

Research Article

Research on Knowledge Graph Completion Model Combining Temporal Convolutional Network and Monte Carlo Tree Search

Ying Wang,1,2 Mingchen Sun,1,2 Hongji Wang,2,3 and Yudong Sun1,2

1College of Computer Science and Technology, Jilin University, Changchun 130012, China
2Key Laboratory of Symbol Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China
3College of Artificial Intelligence, Jilin University, Changchun 130012, China

Correspondence should be addressed to Hongji Wang; wanghj21877@163.com

Received 16 June 2021; Revised 28 December 2021; Accepted 2 February 2022; Published 16 March 2022

Academic Editor: Giuseppe D’Aniello

Copyright © 2022 Ying Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In knowledge graph completion (KGC) and other applications, learning how to move from a source node to a target node with a given query is an important problem. It can be formulated as a reinforcement learning (RL) problem transition model under a given state. In order to overcome the challenges of sparse rewards and historical state encoding, we develop a deep agent network (graph-agent, GA), which combines temporal convolutional network (TCN) and Monte Carlo Tree Search (MCTS). Firstly, we combine MCTS with neural network to generate more positive reward trajectories, which can effectively solve the problem of sparse rewards. TCN is used to encode the history state, which is used for policy and Q-value respectively. Secondly, according to these trajectories, we use Q-Learning to improve the network and parameter sharing to enhance TCN strategy. We apply these steps repeatedly to learn the model. Thirdly, in the prediction stage of the model, Monte Carlo Tree Search combined with Q-value method is used to predict the target nodes. The experimental results on several graph-walking benchmarks show that GA is better than other RL methods based on-policy gradient. The performance of GA is also better than the traditional KGC baselines.

1. Introduction

The original intention of knowledge graph is to describe and store various entities and their relations in the real world. Although a typical knowledge graph may contain millions of entities and billions of relations, it is usually far from completeness. The purpose of knowledge graph completion (KGC) is to use the association information in the existing knowledge graph to predict the missing relation between entities in order to complete the knowledge graph. Embedding-based ranking methods, firstly learn embedding vectors based on existing triples. By replacing the tail entity or head entity with each entity, those methods calculate scores of all the candidate entities and rank the top k entities. Embedded learning of entities and relationships has achieved significant performance improvement in some benchmarks, but it can not model complex relationship paths. Relational path reasoning turns to using path information in graph structure. Random walk reasoning has been widely studied. Furthermore, deep reinforcement learning (RL) is introduced for multi-hop reasoning by formulating path-finding between entity pairs as sequential decision making, specifically a Markov decision process (MDP). The policy based RL agent learns to find a step of relation to extending the reasoning paths via the interaction between the knowledge graph environment, where the policy gradient is utilized for training RL agents. Compared with random walk reasoning, deep reinforcement learning can get a better path.

RL-based KGC can also be understood as constructing a function \( f(G, n_S, q) \) to predict the target node \( n_T \), where \( f(\cdot) \) can be learned from a training dataset composed of samples such as \((n_S, q, n_T)\). In this work, we use a graph walk agent to construct the \( f(G, n_S, q) \) model, which can swim from \( n_S \) to \( n_T \) through intelligent decision. Because \( n_T \) is unknown, the problem cannot be solved by conventional search algorithms.
such as $A^\ast$-search [1], which seeks to find paths between the given source and target nodes. On the contrary, the agent needs to learn its search police from a training dataset, so that after the training, the agent knows how to walk over the graph to reach the correct target node $n_t$ for a pair of given $(n_s, q)$. Moreover, each training sample exists in the form of “(source node, query, target node).” However, the agent only receives the delayed reward: when the agent correctly (or wrongly) predicts the target node in the training set, the agent will get positive (or zero) reward. Therefore, we describe the problem as a Markov decision process (MDP) and train the agent through reinforcement learning (RL).

RL-based KGC presents two major challenges. First of all, because the state of MDP is the whole walking path, in order to get the correct decision, it usually needs not only query, but also the information of all nodes on the whole walking path. The agent must utilize complete historical information with the input query $Q$, to make this decision. Secondly, the reward is sparse, and only at the end of the path can we get the reward of the path.

In this paper, we construct a graph agent network (GA) combining MCTS and reinforcement learning, which can effectively solve these two challenges. Firstly, we combine Monte Carlo Tree Search (MCTS) with MDP transition model to obtain more trajectories with positive reward. Secondly, GA introduces a new temporal convolutional network (TCN) structure, which encodes the whole history of trajectories into vector representation and is further used to model the police and the Q-function. However, their off-police nature prevents them from being used by the RL method. In order to solve this problem, we design a parameter sharing structure between Q-value network and decision-making network, which makes the police network can be improved indirectly by Q-Learning on the off-police trajectories. Thirdly, we utilize the obtained trajectories to train the model and update the parameters. Our method is in sharp contrast to the existing KGC methods based on RL. The latter uses the police gradient (reinforcement) method and usually needs a lot of promotion to obtain a positive reward trajectory, especially in the early stage of learning. The experimental results on several benchmark tests (several actual KBC tasks) show that our method is better than the previous RL-based method and the traditional KGC method. The evaluation results of the model are close to those of other reinforcement learning methods on some tasks, and the best results are obtained on multiple completion tasks. The major contributions of this paper are summarized as follows:

(1) We introduce temporal convolutional network structure to encode the whole history of trajectories into vector representation, which has the characteristics of time sequence and can better represent the historical information.

