The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Distance-Sware DAG Embedding for Proximity Search on Heterogeneous Graphs

Zemin Liu,[1] **Vincent W. Zheng,**[2] **Zhou Zhao,**[1] **Fanwei Zhu,**[3*]
Kevin Chen-Chuan Chang,[4] **Minghui Wu,**[3] **Jing Ying**[1]
[1] Zhejiang University, China; [2] Advanced Digital Sciences Center, Singapore;
[3] Zhejiang University City College, China; [4] University of Illinois at Urbana-Champaign, USA
{liuzemin,zhaozhou}@zju.edu.cn, wenchenzheng@gmail.com, kcchang@illinois.edu, {zhufw,mhwu,yingj}@zucc.edu.cn

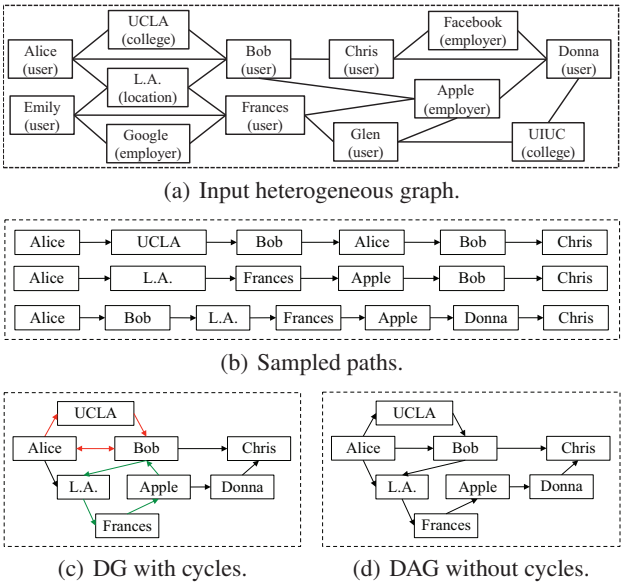
Abstract

Proximity search on heterogeneous graphs aims to measure the proximity between two nodes on a graph w.r.t. some semantic relation for ranking. Pioneer work often tries to measure such proximity by paths connecting the two nodes. However, paths as linear sequences have limited expressiveness for the complex network connections. In this paper, we explore a more expressive DAG (directed acyclic graph) data structure for modeling the connections between two nodes. Particularly, we are interested in learning a representation for the DAGs to encode the proximity between two nodes. We face two challenges to use DAGs, including how to efficiently generate DAGs and how to effectively learn DAG embedding for proximity search. We find distance-awareness as important for proximity search and the key to solve the above challenges. Thus we develop a novel Distance-aware DAG Embedding (D2AGE) model. We evaluate D2AGE on three benchmark data sets with six semantic relations, and we show that D2AGE outperforms the state-of-the-art baselines. We release the code on *https://github.com/shuaiOKshuai*.

Introduction

Semantic proximity search is an important task on heterogeneous networks (Sun et al. 2011; Fang et al. 2016). Given a node in the network as the query, it ranks other nodes according to some semantic relation. For example, in Fig. 1(a), we can take a user (*e.g.*, Alice) as a query, and ask "who are her *schoolmates*?" As the answer, we shall rank Bob higher than the other users since he and Alice both attend UCLA. Such semantic proximity search empowers many applications such as link prediction, circle suggestion and so on.

Pioneer work often tries to measure the semantic proximity by **paths** connecting two nodes (Jeh and Widom 2003; Backstrom and Leskovec 2011; Shi et al. 2017). Take measuring the *schoolmate* proximity between Alice and Chris in Fig. 1(a) as an example. A popular approach is to predefine the relation with some *meta-path* patterns (*e.g.*, "user-college-user"), and then count the number of meta-path instances between Alice and Chris (Sun et al. 2011). To avoid engineering such meta-path patterns, recent work adopts a representation learning approach to first sample paths between Alice and Chris, and then embed them as a low-dimensional vector for further computing the proximity (Liu et al. 2017). Despite the success of these prior methods, paths are simply linear sequences, thus having limited expressiveness for the complex network connections.



(a) Input heterogeneous graph.

(b) Sampled paths.

(c) DG with cycles.

(d) DAG without cycles.

Figure 1: Proximity embedding with different structures.

In this paper, we exploit **DAGs** (directed acyclic graphs) to model the connecting structure between two nodes, and aim to embed the DAGs as a low-dimensional representation for enabling proximity estimation. Next we motivate why we choose DAGs. Let us start with discussing the implications of using paths in Fig. 1(b). On the one hand, the paths are *directional*. This coincides with the intuition of modeling proximity from one node to another. Thus we are looking for a directional structure to model the proximity. On the other hand, the paths are limited in expressiveness. Modeling each path independently and aggregating them later does not fundamentally address this limitation. Thus we are also looking for a richer structure than paths. In all, we arrive at the choice of using DGs (directed graphs) to model the con-

*Corresponding author : Fanwei Zhu
Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

nections from one node to another. However, DGs can have cycles, which are not ideal for the proximity search task. For example, consider an example DG between Alice and Chris in Fig. 1(c). In practice, Alice can get to know Chris through she knowing a schoolmate Bob and Bob knowing Chris personally. Having an extra cycle among Alice, Bob and UCLA can make the distance between Alice and Chris becomes longer, thus weaker to explain their relation. Besides, cycles are not preferred for inference by probabilistic graphical models (Koller and Friedman 2009), thus not suitable for our representation learning as well.

