransE (Leblay and Chekol, 2018), TA-Distmult (García-Durán et al., 2018), HyTE (Dasgupta et al., 2018), DE-SimplE (Goel et al., 2020), TNTComplEx (Lacroix et al., 2020), ChronoR (Sadeghian et al., 2021), TeLM (Xu et al., 2021) and BoxTE (Messner et al., 2022) to represent state-of-the-art embedding-based methods designed for TKGC. For the GNN-based methods, we compare with both TeMP (Wu et al., 2020) and T-GAP (Jung et al., 2021) here.

For NAS methods, since existing methods cannot learn the data-specific architecture for temporal graph, we further provide Random search as the baseline for comparisons based on the proposed search space in Section 2.2.

Implementation and Hyperparameters. For all NAS methods (Random baseline and SPA), we derived the candidate GNNs from the search space in the search process. All the searched candidates are tuned individually with hyperparameters like learning rate, weight decay, etc. In this paper, the 3-layer framework is empirically chosen for all NAS methods on all datasets. We set the negative sampling ratio to 500, i.e. 500 negative samples per positive triple. More details about the implementation and hyperparamters are given in Appendix A.3.

3.2 Performance Comparison

Table 2 shows the overall result on three benchmarks. As can be seen, there is no clear winner among the human-designed baselines on all datasets. Besides, we can see that SPA consistently outperforms all baselines on all datasets, which demonstrates the effectiveness of SPA on searching for data-specific architectures for TKGC.

When it comes to NAS baselines, the performance gains of SPA are also significant. On one hand, the Random baselines achieve considerable performance gains on all these datasets, which

Type	Model	ICEWS14				ICEWS05-15				GDELT			
		MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
Human-designed	TransE	0.326	15.4	43.0	64.4	0.330	15.2	44.0	66.0	0.155	6.0	17.8	33.5
	DistMult	0.441	32.5	49.8	66.8	0.457	33.8	51.5	69.1	0.210	13.3	22.4	36.5
	ComplEx	0.442	40.0	43.0	66.4	0.464	34.7	52.4	69.6	0.213	13.3	22.5	36.6
	SimplE	0.458	34.1	51.6	68.7	0.478	35.9	53.9	70.8	0.206	12.4	22.0	36.6
	TTransE	0.255	7.4	-	60.1	0.271	8.4	-	61.6	0.115	0.0	16.0	31.8
	HyTE	0.297	10.8	41.6	65.5	0.316	11.6	44.5	68.1	0.118	0.0	16.5	32.6
	TA-DistMult	0.477	36.3	-	68.6	0.474	34.6	-	72.8	0.206	12.4	21.9	36.5
	DE-SimplE	0.526	41.8	59.2	72.5	0.513	39.2	57.8	74.8	0.230	14.1	24.8	40.3
	TNTComplEx	0.620	52.0	66.0	76.0	0.670	59.0	71.0	81.0	-	-	-	-
	TIMEPLEX	0.604	51.5	-	77.1	0.640	54.5	-	81.8	-	-	-	-
	ChronoR	0.625	**54.7**	66.9	77.3	0.675	59.6	72.3	82.0	-	-	-	-
	TeLM	0.625	54.5	67.3	77.4	0.678	**59.9**	72.8	82.3	-	-	-	-
	BoxTE	0.613	52.8	66.4	76.3	0.667	58.2	71.9	82.0	0.352	26.9	37.7	**51.1**
	TeMP-GRU	0.601	47.8	68.1	82.8	0.691	56.6	78.2	91.7	0.275	19.1	29.7	43.7
	TeMP-SA	0.607	48.4	68.4	84.0	0.680	55.3	76.9	91.3	0.232	15.2	24.5	37.7
	T-GAP	0.610	50.9	67.7	79.0	0.670	56.8	74.3	84.5	-	-	-	-
NAS	Random	0.642	52.8	72.2	84.3	0.701	58.0	78.8	91.7	0.353	27.1	37.9	**51.1**
	SPA	**0.658**	54.4	**73.7**	**85.7**	**0.713**	58.0	**82.0**	**93.3**	**0.360**	**28.2**	**38.4**	51.0

Table 2: Temporal KG completion evaluation results on ICEWS14, ICEWS05-15 and GDELT. The H@1, H@3, and H@10 metrics are multiplied by 100. Best results are 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.

demonstrates the effectiveness of the search space. On the other hand, compared with Random, which use the designed search space of SPA, the performance gains are from the single path one-shot search algorithm on obtaining better architectures.