(2) In order to solve the challenge of sparse reward, GA takes advantage of the fact that MDP transition model is known and deterministic. Because whenever the agent takes an action, by selecting an edge connected to a next node, the identity of the next node (which the environment will transition to) is already known.

(3) We introduce Monte Carlo Tree Search to obtain more trajectories with positive reward.

(4) We design a parameter sharing structure between Q-value network and decision-making network, which makes the police network can be improved indirectly by Q-Learning on the off-police trajectories.

Section 2 discusses the related work, and Section 3 is about problem statement. Section 4 develops the deep agent network GA, including model structure, training algorithm and test algorithm. Section 5 gives the experimental results. Finally, we summarize the paper in Section 6.

2. Related Work

In the early stage, the classical method of relation extraction, such as path ranking algorithm [2], is realized by using reasoning rules and statistical methods. This method takes each different relation path as a one-dimensional feature, and makes a large number of statistics of different relation paths in the knowledge graph, and constructs the feature vector for classification for the relationship. Finally, the feature analysis model is used to classify the relation features, which achieves good relation extraction effect and becomes one of the representation methods of relation completion. However, this co-occurrence statistics method based on relation faces serious data sparsity problem.

Then, the reasoning and completion model of knowledge graph represented by trans series methods has achieved good results, and a variety of variant algorithms have been produced. The main idea of this model is to learn the representation of entities and relationships in low dimensional dense space by using the semantic and structural relations of knowledge graph, and complete the subsequent reasoning and completion tasks by using the correlation contained in the embedding. However, the Trans models are sensitive to hyper parameters and lacks of scalability. At the same time, the processing ability of dynamic knowledge graph and cold start entity is insufficient. In recent years, with the development of deep learning, some deep neural networks have been applied to the feature mining of knowledge graph, such as graph-NNs [3], GCN [4], GAT [5], and so on. With the popularity of reinforcement learning in recent years, reinforcement learning algorithm is widely used in various fields, such as DeepPath [6], MINERVA [7], and M-walk [8]. DeepPath requires target entity information to be in the state of the RL agent, and cannot be applied to tasks where the target entity is unknown. MINERVA uses a policy gradient method to explore paths during training and test. M-walk further exploits state transition information by integrating the MCTS algorithm. In 2021, Huang [9] proposed a novel knowledge graph completion model named directional multi-dimensional attention convolution model that explores directional information and an inherent deep expressive characteristic of the triple. Jagvaral [10] proposed a new approach for knowledge graph completion that combines bidirectional long short-term memory (BiLSTM) and convolutional neural network modules with an attention
mechanism. Our proposed method uses TCN in encoding history information. The evaluation results of our model are close to those of other reinforcement learning methods on some tasks, and the best results are obtained on multiple completion tasks.

3. Problem Statement

In this section, we introduce and explain some related concepts and formalize the problem of knowledge graph completion.

3.1. Reinforcement Learning. Reinforcement learning (RL) refers to a method, in which an agent tries to act and learn the optimal decision-making strategy from wrong experience in the process of interacting with the environment to solve the sequential decision-making problems in natural science, social science, engineering and other fields [11–14].

There are two key problems in reinforcement learning: environmental design and agent design. Early tasks rely on simple environment, in which the number of state is limited and the action is basically fixed, so the mapping space from state to action can be determined. Therefore, the agent can save the value estimation of each action of each state, and record it in the form of table, that is, the Q value table, which guides the agent to choose. In the view of the complex environment, reinforcement learning technology has made great progress in the past few years, among which the most important is the integration of reinforcement learning and deep learning, which is called deep reinforcement learning [15]. It no longer depends on obtaining all states in the state space, but uses the features extracted from the states to learn the decision-making strategy, and uses the deep neural network to fit more complex function mapping, which solves the problems of complex environment design and complex agent design to a certain extent.

Deep reinforcement learning is widely used in natural language processing, game confrontation and robot control, such as DeepStack [16], AlphaGo [17] and Deep Q-network (DQN) [18]. DQN algorithm is one of the representative algorithms in Q-network type algorithm. Its excellent performance in various research fields has attracted the attention of researchers. Then, various types of improved algorithms are derived, such as Double-DQN [19] and Duel DQN [20], which have made effective improvements in the over estimation of DQN, and have been optimized and improved in performance. Reinforcement learning can be divided into two categories: value method and policy gradient method. Q-Learning is to learn the estimated value of various state actions. Besides, there are other types of methods dominated by policy gradient method, such as trust region method [21, 22], deterministic policy gradient method [23, 24], combination of departure policy and policy gradient method [25, 26], unsupervised reinforcement and assisted learning [27, 28], etc. Nowadays, deep reinforcement learning has explored and practiced for many problems in different fields or combined with different models [29–33].