It is not trivial to consider DAGs for proximity search. **Firstly**, we have to consider the *distance awareness* in DAG embedding. There are a handful of RNNs (recurrent neural network) that try to model DAGs for different applications; *e.g.*, (Shuai et al. 2016; Bianchini et al. 2005) for scene labeling, (Baldi and Pollastri 2003) for protein structure prediction, and (Zhu, Sobhani, and Guo 2016) for semantic compositionality. These models are not designed for proximity search, thus all overlooking the distance awareness in DAG embedding. Specifically, for a DAG, when combining the outputs of the predecessor nodes as the input of the successor, different predecessors should have varying contributions for a target relation due to their varying distances from the start node. For example, consider a DAG in Fig. 1(d). Chris' predecessors are Bob and Donna. The distance from the start node Alice to Bob is much shorter than that to Donna. Thus the connection between Alice and Bob is stronger than that between Alice and Donna. When combining the embedding of Bob and Donna as the inputs for Chris, we have to discriminate them by their own distance from the start node. Note that such distance-awareness can happen at any node in the DAG with multiple predecessors. **Secondly**, we have to consider the efficiency in online generating DAGs. A straightforward approach to generate a DAG from a start node q to an end node v on a graph is to use BFS (Breadth First Search), starting from the start node q. However, running BFS on the whole graph is expensive, taking an $O(|V|+|E|)$ complexity with V and E as the node set and edge set respectively. It is costing in the online stage for new query nodes to compute their DAGs.

We propose a novel **D2AGE** (Distance-aware DAG Embedding) model to address the challenges for proximity embedding. To address the first challenge of distance-aware DAG embedding, our insight is to introduce a *recursive distance discount mechanism* when embedding the DAGs. That is, for each node v in a DAG, we assign different weights to its different predecessors w.r.t. their distances from the start node. Generally, the further v is from the start node, the weaker their connection is. Thus for each node, we design the weights, such that the longer a distance is, the smaller its weight is. To address the second challenge of DAG generation, our insight is two-fold. On the one hand, sampling paths is much more efficient, and can be done offline. On the other hand, they can be easily assembled into more complex structures such as DGs. As we can control the number of paths to assemble for two nodes and the length of these paths, we guarantee the induced DGs to have a constant graph size. Thus running BFS on such a small DGs to generate DAGs is easy. Based on the above insights, in D2AGE, we first sample paths as an approximation for the whole graph, then feed them to a distance-aware DAG generation mechanism to output DAGs and node distances. Secondly, we develop a D2AG-LSTM (Distance-aware DAG-LSTM) model to recursively apply the distance discounts from multiple predecessors in DAG embedding. Thirdly, given the training supervision, we devise an end-to-end solution.

We summarize our contributions as follows. Firstly, we exploit a rich structure of DAGs for proximity embedding, and take the task-specific distance awareness requirement into consideration. Secondly, we propose a novel D2AGE model, which uses a distance-aware DAG generation mechanism to efficiently generate DAGs with distances, and a recursive distance discount mechanism to effectively enable D2AG-LSTM. Finally, we conduct extensive experiments on six semantic relations from three benchmark datasets, and show our method outperforms the state-of-the-art baselines.

Related Work

Earlier work for proximity search, such as SimRank (Jeh and Widom 2002) and Personalized Pagerank (Jeh and Widom 2003), only considers homogeneous graph, and does not differentiate semantic relations. Some recent studies handle heterogeneity. Supervised Random Walk (SRW) (Backstrom and Leskovec 2011) assigns different edge weights to bias the random walk for different relations. Meta-path Proximity (MPP) (Sun et al. 2011) uses predefined path patterns to indicate a relation and counts the number of path instances to measure the proximity. Meta-graph Proximity (MGP) (Fang et al. 2016; Jiang et al. 2017; Zhao et al. 2017), as a richer structure, upgrades the path patterns to frequent subgraph patterns. Recent work uses graph embedding to do link prediction and proximity search; *e.g.*, Deepwork (Perozzi, Al-Rfou, and Skiena 2014), LINE (Tang et al. 2015), node2vec (Grover and Leskovec 2016), and more (Zheng et al. 2016; Nie, Zhu, and Li 2017; Nikolentzos, Meladianos, and Vazirgiannis 2017). They usually learn an embedding for each node based on the network structure (Cai, Zheng, and Chang 2017). Such node embedding can be aggregated for proximity score estimation. ProxEmbed (Liu et al. 2017) considers node embedding as an indirect way to learn the proximity embedding. Instead, it expresses the connections between two nodes as a set of sampled paths, and encodes them into a vector by LSTM (Hochreiter and Schmidhuber 1997). Compared with the above methods, our D2AGE is first of all a representation learning method, which is different from MPP and MGP. Besides, we exploit DAG, which has a richer structure than paths used in MPP and ProxEmbed. Although MGP also considers a richer structure of meta-graphs, they rely on subgraph pattern mining, which is generally NP-hard (Ullmann 1976). Thus MGP models have to deal with only small-size meta-graphs, which limit their performance. Though metapath2vec (Dong, Chawla, and Swami 2017) suggests to sample useful paths by meta-paths, these meta-paths need to be predefined, and the sampled structures are still paths.

Multiple types of data structures have been explored in the recent neural network models. Tree-structured RNN are

considered in natural language processing, such as DT-RNN (Socher et al. 2014), tree-LSTM (Tai, Socher, and Manning 2015) and RNNG (Dyer et al. 2016). There are a handful of RNNs designed for DAGs in various applications, such as (Shuai et al. 2016; Bianchini et al. 2005) in image modelling, (Baldi and Pollastri 2003) in protein structure prediction. Then, to model semantic compositionality in NLP, DAG-LSTM (Zhu, Sobhani, and Guo 2016) is introduced. Compared with the above methods, we have two major differences: **1)** we consider DAG embedding, in contrast with the tree embedding (tree is a special kind of DAG). **2)** we consider DAG embedding for a different application on semantic proximity search, which has to enforce the distance awareness in modeling. It is hard to apply the existing DAG-RNNs, which often customize themselves for certain applications and do not consider distance awareness, to our task.

Problem Formulation

As our task is proximity search on a heterogeneous graph, we first provide a definition for **heterogeneous graph**:

Definition 1 *$G = (V, E, S, \tau)$ is a heterogeneous graph, where V is the set of nodes, E is the set of edges between nodes, S is the set of node types, and $\tau : V \to S$ is the map from a node to a type.*

For example, in Fig.1(a), we have $S = \{$"user", "college", "location", "employer"$\}$, and τ(Alice) = "user", τ(UCLA) = "college", τ(L.A.) = "location", τ(Apple) = "employer".