Figure 4 shows the learning curves of GNN-based methods on ICEWS14 and ICEWS05-15, including TeMP, T-GAP and the proposed SPA. As can be seen, the searched architecture not only outperform baselines, but also have comparable time as the other GNN-based methods, which demonstrates the searched architecture can better capture diverse topological and temporal properties of different TKGs.

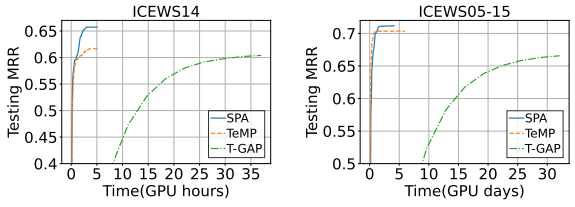


Figure 4: Comparison on convergence between the searched architectures (by SPA) and human-designed GNN-based methods.

Further, we visualize the searched architectures on three benchmark datasets in Figure 7, from which it is clear that different operation combinations of these four modules are obtained, i.e., data-specific architectures. We will discuss the details about the searched architectures in Section 3.5.

Therefore, these results demonstrate the need for data-specific methods for TKGC, and at the same time, the effectiveness of SPA on designing adaptive architectures.

3.3 Understanding the Search Algorithm

In this part, we evaluate the search algorithm from the perspectives of the efficiency of search algorithm, the effectiveness of weight sharing, and the choice of validation metric.

3.3.1 Efficiency of Search Algorithm

To show the efficiency of the search algorithm, we compare SPA with Random search baseline. Figure 5 shows the variation in the number of searched models during the search process.

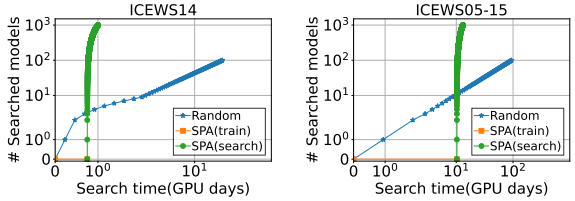


Figure 5: Comparison of SPA with Random search during the search process.

As can be seen, random search have to take a

long time to train each candidate architecture from scratch, while SPA spend most of the time on training supernet. In the stage of architecture search, SPA directly picks the corresponding weights from the trained supernet for the specific architecture evaluation, which significantly improves the efficiency compared to random search. This is mainly attributed to the weight-sharing strategy.

3.3.2 Effectiveness of Weight Sharing

To demonstrate the effectiveness of weight sharing, we empirically visualize the rank correlation of the validation performance between the weight sharing strategy and the stand-alone approach, as shown in Figure 6. For the stand-alone approach, we randomly sample 50 architectures $\mathcal{C}$, train and evaluate them from scratch. About weight sharing, we inherit the corresponding subweights of the trained supernet for each structure in $\mathcal{C}$ and evaluate it.

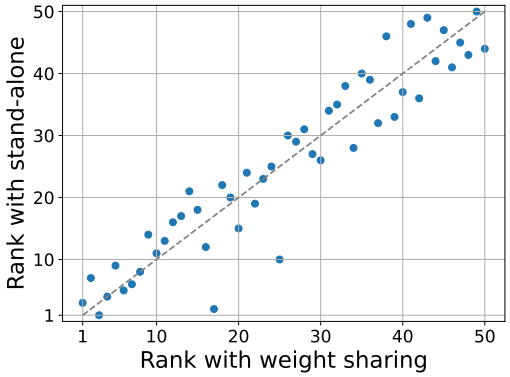


Figure 6: Rank correlation between stand-alone and weight sharing approach.