3.2. Monte Carlo Tree Search. Monte Carlo Tree Search (MCTS) [34] is a general term for a class of tree search algorithms, which is used as the estimation of frequency probability and can effectively solve some problems with large exploration space. In the research of reinforcement learning, Deepmind company has given full play to the role of Monte Carlo Tree Search in the field of Go [15, 35]. In the processing of using state space of Go to construct a search tree, the number of leaf nodes can not be exhaustive, even the previous pruning technology is difficult to play a role in large-scale state space. MCTS explores in the huge search space with the method of simulating tree nodes and expanding, in which the typical algorithm UCB (Upper Confidence Bounds) [36] is to give priority to the child nodes that have not been explored when selecting the child nodes. If all the child nodes have been explored, the selection is based on the score of nodes. The score is not only positively correlated with the probability that the child node can eventually get positive reward, but also negatively correlated with the number of times that the child node has been explored. Therefore, MCTS can explore according to the configuration and utilize different weights, which can achieve more heuristic methods than random or other strategies.

3.3. Sequence Modeling. Sequence modeling is a very common problem, which involves applications such as speech processing, language modeling and time series prediction. In recent years, due to the huge amount of data and large computing power, RNN model [37, 38] has sprung up, and the task of sequence modeling turns to use RNN model. The RNN model can memorize the historical information in the sequence through the hidden state. However, the gradient accumulation of the cycle is easy to cause the problem of gradient disappearance. The Long Short Term Memory (LSTM) [39] model solves the problem of gradient disappearance to a certain extent, and can better realize long-term memory. GRU [40] proposes to simplify the LSTM model and improve the training speed, making the recurrent model absolutely dominant in the processing of sequence problems. Moreover, the use of convolution networks for sequence related work also emerged. For example, Wavenet [41] model uses hole causal convolution to process serialized audio data, Gated CNN [42] model proposes a new gated mechanism, and combines CNN with natural language processing.

In 2017, Bai et al. [43] published a paper on the application of temporal convolutional network (TCN) in sequence modeling, which made the traditional sequence modeling tasks no longer only rely on the recurrent network structure such as LSTM and GRU, and the results in multiple tasks exceeded those of the recurrent neural network. In TCN [44], dilated convolution, causal convolution [45, 46] and residual structure [47] are used to deal with sequence modeling. TCN model successfully surpasses the traditional recurrent neural network in multiple sequence modeling tasks.
Problem. For inputs of our framework, we have node embeddings $N \in \mathbb{R}^{n \times d}$ and edge embeddings $E \in \mathbb{R}^{e \times d}$ of given graph, and we have source node $n_s$ and query vector $q$. We use Monte Carlo Tree Search and Markov decision process to obtain more positive paths $P = \{n_1, n_2, \ldots, n_l\}$. We utilize these trajectories to train our agent network.

For output of our framework, we can get the target node through a path from source node to target node obtained by trained agent network.

The task of our framework is knowledge graph completion. We use TCN to fuse historical status information and neural network to encode action and state. We can get the target node through the agent trained by these trajectories.

4. The Graph-Agent, GA

4.1. Markov of Knowledge Graph Completion. The task of knowledge graph completion is to find the corresponding target entity in the environment accurately according to the given entity $n_s$ and relation $q$ in the environment, and form a complete triple. The process is represented as $(n_s, q, ?) \rightarrow (n_s, q, n_T)$. Reinforcement learning task is usually described by Markov Decision Process (MDP). Markov Decision Process describes the process of spatial transition, and the number of candidate actions depend on its adjacent entities and relations. In addition, the adjacency information constitutes the action space.

State space: the state at any time in MDP needs to contain all historical information. In knowledge graph, if the information of an entity is regarded as the current state, it cannot satisfy the Markov property. We can keep the transition history of the nodes to ensure that the construction of the state satisfies the Markov property, which leads to the new state after each step of transition contains more information of nodes and edges than before, while the encoding of state becomes a difficult problem to solve. The states on the knowledge graph are expressed as follows:

$$s_t = s_{t-1} \cup \{a_{t-1}, n_t, E_t\}, \quad (1)$$

where $s_t$ and $n_t$ are the state and node at time $t$, $a_{t-1}$ is the selected action at time $t - 1$, and $N_t$ and $E_t$ are the sets of neighbours and relations of $n_t$.

Action space: the walking on graph leads to state transition, and the number of candidate actions depends on the number of adjacent entities of current entity. In this paper, we consider the relation between each pair of entities, and encode an entity and its relation together to form a candidate action. We also exclude the entities and relations of the previous state from the action space of each state, because it is meaningless to swim back and forth between two nodes in a path. Therefore, the adjacent entities and relations constitute the action space $A_n = \{a_1, a_2, \ldots, a_{k-1}, a_{stop}\}$, where $a_{stop}$ represents an action of termination in the current state and means that the process of MDP is over and $k$ represents the number of candidate actions. In order to ensure that the trained agent can have the same action space, we use a shared parameter encoding network, which can unify the action space of each entity into a complete action space $A = \{A_1, A_2, \ldots, A_n\}$.

