Curriculum Learning for Heterogeneous Star Network Embedding via Deep Reinforcement Learning

Meng Qu
University of Illinois
at Urbana-Champaign
mengqu2@illinois.edu

Jian Tang
HEC Montreal & Montreal
Institute for Learning Algorithms
tangjianpku@gmail.com

Jiawei Han
University of Illinois
at Urbana-Champaign
hanj@illinois.edu

ABSTRACT

Learning node representations for networks has attracted much attention recently due to its effectiveness in a variety of applications. This paper focuses on learning node representations for heterogeneous star networks, which have a center node type linked with multiple attribute node types through different types of edges. In heterogeneous star networks, we observe that the training order of different types of edges affects the learning performance significantly. Therefore we study learning curricula for node representation learning in heterogeneous star networks, i.e., learning an optimal sequence of edges of different types for the node representation learning process. We formulate the problem as a Markov decision process, with the action as selecting a specific type of edges for learning or terminating the training process, and the state as the sequence of edge types selected so far. The reward is calculated as the performance on external tasks with node representations as features, and the goal is to take a series of actions to maximize the cumulative rewards. We propose an approach based on deep reinforcement learning for this problem. Our approach leverages LSTM models to encode states and further estimate the expected cumulative reward of each state-action pair, which essentially measures the long-term performance of different actions at each state. Experimental results on real-world heterogeneous star networks demonstrate the effectiveness and efficiency of our approach over competitive baseline approaches.

ACM Reference Format:
Meng Qu, Jian Tang, and Jiawei Han. 2018. Curriculum Learning for Heterogeneous Star Network Embedding via Deep Reinforcement Learning. In *WSDM 2018: WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining , February 5–9, 2018, Marina Del Rey, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3159652.3159711

1 INTRODUCTION

Heterogeneous networks, which encode the relationships between different types of objects, are ubiquitous in the real world. Mining heterogeneous networks has been attracting growing attention in recent years. Among all heterogeneous networks, the star network [33] is popular and important. A star network has a center node type and multiple attribute node types, which link to the center nodes through different types of edges. Figure 1 presents an example of the bibliography star network. The center nodes are papers, linked with authors, venues, keywords and references through four types of edges. Analyzing star networks is an important problem in data mining, since a variety of real-world applications can be formulated as certain problems on star networks, such as author identification [5], predictive text embedding [37] and user attribute prediction [18].

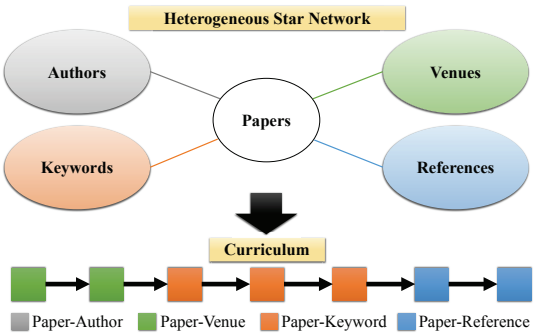


Figure 1: An example of heterogeneous star networks. Papers are the center nodes. Authors, venues, keywords, references are the attribute nodes, which connect to the papers through four types of edges. Our goal is to learn a sequence of edge types for node representation learning.

To mine heterogeneous star networks more effectively, it is helpful to learn meaningful node representations. Traditionally, nodes are represented as bag-of-neighbors, which are both high-dimensional and sparse. Recently, there is a growing trend to embed networks into low-dimensional spaces [11, 27, 38], in which each node is represented as a low-dimensional vector. The learned node representations capture the proximities between nodes, which can benefit various applications, such as node classification [27], link prediction [11] and node visualization [36]. The essential idea for node representation learning is to capture the node proximities encoded in edges. In practice, existing approaches usually sample a set of edges as training data at each learning step. To learn node representations for heterogeneous star networks, a natural solution could be applying these approaches and sampling edges of different types as training data.

Towards the goal of sampling edges in heterogeneous star networks, existing studies usually leverage the random sampling [12, 37] or weighted sampling [5, 18] strategy, without considering the relative order of edges of different types. However, each type of edges encodes a specific kind of knowledge, which may benefit the representation learning process at different training stages. In other

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WSDM 2018, February 5–9, 2018, Marina Del Rey, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5581-0/18/02…$15.00
https://doi.org/10.1145/3159652.3159711

words, the training order of edge types matters during training. For example, in Figure 1, consider learning paper representations that preserve paper semantic meanings. The venue of each paper reflects its coarse semantic domains, whereas the keywords and references capture more concrete semantic meanings. Inspired by the human learning process, the coarse meanings are easier to understand, and may benefit the learning of more concrete semantics. Based on the observation, a more reasonable strategy compared with random or weighted sampling could be first learning from the paper-venue edges, and then moving to the paper-keyword and paper-reference edges. Therefore, the training order of different edge types is likely to affect the representation learning performance significantly.

Indeed, in the machine learning literature, the order of training data has been proved to be an important factor in many applications. Learning a meaningful order of the training data (a.k.a., curriculum learning) can benefit various tasks such as shape recognition [4], handwritten text line recognition [19], word representation learning [41], sequence prediction [3] and many others. The basic idea is to start small [6], that is, selecting easier examples to learn at first, and then gradually increasing the difficulty. Though curriculum learning is widely studied, how to learn a meaningful order of training data remains unexplored for learning node representations in heterogeneous star networks.

This motivated us to study a new problem: *curriculum learning for node representation learning in heterogeneous star networks*, aiming to learn an optimal curriculum for node representation learning, in which a curriculum is defined as a sequence of edge types used for training (Figure 1). The problem is essentially a sequential decision making task, and we formulate it as a Markov decision process [2]. At each step, our action is to select a certain type of edges for node representation learning, or terminate the training process, and the state is defined as the sequence of edge types selected so far. After taking an action at a state, we will move to another state and receive a scalar reward. The goal is to learn a sequence of edge types that maximizes the total sum of the rewards. Despite its practical importance, the task is nontrivial, as the search space is exponential to the sequence length. Therefore, we are seeking an approach which can learn an effective curriculum efficiently.

