Neurally-Guided Semantic Navigation in Knowledge Graph

Liang He, Bin Shao, Yanghua Xiao, *Member, IEEE*, Yatao Li, Tie-Yan Liu, *Fellow, IEEE*, Enhong Chen, *Senior Member, IEEE*, and Huanhuan Xia

Abstract—In this big data era, knowledge becomes increasingly linked, along with the rapid growth in data volume. Connected knowledge is naturally represented and stored as knowledge graphs, which are of more and more importance for many frontier research areas such as machine intelligence. Effectively finding relations between entities in a large knowledge graph plays a key role in many knowledge graph applications, as the most valuable part of a knowledge graph is its rich connectedness, which captures rich information about the objects in the real world. However, due to the intrinsic complexity of real-world knowledge, finding semantically close relations by navigation in a large knowledge graph is very challenging. Canonical graph exploration methods inevitably result in combinatorial explosion especially when the paths connecting two entities are long: the search space is $O(d^l)$, where d is the average graph node degree and l is the length of the path. In this paper, we will systematically study the semantic navigation problem for large knowledge graphs. Inspired by AlphaGo, which was overwhelmingly successful in the game Go, we designed an efficient semantic navigation method based on a well-tailored Monte Carlo Tree Search algorithm with the unique characteristics of knowledge graphs considered. Extensive experiments on different real-life knowledge bases show that our method is not only effective but also very efficient.

Index Terms—Semantic navigation, knowledge graph, path finding, neural network.

1 Introduction

The value of big data largely lies in its rich connectedness. Connected knowledge is naturally represented as knowledge graphs to make knowledge processable by computers. Knowledge graphs equip machines with rich structured knowledge. Effectively making use of a massive amount of machine processable knowledge is an indispensable part to machine intelligence.

A knowledge graph consists of a set of entities and the relations between them. Finding the connections between data is a cornerstone of building machine intelligence via knowledge, as the most valuable part of a knowledge graph is its rich connectedness which captures rich information about the objects in the real world. In a lot of AI-powered applications, finding relations between entities of knowledge graph plays a key role. For example, the paths linking a pair of entities can be used to connect the entities and explain their relationships. Relation finding has a wide range of real-world applications such as suspicious relationship detection [4], relation prediction [17], knowledge completion [20], semantic distances estimation [9], and entity recommendation [24].

There are usually many paths connecting a pair of entities. Which one is of the greatest interest to us? This is not a trivial question and there are no easy answers. In the database field, the problem of finding the shortest paths is extensively studied [30]. However, in real-life applications, the most-valuable paths are those that are short and meaningful rather than just being the shortest. Short paths do reveal strong correlations, but the meaning or semantics of a path is of more importance. Consider the following two paths connecting entities x and y: $x-[brother]\rightarrow a-[married]\rightarrow b-[sister]\rightarrow y$ and $x-[is\_a]\rightarrow person-[instance]\rightarrow y$. The first one is obviously more meaningful than the second one. Moreover, the lengths of the paths representing the same relation vary a lot under different data schemas. For instance, the marriage relation between entities x and y can be represented as either $x-[married]\rightarrow y$ or $x-[marriage]\rightarrow m-[spouse]\rightarrow y$. Obviously, the length alone cannot well measure the importance of a path.

It makes more sense to find the paths that are short and meaningful. Here the distance constraint is relaxed to include longer but meaningful paths. As a result, the search space is greatly enlarged accordingly. Canonical graph exploration methods, including breadth-first search, depth-first search and A*, inevitably result in combinatorial explosion [12], [36].

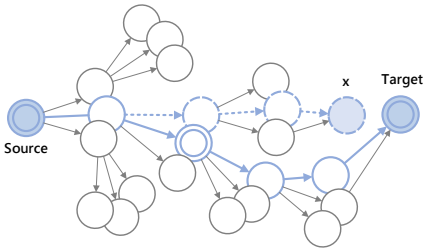

Fig. 1: **Semantic Navigation with the Guide of Knowledge Graph Embeddings.** For the *Source* node, navigation to *Target* by choosing the closest neighbor according to their Euclidean distance in the embedding space in each step leads to a dead end x. A "smarter" navigation is required to get around the dead end via the node marked by two concentric circles.