Transition function: the advantage of network structure data is that its environment is known. Therefore, the description and definition of state transition function $P$ are as follows: (1) When an action $a$ occurs in a state $s_t$, the state will transfer to an entity corresponding to the action, and our environmental state changes to the new state $s_{t+1}$ dominated by the next entity. (2) The agent chooses the terminal action $a_{stop}$, which means that it will not select any new entity and its relation.

Reward function: as the only feedback signal to the agent, reward plays a leading role in the process of agent training. In this paper, we utilize trajectory memory to train agent in the way of off-policy. For any trajectory, the reward is determined by the terminal node $T$ as follows:

$$R_{tra} = \begin{cases} 1, & f \text{ node } T \text{ is target node}, \\ 0, & \text{ else}. \end{cases} \quad (2)$$

The reward of $(s, a, s_{t+1})$ which is a nontermination state transition process is 0. The Q value of the model is updated by using the reward of the termination state and Temporal-Difference (TD) method. In addition, the basic assumption of this reward function is that the two entities connected by a relation can also be connected by other relations through more hops, so that the relation can be represented or inferred by other relations to a certain extent. However, they are not completely equivalent. It is a reasonable assumption that the relation to query is positively correlated with other relations in the path.

4.2. Graph-Agent. There are two main problems in graph such as knowledge graph. One is that the state is successive entities and the relations sequence, and the lengths of these sequences are different, which is difficult to be used as input. The other is that the optional actions of each state depend on the adjacent relations and entities of the entity in the current state, and the number of them is also uncertain. In order to solve the first problem, the temporal convolutional network is used to encode the state, and the dimension of the sequence state is transformed into the same dimension. Meanwhile, the causal convolution preserves the Markov property of the state. For the second problem, all actions are encoded by the same network and converted to the same implicit full action space. As selecting output action, we only need to think about the candidate actions.

This section will introduce the network structure of deep agent (graph-agent) in detail, including action encoding layer $A = ENC(\theta)$, state encoding layer $S = ENC(\theta)$ and decision-making layer $\pi_0$.

4.2.1. Neighbor Information Encoding of Entity and Full Action Space. A state in the knowledge graph is dominated by an entity $n_s$, and the action space of the state changes with its adjacent entities. The size of the space is uncertain. Its action space contains both entities and the relations between entities. In different action spaces under different states, the same action should have the same representation. In addition, the adjacent information $N_t$ and $E_t$ in state
\[ s_t = s_{t-1} \cup \{ a_{t-1}, n_t, N_t, E_t \} \]

4.2.2. State Encoding Layer with Temporal Convolutional Network. In deep agent networks, the input of state is very important. Temporal convolutional network is better at capturing temporal dependence. Convolution is easier to capture local information, and hole convolution can also expand the sensing range, so that the output at any time can perceive all the input of the previous time. In addition, the state \( s_t \) can be expressed as \( s_t = s_{t-1} \cup \{ a_{t-1}, n_t, h_{at} \} \) after encoding the full action space and adjacent information. The temporal convolutional network is used to obtain the historical state information, of which the structure is shown in Figure 2.

In Figure 2, \( n_0 = n_t \) is the initial node of the state, \( n_t \) is the current node, \( h_{at-1} \) is the adjacency information code at \( t-1 \) time, \( h_{at-1} \) is the action code selected at time \( t-1 \). The serialization state is encoded into sequence information, and causality convolution is used to limit the temporality, so that the result of any position of the latter layer is only related to the data before that position of the former layer. At the same time, the receptive field is expanded by using dilation convolution. The recursive structure of state representation makes the terminal state contain all the previous states of the transition. Finally, the complete historical information \( h_{at0}, h_{at1}, h_{at} \) is output. In order to enhance the recognition of the current state of the agent policy network, the encoding of the final state combines the adjacent node encoding \( h_{at} \) and the historical information \( h_{at} \) at time \( t \) to obtain the state at time \( t \):

\[ H_{st} = \text{cat}(h_{at}, h_{at}). \]  

4.2.3. Policy Decision-Making Layer Network. Deep agent network mainly focuses on the decision-making layer, which receives the data extracted from the environment features and judges the probability of each action in the current action space. In this paper, the full connection network is used to map the environmental state features to the output action. Its structure is shown in Figure 3.