As **input**, we are given a heterogeneous graph G and a set of training tuples $\mathcal{T} = \{(q_i, a_i, b_i)|i = 1, ..., m\}$ for a certain relation. Here, q_i is a query node, and it is closer to candidate node a_i than b_i. As **output**, we get a proximity embedding vector for each (q_i, a_i) and (q_i, b_i). Formally, we denote the proximity embedding between a query node q and a target node v as $\mathbf{f}(q, v) \in \mathbf{R}^d$, where d is the dimension of embedding. For symmetric relation we have $\mathbf{f}(q, v) = \mathbf{f}(v, q)$, while for asymmetric $\mathbf{f}(q, v) \neq \mathbf{f}(v, q)$. Then we estimate a proximity score for each $\mathbf{f}(q, v)$ as:

$$\pi(q, v) = \theta^T \mathbf{f}(q, v). \quad (1)$$

where $\theta \in \mathbf{R}^d$ is a parameter vector. Finally, for a training tuple (q_i, a_i, b_i), we can evaluate its ranking loss by their corresponding proximity score $\pi(q_i, a_i)$ and $\pi(q_i, b_i)$.

D2AGE as an End-to-End Solution

Our training objective is to learn both the parameter θ and a proximity embedding model to embed each (q_i, a_i) and (q_i, b_i) in $\mathcal{T}$ as $\mathbf{f}(q_i, a_i)$ and $\mathbf{f}(q_i, b_i)$, such that the ranking loss on $\pi(q_i, a_i)$ and $\pi(q_i, b_i)$ is minimized. Specifically, we define the ranking loss for each training tuple (q_i, a_i, b_i) as:

$$\ell(q_i, a_i, b_i) = -\log \sigma_\mu(\pi(q_i, a_i) - \pi(q_i, b_i)), \quad (2)$$

where $\sigma_\mu(x) = 1/(1+e^{-\mu x})$ is a sigmoid function, and μ is a trade-off parameter. Guided by this ranking loss, we provide supervision for the proximity embedding, thus having an *end-to-end* solution for the ultimate semantic proximity search task. The main technical challenges of this work are about how to learn a proximity embedding for the connecting structure of each (q, v) as $\mathbf{f}(q, v)$. As motivated earlier, we plan to represent the connecting structure of (q, v) with a set of DAGs, and then use a distance-aware DAG embedding model to map these DAGs together into $\mathbf{f}(q, v)$. To achieve this, we need to answer two questions: 1) how to generate DAG, in an efficient manner for a pair of nodes? 2) how to embed DAGs, in an effective manner with distance awareness? In the next two sections, we answer them one by one.

Distance-aware DAG Generation

In semantic proximity search, we often need to online estimate proximity scores for some query nodes with their candidates. Running BFS over the whole graph G to get DAGs for two nodes is computationally prohibitive. We have to come up with some efficient realization to pre-index the network structure and online quickly assemble our structure indices to generate the connections between two nodes. In this paper, we exploit the idea of sampling paths as the *structure index* and online assembling them into DAGs.

Path Sampling. To index the network structure of G, we use random walks to sample paths on G. For each node, we sample t paths, each of length ℓ. In total, we have $|V|t$ paths about the graph, and we denote them as $\mathcal{P}$. Given the training tuples $\mathcal{T} = \{(q_i, a_i, b_i)|i = 1, ..., m\}$, for each query node $q \in \{q_1, q_2, ..., q_m\}$ and a corresponding candidate node $v \in \{a_1, ..., a_m, b_1, ..., b_m\}$, we truncate the paths from $\mathcal{P}$ to extract all the sub-paths that start from q and end at v, or start from v and end at q. For symmetric relations, we denote both the sub-paths from q to v and the sub-paths from v to q as $\mathcal{P}(q, v)$. For asymmetric relations, we denote only the sub-paths from q to v as $\mathcal{P}(q, v)$.

Path Assembling. Given a pair (q, v) and its sampled paths $\mathcal{P}(q, v)$, we assemble the paths in $\mathcal{P}(q, v)$ to generate DAGs. We denote the generated DAGs from q to v as $\mathcal{D}(q, v)$. Note that a trivial assembling of paths result in DGs with possible cycles. For example, the DG in Fig. 1(c) is the assembling result of the paths in Fig. 1(b). Thus we have to come up with an approach to assemble paths, while avoiding cycles. Our intuition is to *order* the nodes in the paths with different integer IDs, such that only a node with smaller ID can have a directional link to another node with larger ID. As a result, the directed graphs are guaranteed to be cycle-free. Besides, since distance awareness is important for DAG embedding later, we also want to estimate the "distance" from the start node q to each other node in the DAG. Although it is possible to compute such distances on a DAG after constructing it, we choose to take a BFS-like graph traversal approach to construct a DAG and estimate the distances simultaneously.

Next we use a running example to explain in details how we construct a DAG and estimate the distances from a start node to a candidate node on a DAG. In Fig. 2(a), we consider constructing a DAG with start node a to end node e. Thanks to the path sampling earlier, we have extracted the sub-paths $\mathcal{P}(a, e) = \{p_1, p_2, p_3\}$. We will start with a, and try to add in its neighbors with directed edges, while avoiding cycles. When a's neighbors are added in, we will continue to iteratively add their neighbors as well to grow the DAG. To record the neighbors of each node, we compile an adjacency list for each node v as $v.adj$. For example, since we observe $a \to d$ and $a \to b$ *sequentially* in p_1 and p_2, we

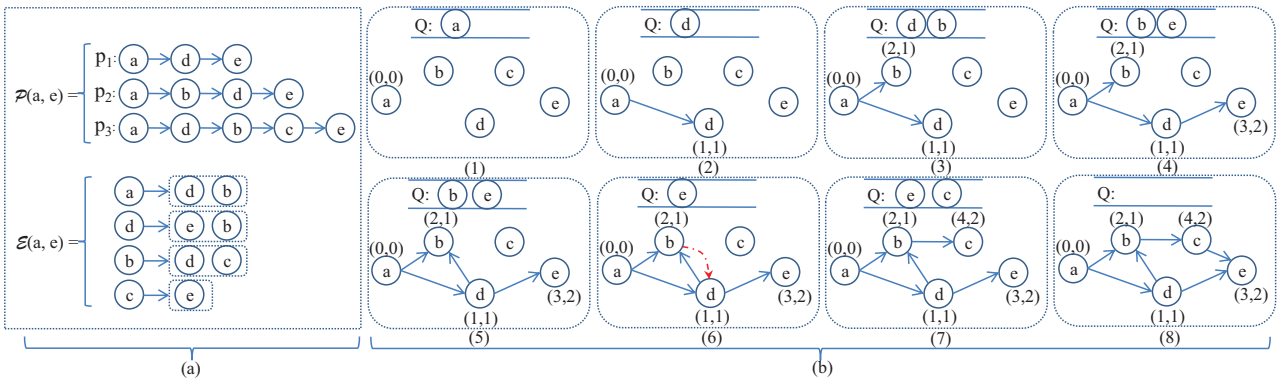


Figure 2: A running example for DAG generation.