In this paper, we propose such an approach based on deep reinforcement learning. Our approach learns the optimal curriculum by estimating the Q value of each state-action pair, which is defined as the expected cumulative reward after taking the action from the state. Once the Q values are learned, the optimal curriculum can be determined by sequentially selecting the action with the maximum Q value at each step. In our approach, we learn the Q value from two sources, i.e., a planning module and a learning module. Given a state, the planning module calculates Q by *looking ahead*, which explores some subsequent actions through simulations and approximates Q with the simulated rewards. On the other hand, the learning module estimates Q by *looking back* on the past experience. Specifically, it employs a deep neural network (LSTM models [13]) to learn from the past experience and further make predictions. With both modules, our approach can estimate the Q value with high accuracy and low time costs. Therefore, we are able to learn a meaningful curriculum both effectively and efficiently.

We conduct experiments on several real-world heterogeneous star networks under the task of node classification. Experimental results in both unsupervised and semi-supervised settings prove the effectiveness and efficiency of our proposed approach.

To summarize,we make the following contributions:

- We define a new problem of *curriculum learning for node representation learning in heterogeneous star networks*, aiming to learn a sequence of edge types for node representation learning.
- We formulate the problem as a Markov decision process, and propose an approach based on deep reinforcement learning.
- We conduct experiments on real-world star networks, which prove the effectiveness and efficiency of our proposed approach.

2 PROBLEM DEFINITION

Heterogeneous networks encode the relationships between different types of objects, which are widely studied recently. In this paper, we focus on a special type of heterogeneous networks, the heterogeneous star network, which is formally defined below:

Definition 2.1. **(Heterogeneous Star Network.)** *A **Heterogeneous Star Network** $G = (V_0 \cup \{V_k\}_{k=1}^K, \{E_k\}_{k=1}^K)$ contains a set of center nodes V_0 and different types of attribute nodes $\{V_k\}_{k=1}^K$, which connect to the center nodes V_0 through different types of edges $E_k = (V_0, V_k)$. Each edge is associated with a weight $w > 0$, indicating the strength of the relationship between the linked nodes.*

An example of heterogeneous star networks is the bibliography star network (Figure 1), in which the center nodes are papers, with the attribute nodes as authors, venues, keywords and references. There are four types of edges in the network: paper-author, paper-venue, paper-keyword and paper-reference.

For many problems of heterogeneous star network analysis, it is critical to learn meaningful node representations. To learn such representations, we can directly apply existing node representation learning algorithms [11, 27, 38] by sampling a set of edges of different types as training data. In practice, we observe that the order of the sampled edges affects the representation learning performance significantly. Therefore we study learning curricula for node representation learning in heterogeneous star networks. In other words, we aim at learning a meaningful order of different types of edges for the given node representation learning algorithm.

The problem is essentially a sequential decision making task, which can be naturally formulated as a Markov decision process. At each step, the **action** is to select a certain edge type and leverage edges of that type for node representation learning (denoted as $a = y_t$ where $y_t \in \{1, 2...K\}$ represents an edge type), or terminate the training process (denoted as $a = STOP$). The **state** is defined as the sequence of edge types selected so far (denoted as $s = (y_1, y_2...y_{t-1})$). After taking an action at a state, we will move to another state and receive a **reward**, which is calculated by a given reward function (denoted as $R(s, a)$ where s is the current state and a is the action we take). For different tasks, we may select different reward functions. For example, in the node classification task, we can define the reward as the accuracy gain after taking an action at a state, with the accuracy calculated on a held-out dataset. Given the above definition, our **goal** is to take a series of actions to maximize the cumulative rewards. In other words, we aim at learning an optimal sequence of edge types for node representation learning. Formally, we define our problem as follows:

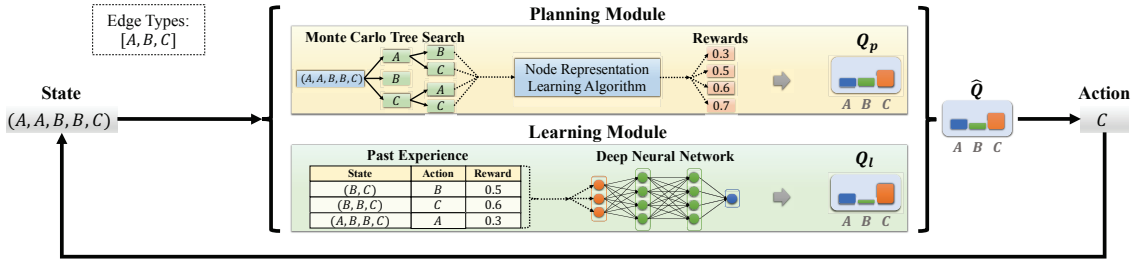


Figure 2: Framework overview. We learn curricula by sequentially selecting actions with the maximum Q values. Given a state, the action Q values are estimated from two modules. The planning module looks ahead, which simulates some actions and approximates Q with the simulated rewards, whereas the learning module looks back on the past experience for prediction.

Definition 2.2. **(Problem Definition.)** *Given a heterogeneous star network $G = (V_0 \cup \{V_k\}_{k=1}^K, \{E_k\}_{k=1}^K)$ and a reward function $R(s, a)$ for each state-action pair (s, a), we aim at taking a series of actions to maximize the total cumulative rewards. In other words, our goal is to learn a sequence of edge types for training.*

3 PRELIMINARY

In this section, we introduce LINE [38], which is a representative node representation learning algorithm. Given a network, LINE samples a number of edges from the network as training data for node representation learning.