In Figure 3, the decision-making layer network of the agent receives the state feature vector \( H_{st} \) given by the \( S = \text{ENC}(\theta) \) layer and the query relation vector query as the input of the decision-making layer \( \pi_h \). The compressed feature data are applied to two parts, one is to generate the selection probability of the terminal action \( a_{stop} \), the other part is to generate the selection probability of other actions \( a_t \) by combining the action encodings generated by the action encoding layer. Its structure is as follows:
Figure 3: The structure of decision-making layer $\pi_\theta$.

\[
\begin{align*}
    u_{\text{stop}} &= f_{\theta_2}(f_{\theta_1}(\text{cat}(\text{query}, H_{st}))), \\
    u_i &= f_{\theta_3}(\text{cat}(f_{\theta_1}(\text{cat}(\text{query}, H_{st})), h_{ai})), \\
    \pi_\theta(\cdot|S) &= \varphi(u_{\text{stop}}, u_1, \ldots, u_n), \\
    a_t &= \max(\pi_\theta(\cdot|S_t)), \\
    Q_\theta(S_t) &= \delta(u_{\text{stop}}, u_1, \ldots, u_n),
\end{align*}
\]

where $f_{\theta_1}$, $f_{\theta_2}$, and $f_{\theta_3}$ are fully connected networks, cat is tensor splicing operation and query is a splicing vector of initial entity representation vector and query relation vector. Feature vector $(\text{query}, H_{st})$ is compressed by network $f_{\theta_1}$. And then as environment feature, the feature vector is directly used to generate $u_{\text{stop}}$ on the one hand. On the other hand, combined with the candidate action vectors generated by the action encoding layer, the value of each action is fitted through the network $f_{\theta_3}$. $\delta$ represents the sigmoid function, which is used to convert the values of $(u_{\text{stop}}, u_1, \ldots, u_n)$ to the Q values in the range of 0 to 1. $\varphi$ represents softmax function and converts $(u_{\text{stop}}, u_1, \ldots, u_n)$ into candidate action probability. The decision-making network selects the action with the maximum probability as the final output action of the current agent network.

Figure 4 shows the complete deep agent network structure. The data represented by the original knowledge graph is input into the action encoding layer at the bottom, $h_{a0}$, $h_{a1}$, ..., $h_{an}$ and each candidate action $h_{ai}$ are output at the same time. The generated vectors are used to construct the temporal historical information and the policy encoding layer network to select from the candidate actions.

4.3. Environment Exploration and Agent Training Prediction

4.3.1. The Algorithm of Upper Confidence Bound Applied to Trees Combined with Policy Exploration. In the large-scale network structure, the nodes relations are complex, and the number of paths from the source node to the target node is huge, which also leads to an important problem: the sampling efficiency in the process of policy network training is low. If the on-policy approach is adopted, the training of the model will become very slow, because the model can not find a positive reward in a long period of time. Therefore, we use Monte Carlo Tree Search method to build a state search tree to save the value of each state and the number of visits, so as to solve the sparse reward problem. Its core algorithm UCT (Upper Confidence Bound Applied to Trees) is represented in the knowledge graph as follows:

\[
a_t = \arg \max_a \left\{ c \times \ln \left( \frac{\sum_a N(s_t, a)}{N(s_t, a)} \right) + W(s_t, a) \right\},
\]

where $W(s_t, a)$, $N(s_t, a)$, and $W(s_t, a)/N(s_t, a)$ are the value, frequency, and estimation value of the selected action $a$ under the state $s_t$. $c \times \sqrt{\ln(\sum_a N(s_t, a))}/N(s_t, a)$ is called exploration component, which can avoid the insufficient exploration caused by only applying value estimation, and can fully mine the less explored actions. When UCT selects the terminal action $a_{\text{stop}}$ or reaches the maximum search depth, MCTS completes a simulation and uses $V_\theta(s_T) = 1$ if node T is target node to update $W(s_t, a)$ and $N(s_t, a)$. The sampling policy in this paper is to improve the UCT algorithm combined with the policy decision-making network and enhance the exploration performance of the sampling, so as to reduce the possibility of falling into the local optima. The algorithm of upper confidence bound applied to trees combined with policy exploration (Policy-UCT) is as follows:

\[
a_t = \arg \max_a \left\{ c \times Q_\theta(s_t) \ln \left( \frac{\sum_a N(s_t, a)}{N(s_t, a)} \right) + W(s_t, a) \right\},
\]

The core idea is that the Q value given by the policy network is used to dynamically adjust the exploration rate of each action in each state during the exploration process of MCTS. Among the candidate actions with the same exploration item value, the more valuable actions given by the policy network are easier to be explored by the algorithm. In the case of the same source node, there may be multiple paths to the target node. The Q value changes with the training process, and the agent thinks that the action with high value is more likely to lead to target node. Therefore, policy-UCT tends to select actions with high value $W(s_t, a)/N(s_t, a)$ for node exploration, while node exploration tends to select actions with higher $Q_\theta$ value and lower access times $N(s_t, a)$.

4.3.2. Model Training. The key idea of this paper is to use policy enhanced Monte Carlo Tree Search to simulate the production of a series of trajectories with positive rewards. Learning from these trajectories can significantly improve $\pi_\theta$. In this paper, we strengthen the deep agent network and repeatedly apply these steps to improve the exploration policy to obtain more sampling trajectories. However, the data is generated by different depth agent
network parameters, namely off-policy data, which breaks the inherent assumption of policy gradient method. So we use this data to update with Q-Learning method, as shown in Figure 5.