As can be seen, it is obvious that the rank of weight sharing validation MRR has near positive correlation with the rank of stand-alone validation MRR. And then, most structures that have high estimated ranks by weight sharing truly have high ranks using the setting of stand-alone. This demonstrates that the weight sharing strategy can search for good structures.

3.3.3 Choice of Validation Metric

In Section 2.4, we discuss the rationality of adopting the SPOS (Guo et al., 2020) method for search algorithm. Here, to show the impact of validation metric for SPOS, we compare the following SPA variants: (i) SPA(train loss), which uses training loss rather than valid MRR for evaluating candidate architecture in the stage of architecture search; (ii) SPA(valid loss), which uses validation loss for evaluating candidate architecture. Moreover, we adopt two variants of DARTS (Liu et al., 2019) as search algorithms, including SPA-D(train loss) and SPA-D(valid loss), which use gradient-based optimization to update architecture parameters by minimizing training loss and validation loss, respectively.

Table 3 shows the testing MRRs of different variants on ICEWS14 and GDELT. As can be seen, the use of validation MRR can help to select the better sub-network. The variants associated with DARTS run out of memory on GDELT with 3 million facts due to the demand for tremendous GPU memory. Besides, when using the same validation metric, the performances of architecture searched by SPOS consistently outperform that of DARTS, which may be due to the coupling of supernet weights and architecture parameters leading to the selection of inferior architectures.

Search algorithm	Variant	ICEWS14	GDELT
DARTS	SPA-D(train loss)	0.547	OOM
	SPA-D(valid loss)	0.615	OOM
SPOS	SPA(train loss)	0.587	0.324
	SPA(valid loss)	0.623	0.341
	SPA(valid MRR)	**0.658**	**0.360**

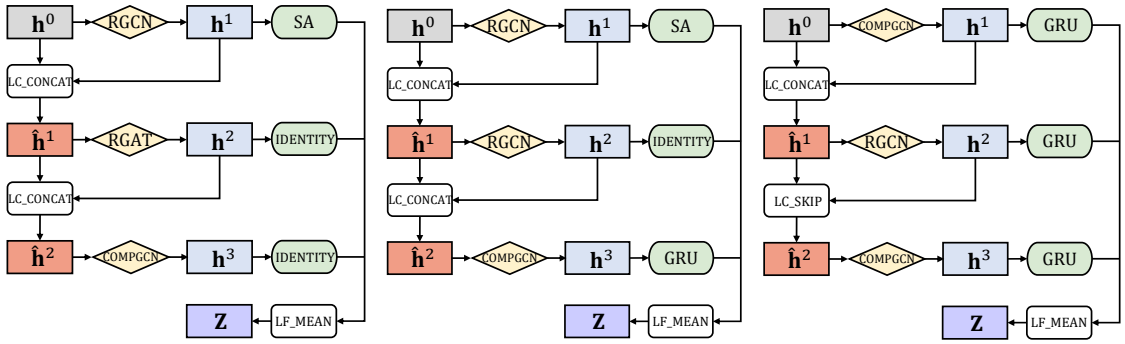
Table 3: Performance of SPA using different variants of search algorithm. "OOM" means out of memory.

3.4 Ablation Studies on the Search Space

We conduct ablation studies to show the influences of the four modules in the search space. For simplicity, we use two datasets: ICEWS14 and GDELT, and run SPA over different variants of search space, for which the results are shown in Table 4.

3.4.1 Spatial Aggregation Module

To evaluate how the spatial aggregation module affects the performance, we only search for the other three modules based on fixed aggregators RGCN and RGAT, which denoted as SPA-RGCN and SPA-RGAT, respectively. As shown in Table 4, with fixed aggregators, SPA-RGCN and SPA-RGAT have a performance drop compared with SPA. This indicatess that the diverse spatial aggregation modules can capture various topological information in different TKGs and significantly improve the model performance.



(a) ICEWS14 (b) ICEWS05-15 (c) GDELT

Figure 7: The searched architectures on three benchmark datasets.