Specifically, LINE defines the probability of a node u generated by another node v as follows:

$$p(u|v) = \frac{\exp(\mathbf{v}^T \mathbf{u})}{\sum_{u' \in V} \exp(\mathbf{v}^T \mathbf{u'})}, \tag{1}$$

where $\mathbf{x}$ is the representation of each node x, and V is the node set.

To preserve the relationships between nodes, LINE minimizes the KL-divergence between the estimated neighborhood distribution $p(\cdot|v)$ and the empirical neighborhood distribution $\hat{p}(\cdot|v)$ for every node v. The empirical distribution is defined as $\hat{p}(u|v) = w_{uv}/d_v$, where w_{uv} is the weight of the edge (u, v) and d_v is the degree of v. The final objective function of LINE can be simplified as:

$$O_{line} = -\sum_{(u,v)\in E} w_{uv} \log p(u|v). \tag{2}$$

Directly optimizing the above objective is computationally expensive because it involves traversing all nodes when computing the softmax function. Therefore LINE adopts the negative sampling techniques [20, 21], which modify the conditional probability $p(u|v)$ in Eqn. 2 as follows:

$$\log \sigma(\mathbf{u}^T \mathbf{v}) + \sum_{n=1}^{N} E_{v' \sim P_{neg}(v)} [\log \sigma(-\mathbf{u}^T \mathbf{v'})], \tag{3}$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function. The first term tries to maximize the probabilities of some observed edges (u, v), and the second term tries to minimize the probabilities of N noisy edges (u, v'), with v' sampled from a noisy distribution $P_{neg}(v) \propto d_v^{3/4}$ and d_v is the degree of node v in the network.

The objective function can be efficiently optimized with edge sampling [38]. In each iteration, the algorithm randomly samples an observed edge with N noisy edges, and then maximizes Eqn. 3.

Next, we take LINE as an example to introduce our approach.

4 METHODOLOGY

In this section, we introduce a deep reinforcement learning approach to the proposed curriculum learning problem. Given a heterogeneous star network, our action at each step is to select an edge type for representation learning or terminate the training process, and the state is the sequence of edge types selected so far. The reward of each action is calculated on an external task, and our goal is to take a series of actions to maximize the cumulative rewards. In other words, we aim at learning the optimal sequence of edge types for training. As the number of possible sequences is exponentially large, learning an effective sequence efficiently is very challenging.

We learn such sequence by estimating the Q value $Q(s, a)$ of each state-action pair (s, a), which is defined as the expected cumulative reward after taking action a from state s. Once the Q value is learned, the optimal series of actions can be easily determined by sequentially selecting the action with the maximum Q value.

Our approach predicts Q from two sources, i.e., a *planning* module and a *learning* module, and we denote the values calculated by them as Q_p and Q_l respectively. After Q_p and Q_l are learned, we further combine them as a more precise estimation $\widehat{Q}$. To learn Q values, given an edge type sequence as state, the planning module will simulate some actions, which either select a type of edges as training data or stop the training process, then the obtained rewards are leveraged to estimate Q_p. Different from the planning module, the learning module learns from the previous simulation results, and infers Q_l based on these past experience. With both modules, our approach can effectively predict the Q value (see Figure 7(a)). Moreover, by carefully utilizing the past experience with the learning module, we can also calculate the Q value very efficiently (see Figure 7(b)).

Even though our approach can learn a curriculum both effectively and efficiently, in practice, we may still have different emphasis on effectiveness and efficiency. To allow flexible trade-off between them, our approach tries to penalize each action. Specifically, when calculating the reward for an action, we will subtract a constant penalty from the original reward calculated by the reward function $R(s, a)$. A small penalty encourages our approach to learn a longer curriculum, which has better performance but is less efficient; whereas a large penalty leads to a shorter curriculum, with worse performance but less time costs. Overall, the two factors can be well balanced by choosing different penalties (see Figure 8).

The overall framework is summarized in Figure 2. Next, we introduce the details of our approach and analyze its time complexity.

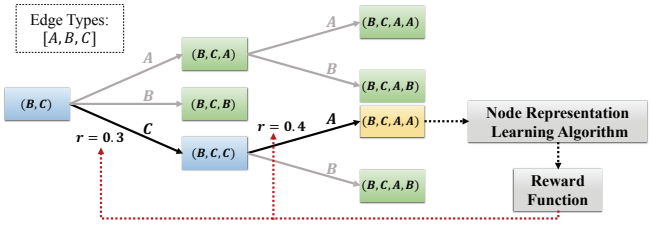


Figure 3: Illustration of the simulation. Given a state (the leftmost one), we simulate a series of actions (actions along the black path) until we reach an unvisited state (the yellow one). Then the rewards are calculated to approximate Q_t.

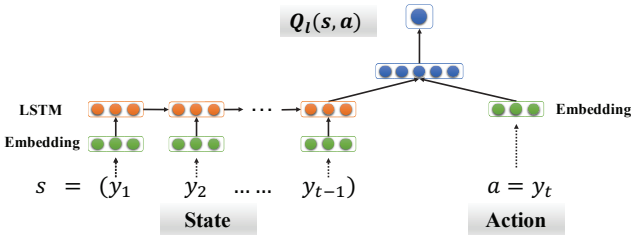


Figure 4: The network structure for the learning module.

4.1 Planning Module

The planning module of our approach estimates Q_p by looking ahead and simulating some subsequent actions starting from the given state. In each simulation, we first choose a series of actions to explore, which either select a type of edges for training or terminate the training process. Then we simulate the actions with the node representation learning algorithm and calculate the rewards, which are further utilized to approximate the expected cumulative reward Q_p. Figure 3 presents an illustration of the workflow.