**Step 1.** Initialization of knowledge graph datasets, action decision-making network and state encoding network. The action network and evaluation network of deep agent with the same parameters are initialized.
Step 2. The Monte Carlo Tree Search of Policy-UCT algorithm is used to collect samples from the knowledge graph, and the samples are stored in the trajectory storage pool in the form of trajectory until the storage pool reaches the maximum capacity. If the storage pool reaches the maximum capacity before the start of sampling, the newly added trajectory will replace the earliest one. After the replacement amount reaches a certain number, the next step will be started.

Step 3. The trajectories are extracted from the storage pool for network training to update the deep agent action network parameters in small batch.

Step 4. The new policy network $\pi_\theta$ will continue to sample from the knowledge graph by the way of Policy-UCT, combining with Monte Carlo Tree Search. Repeat Step 2 and Step 3 to update the deep agent network parameters until the condition of updating Q-net evaluation network. The current network parameters of deep agent are used to cover the parameters of evaluation network.

Step 5. Repeat Steps 1–4 until the loss error reaches the requirement or the preset training times. Output the final deep agent network.

The updating formula of simple DQN is as follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta V_\theta(s_t, a_t) \times \left( y(s_t, a_t) + y \max_{a'\neq a} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t) \right).$$

In order to avoid overestimating the state value, Double-DQN is used to update the parameters of deep agent network. The value of $Q_\theta(s_{t+1}, a')$ is changed to the value of the action $a'$ selected by the action network in the evaluation network:

$$\theta \leftarrow \theta + \alpha \nabla_\theta V_\theta(s_t, a_t) \times \left( y(s_t, a_t) + yQ_\theta(s_{t+1}, \arg\max_{a'\neq a} Q_\theta(s_{t+1}, a')) - Q_\theta(s_t, a_t) \right).$$

In the update of the target network, the action network and the evaluation network have the same initialization condition. In order to reduce the convergence difficulty of training fluctuation caused by the synchronous update of action network and evaluation network, we use the delayed update commonly used in Q-Learning, that is, the evaluation network is updated synchronously after the action network is updated for several epochs. Finally, the new action network will continue to be used in the next MCTS, and the new evaluation network will be used to evaluate the selected actions of the action network.

4.3.3. Model Prediction. In order to make full use of the advantages of network structure, MCTS and Q value are combined to generate Monte Carlo Tree Search in the prediction process, just like the training process. In the process of searching, paths are constantly generated. At the end of the search, the value of feedback propagation will no longer use the tag value of 0 and 1, but use $V_\theta(s_T) = Q(s_T, a_{stop})$ instead. Finally, the generated multiple paths are weighted to obtain the path score $\text{Score(path)} = n_{path}/N$, where the number of all generated paths is $N$, and the number of path is $n_{path}$, and $\text{Score(path)}$ represents the proportion of the current path score. Therefore, the final prediction score of target entity is as follows:

$$\text{Sscore}(n_T) = \text{Sscore(path}_{n_T}) \times Q(s_{T_{path}}, a_{stop}),$$

where $\text{path}_{n_T}$ represents the path with termination node $n_T$ and $s_{T_{path}}$ represents the termination state $s_T$ of the path. It is conceivable that paths with termination node $n_T$ may have different termination states. Therefore, when we consider the case of swimming to the same termination node $n_T$ through different paths, the score is calculated according to the termination action value $Q(s_{T_{path}}, a_{stop})$ given by the agent.
network in the current state. According to the frequency and termination state of the target node in the search process, Scosε(πtj) fully integrate agent network decision and network structure data. Then the completion nodes are sorted according to the score. Finally, many information retrieval evaluation algorithms, such as MRR and MAP, will be used to evaluate the sorting sequence of the completion results.

5. Experiments

5.1. Datasets and Evaluation Metrics. We use WN18RR [48] and NELL995 to verify the actual effect of our model GA. The statistic of knowledge graph completion datasets is showed in Table 1. The WN18RR datasets were obtained from the original WN18 data [49] processed by Dettmers.T. NELL995 datasets are processed and published by Xiong [19], and are processed into a separate data part for multiple relationships. We use the same data segmentation and preprocessing methods as in [19–21]. Experiments and evaluations were conducted on 10 relational tasks in NELL995.

We use HITS@K and mean recurrent rank (MRR) as the evaluation metrics of WN18RR. The mean average precision (MAP) is used as the evaluation metric of NELL995 dataset. HITS@K calculates the percentage of the target entity in the top K. MRR calculates the average value of the reciprocal of the target entity ranking, indicating that the higher the target entity ranks, the higher the MRR scores. GA model is compared with DeepPath and MINERVA which are based on reinforcement learning in the completion task of knowledge graph, the embedding based methods DistMult [50], ComplEx [51] and TransE, and the method of learning logic rules NeuralLP [52]. For all baseline methods, we will adopt the experimental setting published by the corresponding authors and the best experimental results in their papers.

5.2. Experimental Results. The overall performances of datasets NELL995 and WN18RR are shown in Tables 2 and 3 respectively. It is obvious that GA generally achieves the best results in NELL995 compared with MINERVA, DeepPath, PRA, TransE and TransR. And GA still performs well in WN18RR than MINERVA, ComplEx, ConvE, DistMult and NeuralLP.