add d and b with the same order to $a.adj$. We keep all the adjacency lists to $\mathcal{E}(a, e)$. To avoid cycles, we assign each node v with an integer ID $v.id \geq 0$. To record the distance from v to q, we also assign an integer distance score $v.dist \geq 0$. Initally, we set all the nodes v in $\mathcal{P}(a, e)$ with $v.id = -1$ and $v.dist = -1$. Finally, to iteratively add nodes and their directed neighbors for expanding the DAG, we maintain a queue Q, which is initialized as $\emptyset$.

In Fig. 2(b-1), we first $Enqueue(Q, a)$, and start from a with $a.id = 0$ and $a.dist = 0$, denoted as $(0, 0)$. In Fig. 2(b-2), we try to add a's neighbors one by one, by first $Enqueue(Q, d)$. Since d has never been visited in this DAG and so far only a is in this DAG, we assign a new ID to d, with $d.id = 1$. To avoid cycles, we require that an edge can be added, if it is from a smaller-ID node to a larger-ID node. As $d.id > a.id$, we add $a \rightarrow d$ to $D(a, e)$. Thus, we update $d.dist = 1$. According to $a.adj$, a's next neighbor is b, thus in Fig. 2(b-3) we $Enqueue(Q, b)$. Similarly, since b is never visited, we set $b.id = 2$, add $a \rightarrow b$ to $D(a, e)$ and thus update $b.dist = 1$. Now we have visited all a's neighbors, thus we start to explore its neighbors' neighbors. We $Dequeue(Q, d)$, and get $d.adj$. In Fig. 2(b-4), for d's first neighbor e, since is not visited yet, we $Enqueue(Q, e)$, then similarly set $e.id = 3$, add $d \rightarrow e$ to $D(a, e)$ and update $e.dist = 2$ (*i.e.*, two hops from a). In Fig. 2(b-5), we consider d's second neighbor b, which has been visited. Since $d.id < b.id$, we can add $d \rightarrow b$ without causing a cycle. Next, in Fig. 2(b-6), we $Dequeue(Q, b)$, and get $b.adj$. Since b's first neighbor is d but $d.id < b.id$, adding an edge $b \rightarrow d$ (shown as a red dot arrow) can cause a cycle. Thus we add nothing in this step. Then we move on to visit b's next neighbor c in Fig. 2(b-7) and finally completing the DAG in Fig. 2(b-8) after visiting all the possible nodes.

Remark: as we can see, the resulting directed graph in Fig. 2(b-8) is cycle-free, thanks to the constraint of having only directed edges from smaller-ID nodes to larger-ID nodes. Besides, we note that, if the paths in $\mathcal{P}(a, e)$ are ordered differently, our solution will order the nodes, as well as their adjacency lists, differently. Thus, we may have different DAGs. In practice it is not ideal to use one single DAG to represent the connections between two nodes, thus we recommend shuffling $\mathcal{P}(a, e)$ to generate multiple DAGs. In our experiments, we shuffle $|\mathcal{P}(a, e)| \times \eta$ times, and we find $0.05 \leq \eta \leq 0.2$ works empirically well.

Algorithm 1 Distance-aware DAG Generation

Require: graph G, start node q, end node v, paths $\mathcal{P}(q, v)$.
Ensure: a DAG $D(q, v)$, {u.dist} for all u's in $D(q, v)$ to q.

```
1:  D(q,v) = ∅; Q = ∅; latestID = 1;
2:  q.id = 0; q.dist = 0; Enqueue(Q, q);
3:  Shuffle(P(q, v));
4:  for all path p ∈ P(q, v) do
5:      for all directed edge a → b in p do
6:          a.id = −1, b.id = −1, a.dist = −1, b.dist = −1;
7:          Append(a.adj, b);
8:  while Q ≠ ∅ do
9:      curNode ← Dequeue(Q);
10:     for all node a ∈ curNode.adj do
11:         if a.id ≠ −1 then
12:             if a = v or curNode.id < a.id then
13:                 Append(D(q, v), curNode → a);
14:         else
15:             a.id = latestID + +;
16:             a.dist = curNode.dist + 1;
17:             Enqueue(Q, a);
18:             Append(D(q, v), curNode → a);
```

Our above solution in Alg.1. In line 3, we shuffle the paths in $\mathcal{P}(q, v)$. In lines 4–7, we initialize the ID, distance and adjacency list for each node in $\mathcal{P}(q, v)$. In lines 8-18, we maintain a queue to visit all the nodes for DAG expansion. As illustrated ealier, we visit the nodes sequentially, and add directed edges only from smaller-ID nodes to larger-ID ones. For each (q, v) pair, Alg.1 takes a BFS-like traversal approach, thus it only visits $\mathcal{P}(q, v)$ for a constant number of times. Then the complexity of Alg.1 is $O(|\mathcal{P}(q, v)|)$. In practice, we can control the size of $\mathcal{P}(q, v)$, to quickly generate DAGs for (q, v).