Specifically, in the planning module, the $Q_p(s, a)$ value of each state-action pair (s, a) is calculated with a look-up table. Following existing studies on Monte-Carlo tree search [1, 8, 14], given a state s, in each simulation we will recursively select some actions to explore (e.g., actions along the black path in Figure 3), until we reach an unvisited state (e.g., the yellow state in Figure 3). Inspired by the UCT algorithm [8], at each state s, we select the following action a to explore:

$$a = \arg\max_a \{\frac{Q_p(s,a)N(s,a)}{N(s,a)+1} + \frac{Q_l(s,a)}{N(s,a)+1} + \lambda\sqrt{\frac{\ln N(s)}{N(s,a)+1}}\}. \quad (4)$$

For each state-action pair (s, a), $Q_p(s, a)$ and $Q_l(s, a)$ are the Q values calculated by the planning and learning modules respectively, $N(s, a)$ is the visit count and $N(s) = \sum_a N(s, a)$ is the total visit count of the state s.

Such selection rule is quite intuitive. The first term is the Q value calculated by the planning module; while the second term is calculated by the learning module, which serves as priors and decays with repeated visits. Both terms encourage the model to exploit actions with larger Q values in the past, as these actions are more likely to be the optimal ones. For the third term, it favors those actions with less visit counts, as such actions can eventually become superior than others in some cases. To balance the exploitation of the promising actions with the exploration of others, we introduce a parameter λ. A small λ will encourage the exploitation while a large one will encourage the exploration.

After reaching an unvisited state, we will simulate the selected actions with the node representation learning algorithm, that is, leveraging the corresponding types of edges to update the node representations or stop the training process. Then the rewards are calculated by applying the reward function on the learned node representations. Suppose the sequence of the visited states and the selected actions is $(s_t, a_t, ..., s_{t+l}, a_{t+l}, s_u)$, and the corresponding reward sequence is $(r_t, ..., r_{t+l})$ where $r_i = R(s_i, a_i)$. Then we update Q_p based on the temporal difference learning method [35]:

$$Q_p(s_i, a_i) = Q_p(s_i, a_i) + \alpha[r_i + Q_p(s_{i+1}, a_{i+1}) - Q_p(s_i, a_i)], \quad (5)$$

where $\alpha = \frac{1}{N(s_i,a_i)}$ is the learning rate. Basically, r_i is the immediate reward after taking a_i at s_i and $Q_p(s_{i+1}, a_{i+1})$ estimates the long-term reward, and we will use their sum to approximate the expected total reward $Q_p(s_i, a_i)$ from state s_i after taking action a_i.

4.2 Learning Module

Different from the planning module, the learning module of our approach estimates Q_l by looking back on the past experience, without look-ahead search. More specifically, we train a deep neural network to memorize the historical data, that is, the previously explored state-action pairs together with the simulated rewards. Then we infer Q_l based on the neural network.

Formally, the value $Q_l(s, a)$ of each state-action pair (s, a) is calculated by a deep neural network. In the deep neural network, we represent each edge type $y \in \{1, 2...K\}$ and action $a \in \{1, 2...K\}$ with an embedding vector. Then for a state $s = (y_1, y_2...y_t)$, we encode it using an LSTM layer [13]. After that, we concatenate the encoding vectors of state s and action a, and leverage two fully connected layers to calculate $Q_l(s, a)$. Figure 4 shows the structure of the neural network. By leveraging LSTM layers to encode states, we can effectively capture the correlations of different states, which enables us to effectively infer the Q_l values of new state-action pairs based on their similarities with the previously explored pairs.

To learn the parameters of the deep neural network, we treat the state-action pairs and the corresponding rewards explored by the planning module as training data. Formally, suppose the state-action sequence obtained in a simulation is $(s_t, a_t, ..., s_{t+l}, a_{t+l}, s_u)$, and the corresponding reward sequence is $(r_t, ..., r_{t+l})$, then we update the parameters based on the temporal difference learning method [35] as follows:

$$w_l = w_l + \alpha[r_i + Q_l(s_{i+1}, a_{i+1}) - Q_l(s_i, a_i)] \bigtriangledown_{w_l} Q_l(s_i, a_i), \quad (6)$$

where w_l is the parameter set of the neural network, α is the learning rate, which is updated with the RMSProp algorithm [40] and the initial value is set as 0.001. Basically, the immediate rewards r_i and the long-term reward $Q_l(s_{i+1}, a_{i+1})$ are leveraged to approximate the cumulative reward $Q_l(s_i, a_i)$ of the state-action pair (s_i, a_i).

4.3 Integrating Both Modules

Finally in each step, Q_p and Q_l are integrated as $\widehat{Q}$, which gives a more precise estimation of the Q value. Then we decide the optimal action with the integrated value.

Specifically, given the current state s, we calculate $\widehat{Q}(s,a)$ for each action a as follows:

$$\widehat{Q}(s,a) = \{\frac{Q_p(s,a)N(s,a)}{N(s,a)+1} + \frac{Q_l(s,a)}{N(s,a)+1}\}, \quad (7)$$

where $Q_p(s,a)$ is calculated by the planning module, $Q_l(s,a)$ is calculated by the learning module, and $N(s,a)$ is the visit count. We see that $\widehat{Q}$ is a weighted average of Q_p and Q_l. As Q_p is estimated with look-ahead search, which is more precise, we will assign larger weight to this term. For Q_l, it is learned from the past experience, which serves as priors and we will continuously decrease its weight with repeated visits.

The $\widehat{Q}$ value estimates the expected future rewards after taking a at s, and the action a^* with the maximum $\widehat{Q}$ value should be the optimal selection at the current step. Therefore, we will take action a^*, that is, either selecting a type of edges for node representation learning or terminating the training process.

The overall node representation learning process with our curriculum learning approach is summarized in Algorithm 1.

Algorithm 1 Representation learning with our approach.