From the experimental results in Tables 2 and 3, it can be seen that the temporal convolutional network can be used to encode the historical state on the knowledge graph, which is similar to the recurrent neural network encoding. And it can get better results in some relatively long relationship paths.

Among the ten relationships of NELL995, GA almost has the best results. In the training process of WN18RR, it can be clearly found that most of the target entities and the source entities are only connected by one hop relation. Therefore, using HITS as the evaluation metrics, the effect of GA is often the best in the contrast experiment.

5.3. Effect of Policy-UCT Algorithm. Monte Carlo Tree Search can fully return positive reward according to the reward function, as shown in Figures 6–8 is the process of forecast generation path for the datasets on Figures 6 and 7. Obviously, MCTS makes full use of graph structure similar to tree structure, which greatly enhances the efficiency of MCTS exploration. In the early stage of exploration, we tend to explore in breadth, which is time-consuming and less efficient in getting rewards. In the later stage of exploration, we prefer to choose the target node with a larger value, and the path of each search is basically short. In the experiment, we can often see that the length of generated path with positive reward is generally 2, 3, 4 (the longest exploration path is 10). From the calculation time, it can be seen that the average exploration length of the model is also decreasing as the model explores the environment more comprehensively, and the algorithm is more inclined to the path which is easy to obtain positive reward. Through continuous search in the environment, the average positive reward rate of the algorithm is more than 80%. There are also a lot of repeated positive reward paths in the later stage of Monte Carlo Tree Search. Although the positive rate of reward keeps rising, it also brings the risk that the agent falls into local optimum. Therefore, in the experiment, when the rate of positive rewards is higher than a certain value, the sampling method will use autonomous walk of agent, and the obtained path is added to the trajectory storage pool, and random batch training is used to reduce the correlation between data.

5.4. Interpretability Analysis of the Results. Compared with the popular embedding model, our model has higher interpretability in the completion of knowledge graph. The proposed model uses reinforcement learning to walk from the source node, and each step is decided by the model to determine the relation and the corresponding next node until the model selects the terminal action, which generates a complete path from the source entity to the target entity. Since the path is a collection of entities and relations, we can see the relation transformation of each step of the path. In the face of a problem, people often associate it with others through the relation between the problems. Therefore, the way in which the model generates results is very close to the way of human thinking. In order to verify whether the walking path conforms to human reasoning cognition, in NELL995 dataset, we randomly extracts four completion paths from the completion results for relationship (Atlete Plays For Team), and analyzes the rationality of paths in the whole process of target selection, so as to explain the interpretability of the model in nature rather than quantification.
It can be seen from the above results that the entity to be completed is an athlete, and the corresponding relationship is "which team does the athlete play for". The relationship between the first two items is "the team led by the athletes" that precisely explains which team the athletes play for. In the third one, the path passes through the middle node, and the two relations point from the athletes to the home stadium, and then find the team of the home stadium to determine the team of the athletes. When the home stadium and the sport team are the only corresponding, this path has a very good interpretability. Even if it is not unique corresponding, this path can also be used as the basis of inference to a certain extent. The fourth completion path is relatively complex. Although the model finds the target team through existential relation, it cannot be explained perfectly by relationship. Because the choice of each step of the actual model is determined by the relation and entity. Moreover, "the team against the team" is not specific, which weakens the interpretation ability.

As described in the construction of Markov process, whether the target node can be found as the only reward, the agent only associates the policy with the target node, and the walking path is positively related to the real reason of completing the target node, but not equivalent. However, the interpretation ability of the completion path is still hard to quantify. The same relation in the path has different