Distance-aware DAG Embedding

After DAG generation, we obtain a set of DAGs to express the connections between two nodes. Next, we introduce how to embed these DAGs into a proximity embed-

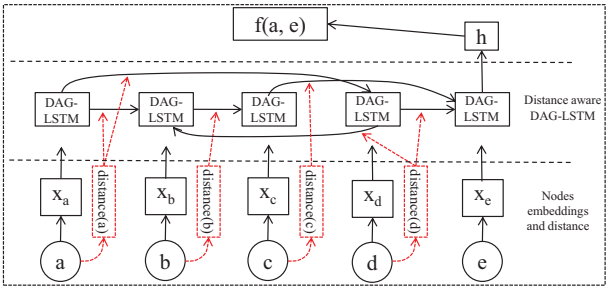


Figure 3: D2AG-LSTM.

ding vector. Becaue the DAGs are directed, we are inspired by the previous work (Sutskever, Vinyals, and Le 2014; Shuai et al. 2016; Zhu, Sobhani, and Guo 2016) to consider recursive and recurrent networks. Particularly, to avoid the gradient vanishing problem, we exploit an LSTM (long-short tem memory) architecture. We propose a novel D2AG-LSTM (Distance-aware DAG-LSTM) model to recursively apply distance discount when embedding each DAG.

D2AG-LSTM. We use an example DAG to illustrate how we model D2AG-LSTM. As shown in Fig. 3, D2AG-LSTM models two parts of information: 1) the topological structure of each DAG with node distances to the start node a; 2) the possible feature inputs $\mathbf{x}_j \in \mathbf{R}^n$ for each node j in the DAG. For the topological structure and the node distances, they are provided in Fig. 2(b-8). For the feature inputs $\mathbf{x}_j$ of node j, we concatenate the following information: 1) *node type* is a $|S|$-dimensional vector, where only the dimension corresponding to j's type is one and the others are zero; 2) *node degree* is a scalar, indicating j's degree in G; 3) *neighbor type distribution* is a $|S|$-dimensional vector, where each dimension is the number of j's neighbors with the corresponding type; 4) *neighbor type entropy* is a scalar, indicating the entropy of the type distribution for j's neighbors.

Similar to LSTM, each of our D2AG-LSTM unit also has an input gate $\mathbf{i}$, a forget gate $\mathbf{f}$, an output gate $\mathbf{o}$, a memory cell C, and a hidden state $\mathbf{h}$. Different from standard LSTM, D2AG-LSTM unit may have to update its gate vectors and memory cell states from multiple predecessor units, and also distributes its memory cell states and hidden states to multiple successor units. Besides, different from the other DAG-RNN models (Shuai et al. 2016; Zhu, Sobhani, and Guo 2016), D2AG-LSTM unit has to take node distance into account. Next, we detail the computation.

• *Hidden state for input*: node j takes the outputs of its possibly multiple predecessors as input. For different predecessors, we consider them as contributing differently, due to their distance differences. Generally, the larger the distance from q to j is, the weaker the connection from q to j is. Denote the predecessors for j as R_j. Thus, in Fig. 2(b-8), $R_b = \{a, d\}$, and $R_e = \{c, d\}$. To generate $\mathbf{h}_{R_j}$ as the input for unit j, we aggregate all j's predecessors' hidden states $\mathbf{h}_k$ for $k \in R_j$ by max pooling with a distance discount:

$$\mathbf{h}_{R_j} = maxPool(\{\mathbf{h}_k \cdot e^{-\alpha \times k.dist} : k \in R_j\}), \quad (3)$$

where parameter $\alpha > 0$ trades off the distance discounts.

• *Input gate*: Denote $W_i \in \mathbf{R}^{d \times n}$, $U_i \in \mathbf{R}^{d \times d}$ and $r_i \in \mathbf{R}^d$ as parameters. D2AG-LSTM computes the input gate as

$$\mathbf{i}_j = \sigma(W_i \mathbf{x}_j + U_i \mathbf{h}_{R_j} + \mathbf{r}_i). \quad (4)$$

• *Forget gate*: Denote $W_f \in \mathbf{R}^{d \times n}$, $U_f \in \mathbf{R}^{d \times d}$ and $r_f \in \mathbf{R}^d$ as parameters. D2AG-LSTM computes the forget gate as

$$\mathbf{f}_j = \sigma(W_f \mathbf{x}_j + U_f \mathbf{h}_{R_j} + \mathbf{r}_f). \quad (5)$$

• *Cell state*: The memory cell can take the memory from multiple predecessors as input. Thus we also use distance discount to differentiate each predecessor. We first compute the cell candidate value $\tilde{C}_j$,

$$\tilde{C}_j = \tanh(W_c \mathbf{x}_j + U_c \mathbf{h}_{R_j} + \mathbf{r}_c), \quad (6)$$

where $W_c \in \mathbf{R}^{d \times n}$, $U_c \in \mathbf{R}^{d \times d}$ and $r_c \in \mathbf{R}^d$ are parameters. Then, we aggregate j's predecessors' cell states C_{R_j} by max pooling with distance discount:

$$C_{R_j} = maxPool(\{C_k \cdot e^{-\alpha \times k.dist} : k \in R_j\}). \quad (7)$$

Now we get j's cell state C_j as

$$C_j = \mathbf{i}_j * \tilde{C}_j + \mathbf{f}_j * C_{R_j}. \quad (8)$$

• *Output gate* and **hidden state**: Denote $W_o \in \mathbf{R}^{d \times n}$, $U_o \in \mathbf{R}^{d \times d}$ and $r_o \in \mathbf{R}^d$ as parameters. D2AG-LSTM computes the output gate as

$$\mathbf{o}_j = \sigma(W_o \mathbf{x}_j + U_o \mathbf{h}_{R_j} + \mathbf{r}_o). \quad (9)$$

Finally, the output vector $\mathbf{h}_j$ is:

$$\mathbf{h}_j = \mathbf{o}_j * \tanh(C_j). \quad (10)$$

After getting the cell state C_j and the hidden state h_j, node j will distribute them to its successors. In this way, the distance discount is recursively enforced in every step of aggregation with multiple predecessors. Finally, we use the end node's hidden state as the embedding of a DAG; *i.e.*, we use $\mathbf{h}_e$ as the DAG embedding in Fig. 3. As there can be multiple DAGs between a node pair, we further aggregate all the DAGs' embedding. Denote $\Upsilon(q, v)$ as the set of DAGs for (q, v), where for *asymmetric* relation we have $\Upsilon(q, v) = \mathcal{D}(q, v)$, and for *symmetric* relation we have $\Upsilon(q, v) = \mathcal{D}(q, v) \cup \mathcal{D}(v, q)$. Once again, since the same end node may have different distances to the same start node in different DAGs, we apply distance discount to combine DAGs. Denote the end node distance in a DAG $D \in \Upsilon(q, v)$ as $v.dist^{(D)}$.

$$\mathbf{f}(q, v) = maxPool(\{\mathbf{h}_D \cdot e^{-\beta \times v.dist^{(D)}}, D \in \Upsilon(q, v)\}), \quad (11)$$

where parameter β trades off the DAGs distance discount.