Input: Star network G, reward function R, the number of simulations S.
Output: Node representations.

for each step **do**
  ⊡ ***Update the learning module:***
  Collect past experience as training data for the learning module.
  Update the learning module according to Eqn. (6).
  ⊡ ***Update the planning module:***
  while simulation $\leq S$ **do**
    Select a series actions according to Eqn. (4).
    Simulate the actions with the embedding algorithm (Eqn. (2)).
    Calculate the rewards with the reward function R.
    Update the planning module according to Eqn. (5).
  end while
  ⊡ ***Integrating both modules:***
  Calculate $\widehat{Q}$ value for each action according to Eqn. (7).
  Select the optimal action a^* based on $\widehat{Q}$.
  if a^* is $STOP$ **then**
    Terminate the training process.
  end if
  Use the corresponding type of edges for training based on Eqn. (2).
  Move to the next state based on the selected action a^*.
end for

4.4 Time Complexity

Finally, we analyze the time complexity of the node representation learning process with our approach. In each step, we will simulate a series of actions, update both modules and decide the optimal action. The time costs mainly come from the simulation process. Suppose we conduct S simulations in each step, and we randomly sample E_s edges for training if an edge type is selected in an action. Then the overall time complexity is $O(ISE_s)$, where I is the number of training steps. Compared with single edge type based approaches (e.g., DeepWalk, LINE, node2vec), the time complexity of our approach ($O(ISE_s)$) is S time larger than theirs ($O(IE_s)$). In the experiment part, we will show that our approach requires very limited simulations S to achieve satisfactory results (see Figure 7). We will also show that our approach is quite efficient compared with several baseline approaches (see Figure 6).

5 EXPERIMENT

In this section, we evaluate our approach on several real-world heterogeneous star networks. We compare different approaches on the center node classification task in both the unsupervised and semi-supervised settings.

Specifically, for each compared algorithm, we treat the learned center node representations as features and train one-vs-rest linear classifiers using the LibLinear package [7] [1] for classification. In the unsupervised setting, we treat all center nodes in the training set as the held-out data, and the **reward** of an action is defined as the classification accuracy gain on the held-out dataset after taking that action. Note that in the unsupervised setting, the labeled center nodes are only used for reward calculation. In some cases, we also expect to improve node representations with the labeling information. Therefore, in the semi-supervised setting, we treat labels as an additional type of attribute nodes to guide the learning process. We use 70% center nodes in the training set to construct edges between center nodes and labels, and treat the remaining 30% as the held-out data. The **reward** of an action is defined as the classification accuracy gain on the held-out dataset after taking that action. The classification performance is evaluated using the Macro-F1 and Micro-F1 scores. Note that Micro-F1 is equal to accuracy in our setting since each node has only one label.

5.1 Experiment Setup

5.1.1 *Datasets.*

- **DBLP**: A bibliography star network constructed from the DBLP dataset [39][2]. The papers are the center nodes and the authors, citations, words in the titles are treated as the attribute nodes. The weights of all edges are set as 1. For the center nodes (i.e., papers), we select eight diverse research fields as labels including "machine learning", "natural language processing", "programming language", "data mining", "database", "system technology", "hardware and theory". For each research domain, several representative conferences are selected, and only papers published in these conferences are collected to construct the networks.
- **Yelp**: A business network constructed from the Yelp dataset [3], with business as the center node type, and user, word in business names, word in business reviews as attribute nodes. The weight of the edge between a business and a user is the number of reviews written by the user for the business. The weight of the edge between a business and a word is the frequency of the word in the business name/reviews. We select six categories including "Restaurants", "Hotels", "Shopping", "Health & Medical", "Beauty & Spas", "Arts & Entertainment" as labels. Businesses with multiple labels are excluded from the labeled set.
- **IMDB**: A movie network constructed from the IMDB dataset [4]. We treat the movies as the center nodes and other entities as the attribute nodes, including users, movies, keywords, actors, actresses, directors, writers. For each (movie, user) pair, the edge weight is set as 1 if the user once rated the movie. For each (movie, movie) pair, the weight is defined as the number of users who

[1] https://www.csie.ntu.edu.tw/~cjlin/liblinear/
[2] https://aminer.org/AMinerNetwork
[3] https://www.yelp.com/dataset_challenge
[4] http://files.grouplens.org/datasets/movielens/ml-10m-README.html

Table 1: Statistics of the Datasets.

Dataset	DBLP			Yelp			IMDB						
# Center Nodes	107,833			77,445			10,692						
Attribute Type	Citation	Author	Word	Review	Name	User	User	Movie	Keyword	Actor	Actress	Director	Writer
# Attribute Nodes	107,833	46,297	23,818	127,203	29,791	552,339	69,878	10,676	58,952	54,171	104,473	3,127	8,164
# Edges	655,030	117,415	346,100	20,858,371	218,034	2,225,213	10,000,054	5,318,912	564,019	90,027	206,290	6,698	15,917
# Training Nodes	13,643			17,468			1,006						
# Test Nodes	30,000			30,000			3,000						

watch one movie immediately after watching the other one. We treat the movie genres as classification labels and movies with more than one genre are excluded from the labeled set.

5.1.2 Compared algorithms.

- **LINE**: A node representation learning algorithm for networks with a single type of edges [38]. We report the best results of LINE with each single type of edges (**LINE**). Besides, we also apply LINE to multiple types of edges by assigning different sampling weights to different edge types, with the weights learned by grid search on the held-out dataset (**LINE-Weight**). Such weight selection strategy is widely adopted by existing methods [5, 18].
- **node2vec**: Another node representation learning approach for a single edge type [11]. Similarly, the best results on a single type of edges (**node2vec**) and the results on multiple types of edges through weight learning (**node2vec-Weight**) are reported.
- **Rand**: Randomly select a type of edges at each training step, and use LINE for node representation learning [37]. The number of training steps is set as 300 in all datasets to ensure convergence.
- **Greedy**: Greedily select the action (i.e., selecting an edge type or terminating training) with the maximum immediate reward at each step, and learn node representations with LINE.
- **DRL**: Learn curricula with our Deep Reinforcement Learning approach, and learn node representations with LINE.
- **DRL-Shuf**: Shuffle the curriculum learned by our approach, and learn node representations with LINE.
- **DRL-P**: Learn curricula with only the planning module, and keep Q_l calculated by the learning module as 0.
- **DRL-L**: Learn curricula with only the learning module, and keep Q_p calculated by the planning module as 0.