Fixed	Variant	ICEWS14	GDELT
Spatial Aggregation	SPA-RGCN	0.648	0.347
	SPA-RGAT	0.653	0.357
Temporal Aggregation	SPA-IDENTITY	0.654	0.342
	SPA-GRU	0.585	0.358
Layer Connection	SPA-LC_SKIP	0.655	0.356
Layer Fusion	SPA-LF_SKIP	0.623	0.349
	SPA	**0.658**	**0.360**

Table 4: Performance of SPA using different search spaces. The first column represents the corresponding module we try to evaluate by fixing it with one OP in the reduced search space.

3.4.2 Temporal Aggregation Module

To evaluate the importance of searching for temporal aggregation module, we learn to design architectures with fixed temporal aggregation module instead. In Table 4, with the two predefined temporal aggregation operations, the degree of performance degradation is inconsistent across different datasets. To be specific, the performance drop is evident on SPA-IDENTITY for GDELT. But for ICEWS14, the performance of SPA-GRU drops significantly compared to SPA. This observation shows the importance of including temporal aggregation module in the search space. Meanwhile, it shows that the temporal aggregation operations should also be data-specific for TKGC.

3.4.3 Layer Connection and Layer Fusion Module

In this section, we evaluate the proposed Layer Connection and Layer Fusion Module, which are novel compared to existing GNN-based architectures for TKGC. By fixing the skip-connection function as `LC_SKIP`, we create the variant SPA-LC_SKIP, which means that we do not search for different skip-connection functions. By fixing the layer fusion function as `LF_SKIP`, we only preserve the output of last temporal aggregation module as entity representation. This variant is denoted by SPA-LA_SKIP, which means the outputs of intermediate layers are not used.

From Table 4, we can see that

- The performance drop of SPA-LC_SKIP means that the spatial aggregation module can benefit from skip-connection, which have been shown in previous works (Li et al., 2021; Li and King, 2020).
- The performance drop of SPA-LA_SKIP means that the outputs of intermediate layers are important for the final representation in temporal graph learning. Thus, it demonstrates the importance of the proposed Layer Fusion Module.

Taking all results in Table 4 into consideration, we can see that it is important for TKGC to search for combinations of operations from the four essential modules by SPA, which demonstrates the contribution of the proposed framework and the designed search space.

3.5 Case Study

We visualize the searched architectures on three benchmark datasets in Figure 7. Especially, the searched temporal aggregation modules contain more `IDENTITY` operations in ICEWS14, while in GDELT more `SA` operations are searched. This observation implies that for the ICEWS14 dataset, capturing complex temporal context may not be necessary in comparison to GDELT.

To verify above conjecture, we compare the differences in temporal properties between ICEWS14

and GDELT. From the perspective of temporal properties, as shown in Figure 8, the activity frequency of entities on GDELT is much higher than ICEWS14. This means that we do not need to design architectures with complicated sequential models for ICEWS14, but it is useful for GDELT. This result confirms our conjecture and the importance of designing data-specific architectures for TKGC.

Taking into consideration these experimental results from Figure 7 and 8, it indicates the effectiveness of our method in finding data-specific architectures for TKGC.

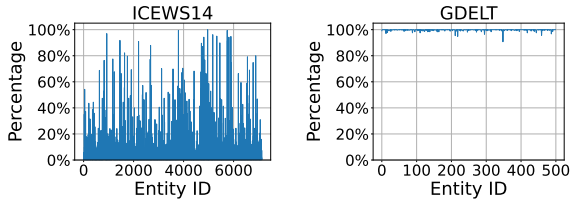


Figure 8: Difference in temporal property between two datasets. The figure represents the proportion of timestamps when the entity is active[1] to the total timestamps. As can be seen, entities in GDELT are much more active than those in ICEWS14.

4 Related Works

4.1 Temporal Knowledge Graph Completion (TKGC)

In the literature, existing methods for TKGC can be roughly divided into two categories: the embedding-based method and the GNN-based method. Embedding-based methods (Leblay and Chekol, 2018; Dasgupta et al., 2018; Goel et al., 2020; Lacroix et al., 2020; Messner et al., 2022) design time-aware score functions to measure the correctness of quadruples in TKGs. Although embedding-based methods well capture the semantic patterns in TKGs, they fail to capture the more complex topological patterns.