Denote the parameters in D2AG-LSTM as $\Theta = \{\theta, W_i, U_i, r_i, W_f, U_f, r_f, W_c, U_c, r_c, W_o, U_o, r_o\}$. Then we can train D2AG-LSTM by minimizing:

$$L(\Theta) = \sum_i^m \ell(q_i, a_i, b_i) + \lambda \Omega(\Theta), \quad (12)$$

where λ is a trade-off parameter, and $\Omega(\cdot)$ is the regularization function(e.g., the sum of the l_2-norm). We summarize D2AG-LSTM in Alg.2. Line 1 generates training tuple

Algorithm 2 Distance-aware DAG Embedding

Require: graph G, training tuples $\mathcal{T} = \{(q_i, a_i, b_i)\}$, pre-computed DAGs Υ, hyper-parameters $\{d, \alpha, \beta, \mu\}$.
Ensure: model parameters Θ.

1: $\mathcal{B} \leftarrow$ GenerateBatches($\mathcal{T}$);
2: **for all** batch $b \in \mathcal{B}$ **do**
3: Initialize loss for batch b as $L_b = 0$;
4: **for all** each $(q_i, a_i, b_i) \in b$ **do**
5: $\mathbf{f}(q_i, a_i) \leftarrow$ D2AG-LSTM($\Upsilon, q_i, a_i, \alpha, \beta$);
6: $\mathbf{f}(q_i, b_i) \leftarrow$ D2AG-LSTM($\Upsilon, q_i, b_i, \alpha, \beta$);
7: $L_b = L_b + \ell(q_i, a_i, b_i)$, based on Eq. 2;
8: $L_b = L_b + \lambda\Omega(\Theta)$;
9: Update Θ based on L_b by gradient descent.

Table 1: Data sets with symmetric / asymmetric proximities.

	$\|V\|$	$\|E\|$	$\|C\|$	#(queries)
LinkedIn	65,925	220,812	4	172 (*school.*), 173 (*collea.*)
Facebook	5,025	100,356	10	340 (*family*), 904 (*class.*)
DBLP	165,728	928,513	5	2,439 (*advisor*), 1,204 (*advisee*)

batches for model training. Lines 5–6 apply D2AG-LSTM to learn the proximity embedding for each training pair (q_i, a_i) and (q_i, b_i). Then, line 7 estimates the ranking loss. Line 9 applies gradient descent w.r.t. the batch ranking loss. Meanwhile, the complexity of learning one DAG with n nodes is $O(nk)$, where k is the average node degree in the DAG. This is because for each node j in DAG, we consider embeddings from its k predecessors to form its input.

Experiments

Datasets. We test our model by six semantic relations from three real world datasets. The Linkedin dataset (Li, Wang, and Chang 2014) contains two symmetric relations: *schoolmate* and *colleague*; the Facebook dataset (Li, Wang, and Chang 2014) contains two symmetric relations: *classmate* and *family*; and the DBLP dataset (Wang et al. 2010) contains two asymmetric relations: *advisor* and *advisee*. The statistics of the three datasets are summarized in Table.1.

Set up. Since we are using the public benchmark data sets, we follow the same procedure with previous work (Fang et al. 2016) to generate our training and test data. For each semantic relation, a node q is used as a query node if there exists at least another node v such that q and v belong to the desired type of relation in our ground truth. We randomly sample 20% of queries as the training set and 80% as the test set. We repeat this procedure for five times and report the average performance. For each query node q_i, we construct training tuples $\{(q_i, a_i, b_i)\}$ from the ground truth, where q_i and a_i belong to the desired class of relation while q_i and b_i do not. For testing, we construct an ideal ranking for each test query q_j, where the ground truth nodes of q_j ranked higher than the non-ground-truth nodes or the nodes without labels. Next, we compare the ranking generated by our model against the ideal ranking for evaluation. NDCG (Järvelin and Kekäläinen 2002) and MAP are adopted to evaluate the top 10 nodes in each ranking.

Table 2: Comparison with the baselines with 100 labels.

	Methods	Symmetric				Asymmetric	
		Linkedin		Facebook		DBLP	
		school.	*collea.*	*class.*	*family*	*advisor*	*advisee*
NDCG@10	SRW	0.515	0.502	0.389	0.389	0.689	0.402
	DWR	0.518	0.506	0.689	0.575	-	-
	MPP	0.504	0.510	0.803	0.647	-	-
	MGP	0.568	0.546	0.843	0.732	-	-
	ProxEmbed	0.652	0.606	0.851	0.743	0.765	0.405
	D2AGE-α	0.669	0.626	0.824	0.747	0.760	0.405
	D2AGE-β	0.658	0.571	0.374	0.495	0.764	0.391
	D2AGE-$\alpha\beta$	0.648	0.568	0.362	0.526	0.748	0.396
	D2AGE	**0.678**	**0.639**	**0.856**	**0.765**	**0.779**	**0.417**
MAP@10	SRW	0.272	0.289	0.324	0.255	0.630	**0.294**
	DWR	0.391	0.369	0.575	0.455	-	-
	MPP	0.260	0.305	0.681	0.543	-	-
	MGP	0.305	0.333	0.729	0.651	-	-
	ProxEmbed	0.541	0.492	0.801	0.678	0.690	0.283
	D2AGE-α	0.556	0.512	0.760	0.683	0.664	0.273
	D2AGE-β	0.542	0.443	0.256	0.381	0.666	0.263
	D2AGE-$\alpha\beta$	0.536	0.435	0.246	0.454	0.647	0.268
	D2AGE	**0.587**	**0.528**	**0.808**	**0.713**	**0.703**	0.290

Heterogeneity. We consider the heterogeneity of the graph from two aspects. Firstly, we take the node features into consideration for each node j to form the feature vector $\mathbf{x}_j$. Secondly, The types of nodes in some graphs may be unbalanced, e.g., in Facebook, there are much more "users", but fewer "college". To avoid the issue of type imbalance, we follow (Liu et al. 2017) to take the nodes types into consideration when sampling paths before generating DAGs. Specifically, when sampling the next node for the current node v, instead of randomly choosing one node from v's neighbours, we first sample a node type from the node type(s) of v's neighbours, and then sample a node of the specific type.