5.1.3 Parameter Settings. For all approaches, we set the dimension of the node representations as 100. For node2vec, we set the window size as 10, the walk length as 40, as suggested in [27]. The parameters p and q for controlling the random walk process are selected on the held-out dataset. For LINE, the number of negative samples is set as 5, and the learning rate is set as 0.015. For all the curriculum learning based approaches, if an action is to select an edge type for training, we will randomly sample 1M edges of that type as training data to update the node representations. For the learning module of our approach, the dimensions of the embedding layer and the hidden layer are set as 10. For the planning module of our approach, the parameter λ for balancing the exploitation and exploration is set as 0.2, and the penalty of each action is set as 10^{-4} by default. The number of simulations at each step is set as 12 for the DBLP and Yelp datasets, and 20 for the IMDB dataset.

5.2 Quantitative Results

In this section, we report the quantitative results on the center node classification task in both the unsupervised setting and the semi-supervised setting.

5.2.1 Unsupervised Setting. Table 2 presents the results in the unsupervised setting. We see that considering multiple types of edges achieves much better results compared with single type based approaches (single type). For different strategies combining multiple edge types, we observe that learning their orders (curriculum learning) consistently outperforms learning the sampling weights (weight learning), which demonstrates that curriculum learning is a more effective strategy for the node representation learning problem in star networks. For different curriculum learning approaches, the performance of random sampling (Rand) is the worst. The approach of greedily selecting the action with the maximum immediate reward (Greedy) performs much better.

With both the learning and planning modules, our deep reinforcement learning based approach (DRL) significantly improves the performance. Comparing our approach with its variants, we see that ignoring either the planning module or the learning module (DRL-P and DRL-L) leads to much worse results, which proves that the two modules can indeed mutually complement to improve the performance. Besides, if we shuffle the learned curriculum (DRL-Shuf), the results significantly drop. This shows that the training order of different edge types can indeed affect the performance. Overall, our approach can learn an effective curriculum to improve the node representation learning process.

5.2.2 Semi-supervised Setting. The results in the semi-supervised setting are presented in Table 3. Similar results are observed as in the unsupervised setting. The curriculum learning based approaches still outperform those weight learning based approaches. Besides, our deep reinforcement learning based approach (DRL) significantly outperforms other curriculum learning approaches with random (Rand) or greedy (Greedy) strategies.

Comparing Table 3 with Table 2, we can see that the results of our approach (DRL) are consistently improved by considering the labeled nodes during training, which shows that our approach can also effectively integrate the labeled and unlabeled information to learn more effective node representations.

5.3 Convergence Comparison

The quantitative results above show that our curriculum learning approach is able to learn an effective sequence of edge types for node representation learning. To intuitively understand how such strategy affects the training process, in this part we report the performance of different approaches at each training step. We take the DBLP dataset as an example, and the results in both settings are reported. To ensure the convergence of all approaches, we constrain the number of training steps as 50.

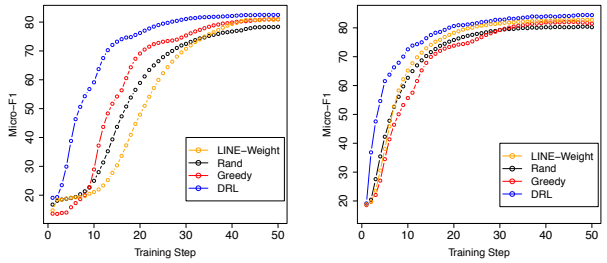
Figure 5 presents the results. Compared with the weight learning based approaches (LINE-Weight) and other curriculum learning based approaches (Rand and Greedy), DRL converges faster and has better performance. Therefore, our approach can both help accelerate the convergence and improve the learned representations.

Table 2: Quantitative results in the unsupervised setting. Curriculum learning based approaches outperform other approaches. Our deep reinforcement learning based approach performs the best among all compared algorithms.

Type	Algorithm	DBLP		Yelp		IMDB	
		Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
Single Type	LINE	75.59	76.26	80.91	87.63	28.26	72.60
	node2vec	78.40	79.79	72.44	81.05	24.28	70.13
Weight Learning	LINE-Weight	79.76	80.99	86.78	91.32	30.63	75.60
	node2vec-Weight	80.50	81.42	85.42	90.63	26.43	74.03
Curriculum Learning	Rand	76.85	78.37	82.16	88.61	28.35	73.93
	Greedy	79.60	80.96	85.89	91.02	28.81	74.57
	DRL-P	81.04	81.95	88.87	92.85	29.55	75.93
	DRL-L	80.15	81.18	86.43	91.46	29.19	72.70
	DRL-Shuf	79.36	80.42	86.28	91.20	28.29	75.37
	DRL-MCT	**81.33**	**82.46**	**89.30**	**93.31**	**33.09**	**78.60**

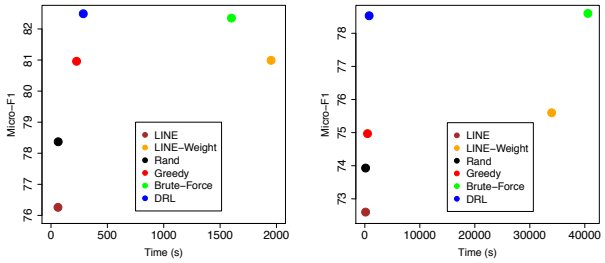
Table 3: Quantitative results in the semi-supervised setting. Our deep reinforcement learning based approach outperforms all other approaches and achieves better results compared with the results in the unsupervised setting.