Recently, with the success of graph neural networks (GNNs), GNN has achieved significant progress in temporal knowledge graph completion. TeMP (Wu et al., 2020), uses structural encoder to obtain entity representations including multi-hop neighbor information and relies on temporal encoder to incorporate structural and temporal information into entity representation. T-GAP (Jung et al., 2021) designs one temporal GNN to learn structural and temporal information on TKG, and another GNN to dynamically expand and prune the inference subgraph from the query entity e_q by attention flow (Xu et al., 2018b). However, existing GNN-based methods use predefined structure and temporal encoder, which are difficult to adapt to various datasets.

4.2 Graph Neural Architecture Search

Neural architecture search (NAS) aims to automatically find suitable neural architecture for the given dataset, which has been demonstrated as a promising technique in many research fields such as computer vision and neural language processing.

More recently, some works focus on automatically designing GNNs by NAS. GraphNAS (Gao et al., 2021), AGNN (Zhou et al., 2019) learn to design aggregation operation. AutoGraph (Li and King, 2020) learns to select the connections in each intermediate layer. SNAG (Zhao et al., 2020) and SANE (Zhao et al., 2021) search to select and fuse the features of intermediate layers in the output node. AutoGEL (Wang et al., 2021b) focuses on designing intra-layer and inter-layer message passing GNN architectures automatically. However, no work applies NAS technique to design GNN for dynamic graphs or temporal knowledge graphs. To the best of our knowledge, SPA is the first method to learn data-specific GNN architectures for TKG completion.

5 Conclusion

In this paper, we propose a novel method SPA to automatically design data-specific architectures for TKGC task. We define a novel and expressive search space, in which different combinations of operations can capture various patterns of different TKGs. To enable efficient search on top of the search space, we adopt a flexible and effective search algorithm, which trains a simplified supernet in that each architecture is a single path, thus greatly reducing the GPU memory cost. To demonstrate the effectiveness of SPA for TKGC, we conduct extensive experiments on three datasets. The experimental results show that SPA can search SOTA data-specific architectures for TKGC.

For future work, we will explore more advanced NAS approaches to further improve the search efficiency of SPA. Besides, a promising direction is to explore how to efficiently search network architectures and hyper-parameters simultaneously.

[1] An entity is active at a timestep if it has at least one neighboring entity in the same KG snapshot (Wu et al., 2020).

Limitations

There are two limitations for SPA. (1) SPA is focused on method design rather than system design. In the future, we will co-design the algorithm and the system to further improve the efficiency. (2) At present, SPA only search for data-specific architectures, while hyper-parameters are also important for TKGC. A promising direction is to explore how to efficiently search network architectures and hyper-parameters simultaneously.

Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work was supported in part by the National Key Research and Development Project of China (No. 2020AAA0107704), the National Natural Science Foundation of China (Nos. U1803263, U22B2036), the National Science Fund for Distinguished Young Scholarship of China (No. 62025602), Fok Ying-Tong Education Foundationm China (No. 171105), Key Technology Research and Development Program of Science and Technology-Scientific and Technological Innovation Team of Shaanxi Province (No. 2020TD-013), and the XPLORER PRIZE.

Q. Yao is sponsored by CCF-Baidu Open Fund and Tsinghua University-Foshan Institute of Advanced Manufacturing.

References

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 2787–2795.

Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y Hammerla. 2019. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*.

Borui Cai, Yong Xiang, Longxiang Gao, He Zhang, Yunfeng Li, and Jianxin Li. 2022. Temporal knowledge graph completion: A survey. *arXiv preprint arXiv:2201.08236*.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. 2018. HyTE: Hyperplane-based temporally aware knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2001–2011, Brussels, Belgium. Association for Computational Linguistics.