Motivated by the fact that humans usually leverage the semantics of concepts in memory associations for relation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2805363, IEEE Transactions on Big Data

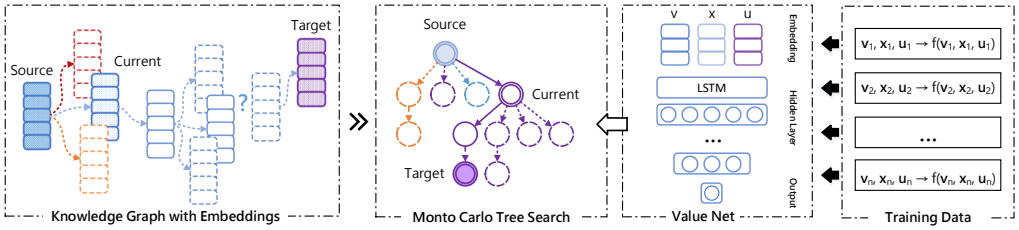

Fig. 2: Overview of Neurally-Guided Semantic Navigation

finding, we propose to leverage the entity semantics to aid relation finding. The semantics can not only provide explicit guidance for finding meaningful connections but also help on finding more explainable results. In this paper, we call the process navigating from one entity to another guided by entity semantics *semantic navigation*.

For knowledge graph, a widely adopted approach of encoding semantics is via knowledge graph embedding [7], [14], [32]. In spite of the great progress in this area, how to use the knowledge embeddings to find meaningful paths remains a challenging task. First of all, there are no agreed criteria for determining the meaningfulness of a path. It is hard to explicitly measure the meaningfulness using a well-defined metric. Furthermore, the navigation might lead to a dead end as shown in Figure 1 because knowledge graph embeddings themselves model semantics instead of topology connections, i.e. there is no guarantee that the navigation can reach the destination node by following the guide of graph embedding solely.

To tackle these challenges, inspired by AlphaGo [26] which was overwhelmingly successful in game Go, we designed an efficient semantic navigation method based on a neural value net and a well-tailored Monte Carlo tree search (MCTS) algorithm with the unique characteristics of knowledge graph considered. Despite the fact that playing Go game and navigating in a knowledge graph are very different, the core problem solved by AlphaGo is analogous to the task of graph navigation: from a board state AlphaGo needs to find where to put next stone, similarly *semantic navigation* needs to select a node through which to further traverse the graph. If we unroll the whole search space from the source node to the destination node, the search space is then very similar to the tree search space of AlphaGo. Intuitively, applying the design philosophy of AlphaGo can greatly boost the performance of semantic navigation. Considering the characteristics of navigation in knowledge graph, we tailored a Monte Carlo tree search algorithm [26] with a pre-trained value net to guide semantic navigation in knowledge graph. Figure 2 provides an overview of the proposed approach: Each trial of the tree search selects the candidate according to the score given by a pre-trained value net. A higher score indicates higher likelihood to reach the destination node through the current node.

The contributions of this paper are summarized as follows:

- We propose a novel neurally-guided semantic navigation method as an effective relation finding approach to searching for short and meaningful paths. A well-tailored Monte Carlo tree search algorithm with a value net is devised to guide the navigation.
- We conducted extensive experiments to evaluate the performance and costs of the proposed approach on real-life knowledge graph datasets. The experimental results show that the proposed method is not only effective but also very efficient for finding short and meaning paths in real-life knowledge graphs.

2 Related Work

Knowledge Graphs and Embedding Models. Knowledge graphs become more and more important components for many machine intelligence applications such as web search, question answering, e-Education, ChatBot. Recent years have witnessed the emergence of many knowledge graphs either built by human experts such as CYC [16] and Freebase [6] or built by the automatic extraction from text corpus such as WordNet [19], and DBpedia [1], [5].