Type	Algorithm	DBLP		Yelp		IMDB	
		Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
Single Type	LINE	75.59	76.26	80.91	87.63	28.26	72.60
	node2vec	78.40	79.79	72.44	81.05	24.28	70.13
Weight Learning	LINE-Weight	82.09	82.75	87.41	91.85	31.68	76.73
	node2vec-Weight	82.17	83.12	87.05	91.18	27.14	74.57
Curriculum Learning	Rand	79.88	80.23	88.30	89.23	28.79	74.40
	Greedy	80.26	81.49	87.42	92.17	30.08	76.33
	DRL-P	82.98	83.80	89.09	93.39	31.62	77.13
	DRL-L	82.26	83.16	88.07	92.41	29.52	73.87
	DRL-Shuf	81.73	82.22	87.06	91.74	31.09	76.07
	DRL	**83.63**	**84.49**	**89.76**	**93.97**	**35.51**	**79.67**



(a) DBLP (unsupervised) (b) DBLP (semi-supervised)

Figure 5: Performance w.r.t. training steps on the DBLP dataset. Our approach improves both the convergence and quality of the learned node representations.



(a) DBLP (unsupervised) (b) IMDB (unsupervised)

Figure 6: Performance v.s. the running time. The upper left indicates the optimal performance. Our approach is both effective and efficient.

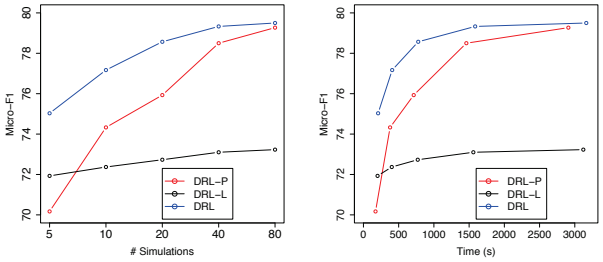
5.4 Efficiency Comparison

In the above sections, we have compared the effectiveness of different approaches. Next, we further evaluate their efficiency. To better illustrate the efficiency of our approach, we introduce another baseline approach Brute-Force, which learns the optimal action at each step through brute-force search based on the obtained rewards, and the maximal search depth is set as 3. We take the DBLP and the IMDB datasets as examples, and report both the performance and running time of each compared algorithm. The number of training threads is set as 16 for all approaches. For our approach, the planning module is trained on CPU with 16 threads, whereas the learning module is learned using a GPU.

Figure 6 presents the results. Comparing our approach (DRL) with the approaches based on weight learning (LINE-Weight) and brute-force search (Brute-Force), we see that our approach achieves close results but is much more efficient. On the other hand, our approach significantly outperforms other curriculum learning approaches (Rand and Greedy), and the efficiency is close. Overall, our approach can help learn more effective node representations with close efficiency.

5.5 Parameter Sensitivity

Performance w.r.t. the number of simulations. At each training step, our approach conducts several simulations to evaluate actions. Next, we study the performance of our approach under different numbers of simulations. We take the IMDB dataset in the unsupervised setting as an example. We report the Micro-F1 with

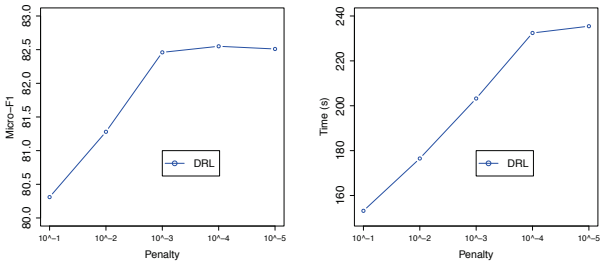


(a) Micro-F1 w.r.t. #simulations (b) Curves of Micro-F1 and time

Figure 7: Performance w.r.t. #simulations on the IMDB dataset. Our approach requires limited simulations to achieve compelling results (left). Leveraging both modules improves the effectiveness and efficiency (right).

respect to #simulations in Figure 7(a), and show the curves of Micro-F1 and running time with different #simulations in Figure 7(b) (5, 10, 20, 40, 80 simulations from left to right).

From Figure 7(a), we see that with both planning and learning strategies, our approach (DPL) requires very limited simulations to achieve compelling results, which shows that integrating them indeed improves the effectiveness. From Figure 7(b), we observe that compared with either module (DRL-L and DRL-P), our approach (DRL) achieves close results with lower time costs, demonstrating that both modules can work together to improve the efficiency.



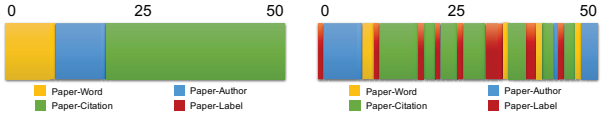
(a) Micro-F1 (b) Time

Figure 8: Performance w.r.t. penalty. We can flexibly balance the effectiveness with efficiency by adjusting penalties.

Performance w.r.t. the penalty. To balance the effectiveness with efficiency, our approach introduces a penalty for each action when calculating the rewards. A small penalty emphasizes the effectiveness while a large penalty emphasizes the efficiency. In this part, we study the performance under different penalties. We take the DBLP dataset in the unsupervised setting as an example, and report both the Micro-F1 and curriculum length (i.e., the number of training steps in the curriculum) under different penalties.

Figure 8 presents the results. When the penalty is large, we observe that the performance is quite limited but the whole training process is very efficient. As we decrease the penalty, the results are significantly improved, but the training process also takes longer time. Overall, with proper penalties, we can effectively trade off between the effectiveness and efficiency.

5.6 Case Study



(a) DBLP (unsupervised) (b) DBLP (semi-supervised)

Figure 9: The learned curriculum on the DBLP dataset.