Chao Gao, Junyou Zhu, Fan Zhang, Zhen Wang, and Xuelong Li. 2022. A novel representation learning for dynamic graphs based on graph convolutional networks. *IEEE Transactions on Cybernetics*, pages 1–14.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2021. Graph neural architecture search. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1403–1409.

Alberto García-Durán, Sebastijan Dumančić, and Mathias Niepert. 2018. Learning sequence encoders for temporal knowledge graph completion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4816–4821, Brussels, Belgium. Association for Computational Linguistics.

Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic embedding for temporal knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3988–3995.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

Jaehun Jung, Jinhong Jung, and U Kang. 2021. Learning to walk across time for interpretable temporal knowledge graph completion. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 786–795.

Seyed Mehran Kazemi and David Poole. 2018. Simple embedding for link prediction in knowledge graphs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4289–4300.

Timothée Lacroix, Guillaume Obozinski, and Nicolas Usunier. 2020. Tensor decompositions for temporal knowledge base completion. In *International Conference on Learning Representations*.

Julien Leblay and Melisachew Wudage Chekol. 2018. Deriving validity time in knowledge graph. In *Companion Proceedings of the The Web Conference 2018*, pages 1771–1776.

Kalev Leetaru and Philip A Schrodt. 2013. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*, volume 2, pages 1–49. Citeseer.

Guohao Li, Matthias Mueller, Guocheng Qian, Itzel Carolina Delgadillo Perez, Abdulellah Abualshour, Ali Kassem Thabet, and Bernard Ghanem. 2021. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Yaoman Li and Irwin King. 2020. Autograph: Automated graph neural network. In *International Conference on Neural Information Processing*, pages 189–201. Springer.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. Darts: Differentiable architecture search. In *International Conference on Learning Representations*.

Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000.

Johannes Messner, Ralph Abboud, and Ismail Ilkan Ceylan. 2022. Temporal knowledge graph completion using box embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7779–7787.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8026–8037.

Ali Sadeghian, Mohammadreza Armandpour, Anthony Colas, and Daisy Zhe Wang. 2021. Chronor: Rotation based temporal knowledge graph embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6471–6479.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 519–527.

Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. 2021. Question answering over temporal knowledge graphs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6663–6676, Online. Association for Computational Linguistics.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer.

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR.

Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2019. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion Proceedings of The 2019 World Wide Web Conference*, page 301–307.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2020. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. volume 30.

Huandong Wang, Qiaohong Yu, Yu Liu, Depeng Jin, and Yong Li. 2021a. Spatio-temporal urban knowledge graph enabled mobility prediction. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(4):1–24.

Zhen Wang, Chunyu Wang, Xianghua Li, Chao Gao, Xuelong Li, and Junyou Zhu. 2022. Evolutionary markov dynamics for network community detection. *IEEE Transactions on Knowledge and Data Engineering*, 34(3):1206–1220.

Zhili Wang, Shimin Di, and Lei Chen. 2021b. Autogel: An automated graph neural network with explicit link information. In *Advances in Neural Information Processing Systems*, volume 34, pages 24509–24522. Curran Associates, Inc.

Jiapeng Wu, Meng Cao, Jackie Chi Kit Cheung, and William L. Hamilton. 2020. TeMP: Temporal message passing for temporal knowledge graph completion. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5730–5746, Online. Association for Computational Linguistics.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. Snas: stochastic neural architecture search. In *International Conference on Learning Representations*.

Chengjin Xu, Yung-Yu Chen, Mojtaba Nayyeri, and Jens Lehmann. 2021. Temporal knowledge graph completion using a linear temporal regularizer and multivector embeddings. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2569–2578.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018a. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR.

Xiaoran Xu, Songpeng Zu, Chengliang Gao, Yuan Zhang, and Wei Feng. 2018b. Modeling attention flow on graphs. *arXiv preprint arXiv:1811.00497*.

Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*.

Hui Zhang, Quanming Yao, James T. Kwok, and Xiang Bai. 2022a. Searching a high performance feature extractor for text recognition network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–15.