Parameters and environment. In this work, we adopt the paths sampling method in (Liu et al. 2017). For fair comparison, we feed the same set of sampled paths to both our baselines (i.e., DWR, ProxEmbd) and our own model. In the experiments, we set $\lambda = 0.0001$, then we tune parameters d, α, β, μ for different datasets. We run our experiments on Linux machines with eight 2.27GHz Intel Xeon(R) CPUs and 32GB memory. We use java-1.8 and Theano (Team 2016) for development.

Baselines. We compare D2ADE with the following start-of-the-art semantic proximity search baselines.

• *SRW*: Supervised Random Walk (Backstrom and Leskovec 2011) learns different edge weights to bias the random walk routes, so that the ranking order can be consistent with the ground truth.

• *DWR*: DeepWalk Ranking first learns the embedding for each node using DeepWalk (Perozzi, Al-Rfou, and Skiena 2014). After that, the proximity embedding of a node pair is derived by applying Hadamard product on their node embedding. We then optimize the loss function in Eq.2.

• *MPP*: Meta-path Proximity (Sun et al. 2011) uses meta-paths between nodes to do the semantic proximity search. We generate the meta-paths following the method in (Fang et al. 2016).

• *MGP*: Meta-graph Proximity (Fang et al. 2016) uses meta-graphs between nodes for proximity search.

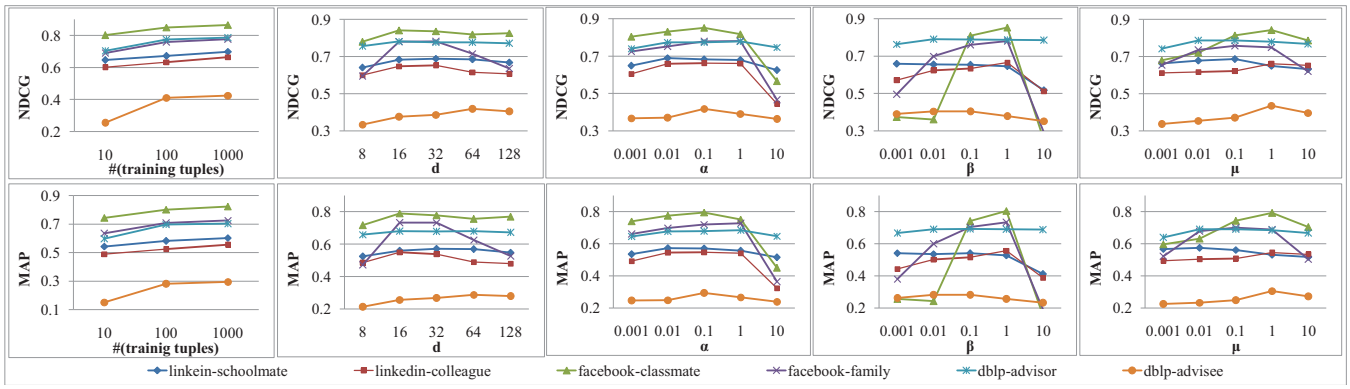


Figure 4: Impact of parameters: number of training tuples, dimension d, discount α, discount β, loss parameter μ

• *ProxEmbed*: ProxEmbed (Liu et al. 2017) uses paths between two nodes for proximity embedding.

• *D2AGE-α*: D2AGE without parameter α. We remove distance pooling within a DAG. We use simple max-pooling over the predecessors outputs.

• *D2AGE-β*: D2AGE without parameter β. We remove distance pooling among DAGs. We use simple max-pooling over multiple DAGs.

• *D2AGE-α-β*: D2AGE without parameter α and β. We completely remove the distance awareness.We use simple max-pooling when combining predecessors and DAGs. It is worth noting that, this baseline can be seen as basically applying DAG-LSTM (Zhu, Sobhani, and Guo 2016) with minor changes (*e.g.*, using max pooling instead of sum pooling) on our generated DAGs.

DWR, *MPP* and *MGP* are designed for symmetric proximity relation, and thus are not applicable for asymmetric proximity setting. We exclude them in asymmetric relation. For the baselines' parameters, we use the best parameter values reported by (Fang et al. 2016). Specifically, for *SRW*, we set its regularization parameter $\lambda = 10$, random walk teleportation parameter $\alpha = 0.2$, and loss parameter $b = 0.1$. We set the dimension of *DWR* as 128.

Comparison with baselines. In Table 2, we report the results of our method and the baselines, when using 100 randomly sampled training tuples for the training queries. As shown in the table, our approach D2AGE generally outperforms all the baselines. Firstly, D2AGE is better than *SRW* (except MAP for *advisee*), which uses the random walk to bias the route. Besides, D2AGE is better than *DWR*, which takes an indirect method, showing that the indirect methods are less effective for proximity search. Secondly, D2AGE is better than *MPP* and *MGP*, showing that feature learning is more effective than feature engineering in proximity search. Thirdly, D2AGE is better than *ProxEmbed*, showing that the complex structure DAG is more effective than the simple paths in proximity search. DAGs are more informative and expressive structures, and can describe more complex betweenness relations. Fourthly, D2AGE-α's performance is comparable with *ProxEmbed*'s, while D2AGE's performance is significantly better than *ProxEmbed*'s. This implies that the improvement comes from both rich structure and distance-aware modelling, and not simply each factor alone. DAG, as a richer structure than path, requires proper modelling of its structure (*i.e.*, distance awareness inside a DAG) to be fully useful. Once modelled properly, DAG can greatly improve the performance. D2AGE is also better than D2AGE-β, showing that the distance discount β used when combining the DAG embeddings to form the final embedding for the betweenness relation is pretty necessary. And D2AGE is better than D2AGE-α-β, showing the necessity of the distance discount inside and between DAGs.