---

**Table 2: Knowledge graph completion datasets statistics.**

<table>
<thead>
<tr>
<th>Task (MAP)</th>
<th>GA</th>
<th>MINERVA</th>
<th>DeepPath</th>
<th>PRA</th>
<th>TransE</th>
<th>TransR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AthletePlaysForTeam</td>
<td>85.2 (0.8)</td>
<td>82.7 (0.8)</td>
<td>72.1 (1.2)</td>
<td>54.7</td>
<td>62.7</td>
<td>67.3</td>
</tr>
<tr>
<td>AthletePlaysInLeague</td>
<td>97.9 (0.4)</td>
<td>95.2 (0.8)</td>
<td>92.7 (5.3)</td>
<td>84.1</td>
<td>77.3</td>
<td>91.2</td>
</tr>
<tr>
<td>AthleteHomeStadium</td>
<td>91.7 (0.1)</td>
<td>92.8 (0.1)</td>
<td>84.6 (0.8)</td>
<td>85.9</td>
<td>71.8</td>
<td>72.2</td>
</tr>
<tr>
<td>AthletePlaysSport</td>
<td>96.4 (0.2)</td>
<td>98.6 (0.1)</td>
<td>91.7 (4.1)</td>
<td>47.4</td>
<td>87.6</td>
<td>96.3</td>
</tr>
<tr>
<td>TeamPlaysSports</td>
<td>87.5 (0.2)</td>
<td>87.5 (0.5)</td>
<td>69.6 (6.7)</td>
<td>79.1</td>
<td>76.1</td>
<td>81.4</td>
</tr>
<tr>
<td>OrgHeadquarterCity</td>
<td>95.4 (0.3)</td>
<td>94.5 (0.3)</td>
<td>79.0 (0.0)</td>
<td>81.1</td>
<td>62.0</td>
<td>65.7</td>
</tr>
<tr>
<td>BornLocation</td>
<td>82.5 (0.1)</td>
<td>78.2 (0.0)</td>
<td>75.5 (0.5)</td>
<td>66.8</td>
<td>71.2</td>
<td>81.2</td>
</tr>
<tr>
<td>WorksFor</td>
<td>83.8 (0.4)</td>
<td>82.7 (0.5)</td>
<td>69.9 (0.3)</td>
<td>68.1</td>
<td>67.7</td>
<td>69.2</td>
</tr>
<tr>
<td>PersonLeadsOrg</td>
<td>88.2 (1.3)</td>
<td>83.0 (2.6)</td>
<td>79.0 (1.0)</td>
<td>70.0</td>
<td>75.1</td>
<td>77.2</td>
</tr>
<tr>
<td>OrgHiredPerson</td>
<td>90.1 (0.7)</td>
<td>87.0 (0.3)</td>
<td>73.8 (1.9)</td>
<td>59.9</td>
<td>71.9</td>
<td>73.7</td>
</tr>
<tr>
<td>Overall</td>
<td>89.9</td>
<td>87.6</td>
<td>78.8</td>
<td>69.7</td>
<td>72.3</td>
<td>77.5</td>
</tr>
</tbody>
</table>

**Table 3: Knowledge graph completion datasets statistics.**

<table>
<thead>
<tr>
<th>Metric (%)</th>
<th>GA</th>
<th>MINERVA</th>
<th>ComplEx</th>
<th>ConvE</th>
<th>DistMult</th>
<th>NeurallP</th>
</tr>
</thead>
<tbody>
<tr>
<td>HITS@1</td>
<td>40.2 (0.2)</td>
<td>35.1 (0.1)</td>
<td>38.5 (0.3)</td>
<td>39.6 (0.3)</td>
<td>38.4 (0.4)</td>
<td>37.2 (0.1)</td>
</tr>
<tr>
<td>HITS@3</td>
<td>44.3 (0.3)</td>
<td>44.5 (0.4)</td>
<td>43.9 (0.3)</td>
<td>44.7 (0.2)</td>
<td>42.4 (0.3)</td>
<td>43.4 (0.1)</td>
</tr>
<tr>
<td>MRR</td>
<td>43.5 (0.2)</td>
<td>40.9 (0.1)</td>
<td>42.2 (0.2)</td>
<td>43.3 (0.2)</td>
<td>41.3 (0.3)</td>
<td>43.5 (0.1)</td>
</tr>
</tbody>
</table>

**Figure 6: Positive rate of rewards of Policy-UCT search on each dataset (C = 1.96).**
interpretations in different query relation. For example, if the relation to be queried is “Athlete Plays In League,” then “team against team” can become a bridge to the goal and has excellent interpretation ability, because the League of the played against team is also the League, in which the athlete plays. Therefore, the interpretability of the model needs further study.

6. Conclusion

This paper summarizes and expands on the basis of predecessors, constructs reinforcement learning Markov process description on network structure, and designs environment exploration and deep agent network. The definition of Markov process on knowledge graph is also applicable to other forms of graph structure. Considering that the states in the knowledge graph environment are represented by sequence data, the temporal convolution network which is easy to parallelize is usually used for feature extraction and encoding of states. The flexible convolution kernel size is more suitable for capturing recursive state structure. Compared with the output and input of the recurrent neural network, it greatly reduces the time-consuming. In addition, due to the complexity of network structure and cross-linking, it is difficult to train the agent network effectively because of the limited ability of searching positive reward. In this paper, we use M-walk and AlphaGO environment exploration method Monte Carlo Tree Search, which greatly increases the acquisition rate of positive rewards and makes full use of the similarity between graph and search tree. The average positive reward in the test dataset is more than 80%. After sufficient training, the agent can accurately walk to the target node. We use a reasonable positive correlation hypothesis, but the real completion path corresponding to the relation is not completely equivalent to the path generated by using the positive reward. In the following work, we can consider a more close set of positive correlation hypothesis and describe it in the form of a new return function with additional constraints. The interpretability of the completion model also has some challenges and can be a part of future work.

Data Availability

Our datasets are available at https://github.com/yelongshen/GraphWalk/tree/master/dataset.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Foundation of the National Natural Science Foundation of China (no. 61872161), the Jilin Educational Committee (no. JJKH20191257J), the Nature Science Foundation of Jilin Province (no. 20200201297J), the Foundation of Development and
Reform of Jilin Province (no. 2019C053-8), and the Interdisciplinary and integrated innovation of JLU (no. JLU2020C020207).

References