Yongqi Zhang, Quanming Yao, and James. T Kwok. 2022b. Bilinear scoring function search for knowledge graph learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2021. Automated machine learning on graphs: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4704–4712. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Huan Zhao, Lanning Wei, and Quanming Yao. 2020. Simplifying architecture search for graph neural network. *arXiv preprint arXiv:2008.11652*.

Huan Zhao, Quanming Yao, and Weiwei Tu. 2021. Search to aggregate neighborhood for graph neural network. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 552–563. IEEE.

Yuyue Zhao, Xiang Wang, Jiawei Chen, Yashen Wang, Wei Tang, Xiangnan He, and Haiyong Xie. 2022. Time-aware path reasoning on knowledge graph for recommendation. *ACM Transactions on Information Systems (TOIS)*.

Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. 2019. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*.

A Appendix

A.1 Loss Function

To train our TKGC model using score function, the model parameters are learned using gradient-based optimization in mini-batches. Specifically. for each quadruple $\eta = (s, r, o, t) \in \mathcal{D}^+$, we sample a negative set of entities $\mathcal{D}^-_{\eta,o} = \{o' | (s, r, o', t) \notin \mathcal{D}^+\}$. Then, we apply the cross-entropy loss function for object queries to train the model:

$$\mathcal{L}_{\text{obj}} = -\sum_{(s,r,o,t)\in\mathcal{D}^+} \frac{\exp(\phi(s,r,o,t))}{\sum_{o'\in\mathcal{D}^-_{\eta,o}} \exp(\phi(s,r,o',t))}. \tag{8}$$

Similarly, we can also obtain the loss for subject queries $\mathcal{L}_{\text{sub}}$. The final training loss is the sum of losses for two types of queries: $\mathcal{L} = \mathcal{L}_{\text{sub}} + \mathcal{L}_{\text{obj}}$.

A.2 Dataset Statistics and Characteristics

The dataset statistics are summarized in Table 5.

A.3 Implementation Details

All the experiments are implemented in Python with the PyTorch framework (Paszke et al., 2019) and run on a single NVIDIA RTX 3090 GPU with 24GB memory.

For Random, we use the Adam optimizer, set learning rate is 0.001, dropout rate = 0.1, and L2 norm to 0.0005. We randomly sample 100 architectures from the designed search space and train them from scratch. After training finished, we select one candidate with the highest validation performance.

For SPA, we set the epoch T_1 for supernet training is 800 and the epoch T_2 for architecture searching is 1000. in each minibatch sample single path to train supernet. After training process is finished, we derive the candidate architecture with the highest validation performance from the supernet by random search. Repeat 5 times with different seeds, we can get 5 candidates.

Other hyperparameters settings for NAS methods during the search process are shown in Table 6.

The searched candidates are finetuned individually with the hyper-parameters shown in Table 7. In the stage of fine-tuning, we use the ReduceLROnPlateau scheduler. Each method candidates 30 hyper steps. In each hyper step, a set of hyperparameters will be sampled from Table 7 based on Hyperopt, and then generate final performance on the testing data.

Dataset	# entities	# relations	# time steps	N_{train}	N_{valid}	N_{test}	N_{total}
ICEWS14	7,128	230	365	72,826	8,941	8,963	90,730
ICEWS05-15	10,488	251	4017	386,962	46,275	46,092	479,329
GDELT	500	20	366	2,735,685	341,961	341,961	3,419,607

Table 5: Statistics of ICEWS14, ICEWS05-15 and GDELT datasets.

Dataset	Batch size	τ	Head number of spatial module	Head number of temporal module	Embedding size	Hidden size	Gradient clipping
ICEWS14	8	8	4	4	100	100	1
ICEWS05-15	8	8	4	4	100	100	1
GDELT	2	4	4	4	100	100	1

Table 6: Other hyperparameters setting for SPA during the search process.

Hyperparameter	Value range
Head number of spatial module	$\{2, 4, 8\}$
Head number of temporal module	$\{2, 4, 8\}$
Weight decay	$[10^{-5}, 10^{-3}]$

Table 7: Hyperparameters we used during the fine-tuning stage.