Parameter Sensitivity. We test the robustness of our model by varying the number of training tuples. As shown in Fig.4, the performance is greatly improved as the number of training tuples is increased from 10 to 100 and then the performance improvements flattens when the number of training tuples is further increased. We also test the parameter sensitivity for our model by tuning the parameters d, α, β and μ using 100 training tuples. As we can see, for number of dimension d, when d is too small it gets worse results. For parameter α, $\alpha = 0.1$ is almost the best. When α is too small or too large, the result becomes worse. This means the recursive distance discount within DAG is necessary, and α should be in an appropriate range. For parameter β, it gets the best results when $\beta \in (0.1, 1.0)$. This shows the necessity of distance discount for each DAG by their length when combining them. Besides, for the loss function discount parameter μ, $\mu = 1.0$ is mostly the best. This suggests having a moderate discount on the ranking loss.

Conclusion

In this paper, we consider the problem of semantic proximity search on heterogeneous graph. We propose a novel distance-aware DAG embedding model D2AGE, to leverage more complex structure DAGs between two nodes to describe their relation. In D2AGE, we first use a distance-aware DAG generation approach to generate the the DAGs between two nodes; then we use a recursive distance-aware D2AG-LSTM model to embed these DAGs to a vector representing the relation. With such a proximity embedding vector, we train the parameters by the given supervision. We

test D2AGE on six proximity relations from three real world datasets, and it outperforms the baselines.

In the future, we will consider the proximity search for weighted directed graph, with deeper network models.

Acknowledgements

We thank the support from: National Natural Science Foundation of China (No. 61502418 and No. 61602405), Zhejiang Provincial Natural Science Foundation (No. LQ14F020002), Zhejiang Science and Technology Plan Project (China, No. 2015C01027), Research Grant for Human-centered Cyber-physical Systems Programme at Advanced Digital Sciences Center from Singapore A*STAR.

References

Backstrom, L., and Leskovec, J. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 635–644. ACM.

Baldi, P., and Pollastri, G. 2003. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *JMLR* 4:575–602.

Bianchini, M.; Maggini, M.; Sarti, L.; and Scarselli, F. 2005. Recursive neural networks for processing graphs with labelled edges: theory and applications. *Neural Networks* 18(8):1040–1050.

Cai, H.; Zheng, V. W.; and Chang, K. C.-C. 2017. A comprehensive survey of graph embedding: Problems, techniques and applications. *arXiv preprint arXiv:1709.07604*.

Dong, Y.; Chawla, N. V.; and Swami, A. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 135–144. ACM.

Dyer, C.; Kuncoro, A.; Ballesteros, M.; and Smith, N. A. 2016. Recurrent neural network grammars. In *NAACL HLT*, 199–209.

Fang, Y.; Lin, W.; Zheng, V. W.; Wu, M.; Chang, K. C.-C.; and Li, X.-L. 2016. Semantic proximity search on graphs with metagraph-based learning. In *ICDE*, 277–288. IEEE.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864. ACM.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780.

Järvelin, K., and Kekäläinen, J. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)* 20(4):422–446.

Jeh, G., and Widom, J. 2002. Simrank: a measure of structural-context similarity. In *KDD*, 538–543. ACM.

Jeh, G., and Widom, J. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, 271–279. ACM.

Jiang, H.; Song, Y.; Wang, C.; Zhang, M.; and Sun, Y. 2017. Semi-supervised learning over heterogeneous information networks by ensemble of meta-graph guided random walks. In *IJCAI*, 1944–1950.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

Li, R.; Wang, C.; and Chang, K. C.-C. 2014. User profiling in an ego network: co-profiling attributes and relationships. In *WWW*, 819–830. ACM.

Liu, Z.; Zheng, V. W.; Zhao, Z.; Zhu, F.; Chang, K. C.-C.; Wu, M.; and Ying, J. 2017. Semantic proximity search on heterogeneous graph by proximity embedding. In *AAAI*, 154–160.

Nie, F.; Zhu, W.; and Li, X. 2017. Unsupervised large graph embedding. In *AAAI*, 2422–2428.

Nikolentzos, G.; Meladianos, P.; and Vazirgiannis, M. 2017. Matching node embeddings for graph similarity. In *AAAI*, 2429–2435.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*, 701–710.

Shi, Y.; Chan, P.; Zhuang, H.; Gui, H.; and Han, J. 2017. Prep: Path-based relevance from a probabilistic perspective in heterogeneous information networks. In *KDD*, 425–434.

Shuai, B.; Zuo, Z.; Wang, B.; and Wang, G. 2016. Dag-recurrent neural networks for scene labeling. In *CVPR*, 3620–3629.

Socher, R.; Karpathy, A.; Le, Q. V.; Manning, C. D.; and Ng, A. Y. 2014. Grounded compositional semantics for finding and describing images with sentences. *TACL* 2:207–218.

Sun, Y.; Han, J.; Yan, X.; Yu, P. S.; and Wu, T. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4(11):992–1003.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*, 1556–1566.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*, 1067–1077.

Team, T. D. 2016. Theano: A Python framework for fast computation of mathematical expressions. *CoRR* abs/1605.02688.

Ullmann, J. R. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23(1):31–42.

Wang, C.; Han, J.; Jia, Y.; Tang, J.; Zhang, D.; Yu, Y.; and Guo, J. 2010. Mining advisor-advisee relationships from research publication networks. In *KDD*, 203–212. ACM.

Zhao, H.; Yao, Q.; Li, J.; Song, Y.; and Lee, D. L. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *KDD*, 635–644.

Zheng, V. W.; Cavallari, S.; Cai, H.; Chang, K. C.-C.; and Cambria, E. 2016. From node embedding to community embedding. *arXiv preprint arXiv:1610.09950*.

Zhu, X.-D.; Sobhani, P.; and Guo, H. 2016. Dag-structured long short-term memory for semantic compositionality. In *HLT-NAACL*, 917–926.