Finally, we present some intuitive examples to show that the curriculum learned by our approach is indeed reasonable. Taking the DBLP dataset as an example, we present the learned curriculum in Figure 9, in which we constrain the number of training steps as 50 and each color corresponds to a type of edges.

In the unsupervised setting, the edges between papers and title words are selected at the beginning steps. The reason may be that the paper titles provide the most general information, which can benefit the learning of other knowledge. In the semi-supervised setting, the learned curriculum is very interesting. At the first training step, the labeled data (i.e., edges between papers and labels) are selected, which is very intuitive. Overall, both the labeled data (red) and unlabeled data (other colors) are selected during training, showing that both information is useful for learning meaningful node representations, and our approach can effectively integrate both kinds of information.

6 RELATED WORK

Our work is related to node representation learning, which learns low-dimensional vector representations for nodes in networks. Most existing approaches including DeepWalk [27], LINE [38] and node2vec [11] aim to preserve the structure information of networks, such that nodes with similar neighbors tend to have similar representations. There are also some studies [5, 12, 16, 18, 28, 37] focusing on heterogeneous networks, which exploit multiple types of edges. However, all these approaches ignore the training order of different types of edges, while in this paper we aim at learning a meaningful training order of edges of different types to improve the representation learning performance.

Another category of related work is curriculum learning, which aims at learning a meaningful order of the training data. Bengio et al. [4] first proposed the concept of curriculum learning, and applied the idea to the tasks of shape recognition and language modeling. The strategy was then successfully applied to various research domains such as computer vision [10, 15, 19, 25] and natural language processing [3, 31, 41]. The basic idea of these approaches is to start training with easy examples to first learn some simple aspects of a given task, and then gradually increase the difficulty to learn more complex aspects. Our work shares similar intuition with these studies, but we focus on learning node representations for heterogeneous star networks, which remains unexplored yet.

Our work is also related to reinforcement learning and planning approaches, which aim to solve the sequential decision making problems by either interacting with environments or conducting simulations. These approaches have achieved impressive results in a variety of applications, including the GO game [9, 29], real-time video games [23, 26], image classification [22], dialogue generation [17] and optimization [24, 42]. All these approaches either

leverage the learning strategy or the planning strategy, whereas we integrate both of them in our approach. There are also some studies combining learning and planning strategies [30, 34], which leverage linear models to approximate Q values. Different from them, we utilize deep neural networks (LSTM) to calculate Q values, and we focus on different applications from theirs.

7 CONCLUSION

In this paper, we studied the problem of curriculum learning for node representation learning in heterogeneous star networks. We proposed a deep reinforcement learning based approach, which integrates the learning and planning strategies. Experimental results proved the effectiveness and efficiency of our approach. In the future, we plan to apply our framework to general heterogeneous networks, in which a curriculum is defined as a sequence of meta-paths [32] or hyper-edges [12].

ACKNOWLEDGMENTS

Research was sponsored in part by the U.S. Army Research Lab. under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA), National Science Foundation IIS-1320617 and IIS 16-18481, and grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov). The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies of the U.S. Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.
[2] R. Bellman. A markovian decision process. Technical report, DTIC Document, 1957.
[3] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.
[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.
[5] T. Chen and Y. Sun. Task-guided and path-augmented heterogeneous network embedding for author identification. *arXiv preprint arXiv:1612.02814*, 2016.
[6] J. L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.
[7] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.
[8] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.
[9] S. Gelly, Y. Wang, O. Teytaud, M. U. Patterns, and P. Tao. Modification of uct with patterns in monte-carlo go. 2006.
[10] K. J. Geras and C. Sutton. Scheduled denoising autoencoders. *arXiv preprint arXiv:1406.3269*, 2014.
[11] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.
[12] H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, and J. Han. Large-scale embedding learning in heterogeneous event data. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 907–912. IEEE, 2016.
[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[14] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.
[15] Y. J. Lee and K. Grauman. Learning the easy things first: Self-paced visual category discovery. In *CVPR*, pages 1721–1728. IEEE, 2011.
[16] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu. Attributed network embedding for learning in a dynamic environment. *arXiv preprint arXiv:1706.01860*, 2017.
[17] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.
[18] J. Li, A. Ritter, and D. Jurafsky. Learning multi-faceted representations of individuals from heterogeneous evidence using neural networks. *arXiv preprint arXiv:1510.05198*, 2015.
[19] J. Louradour and C. Kermorvant. Curriculum learning for handwritten text line recognition. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 56–60. IEEE, 2014.
[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
[21] A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.
[22] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.
[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
[24] R. Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. 2014.
[25] A. Pentina, V. Sharmanska, and C. H. Lampert. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5492–5500, 2015.
[26] T. Pepels, M. H. Winands, and M. Lanctot. Real-time monte carlo tree search in ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):245–257, 2014.
[27] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
[28] M. Qu, J. Tang, J. Shang, X. Ren, M. Zhang, and J. Han. An attention-based collaboration framework for multi-view network representation learning. *arXiv preprint arXiv:1709.06636*, 2017.
[29] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*.
[30] D. Silver, R. S. Sutton, and M. Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pages 968–975. ACM, 2008.
[31] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. Baby steps: How "less is more" in unsupervised dependency parsing. *NIPS: Grammar Induction, Representation of Language and Language Learning*, pages 1–10, 2009.
[32] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.
[33] Y. Sun, Y. Yu, and J. Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 797–806. ACM, 2009.
[34] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.
[35] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1.
[36] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297. International World Wide Web Conferences, 2016.
[37] J. Tang, M. Qu, and Q. Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174. ACM, 2015.
[38] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.
[39] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM, 2008.
[40] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.
[41] Y. Tsvetkov, M. Faruqui, W. Ling, and C. Dyer. Learning the curriculum with bayesian optimization for task-specific word representation learning. *arXiv preprint arXiv:1605.03852*, 2016.
[42] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.