Many knowledge graph embedding models are proposed for a variety of applications, such as knowledge-base completion, question answering, and knowledge reasoning. In these models, entities and relations are embedded into vector spaces according to their scoring functions. In this paper, we leverage knowledge graph embeddings as the guidance for solving the navigation problem in knowledge graph. There are many knowledge graph embedding models. Among them, translation-based models are the most prevalent ones including *TransE* [7], *TransH* [32], and *TransR* [18]. These models use relation r as translation from a head entity h to a tail entity t for calculating their embedding vectors of (h, r, t). They follow a similar assumption that $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. *TransE* represents a relation by a translation vector $\mathbf{r}$ so that the total error of $||\mathbf{h}+\mathbf{r}-\mathbf{t}||$ for each (h, r, t) is minimized. There are many other models proposed for knowledge graph embedding, such as *Distant Model* [8], *Bilinear Model* [14], [31], and *NTN* [29].

Empirical Analysis for Human Navigation. Many efforts are devoted to studying the patterns of human navigation behavior. Prior empirical analyses are driven by game log data, including Wikispeedia [25], [27], [34], and WordNet [2]. In general, humans can easily find intuitive but not necessarily the shortest paths [34]. Examples presented in [34] illustrated the striking difference between the paths found by human and the results of a shortest-path algorithm: the shortest paths are far less semantically meaningful than the paths found by the human. These human navigation paths are also used for calculating semantic similarities, such as constrained navigation paths [9], [28] and unconstrained navigation paths [11], [21], [22]. Inspired by these studies, we propose to tackle the knowledge graph navigation problem with the guidance of semantic embeddings.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2805363, IEEE Transactions on Big Data

Navigation in Information Networks. [10] introduced a random walk based navigation method for information networks. [3] also uses random walks for building graph embeddings and use the embeddings for finding shortest paths. [33] proposes heuristic-based agents based on node degrees and contents as well as machine learning agents for navigation. [15] uses ontologies as background knowledge and applies a navigation simulation method for information network navigation. [23] proposes to train an agent navigating through a web graph to find a web page in which a query appears.

In this paper, we are concerned about both short and meaningful paths instead of just shortest paths as we have argued above. Intuitively speaking, with the help of a neural value net and the guidance of semantic knowledge graph embeddings, our proposed method is essentially an intelligent knowledge-guided graph walk method.

3 SEMANTIC NAVIGATION

We leverage semantic knowledge graph embeddings to derive the guidance for navigation. *TransE* model [7] is chosen for this task due to its simplicity and good performance for predictive tasks. Based on the embeddings, we estimate the similarity distance between a pair of nodes in a knowledge graph. We use this distance metric as the guidance for choosing adjacent graph nodes in the navigation. In the following, we first introduce a straightforward navigation strategy as a baseline approach. We then present our neurally-guided navigation method, i.e. navigation with neural value net and Monte Carlo tree search.

3.1 SIMNAV: Navigation with Semantic Similarity

We start with a straightforward strategy that *chooses the semantically closest entity to the destination at each step*. With this strategy, we use L_2-norm for a pair of entities (v, u) as the distance measure:

$$d(v, u) = ||\mathbf{v} - \mathbf{u}||_2, \tag{1}$$

where $\mathbf{v}, \mathbf{u} \in \mathbb{R}^k$ are the k-dimensional semantic embeddings of v and u. As shown by Algorithm 1, the navigation starts from the source entity and in each step the procedure chooses one unvisited neighbor that is semantically closest to the destination entity until it reaches a dead end (line 4) or the destination entity (line 5).

Algorithm 1 Navigation with Semantic Similarity

1: **procedure** SIMNAV(v, u)
2: $P \leftarrow P \circ \{v\}$ ▷ P is the current path, initialized as $\emptyset$
3: $N \leftarrow$ unvisited neighbors of v
4: If N is empty, return failed
5: If $u \in N$, return $P \circ \{u\}$ ▷ $\circ$ stands for concatenation
6: SIMNAV(v', u) *s.t.* $v' = \arg\min_{x \in N} d(x, u)$
7: **end procedure**

There is one obvious drawback of this approach. It is easy to see that SIMNAV algorithm might lead to a dead end. If the navigation reaches a dead end, the navigation fails to find a path for the given pair of entities. We tackle the problem by leveraging a neural value net and a tailored Monte Carlo tree search algorithm for navigation.

3.2 NEURALNAV: Navigation with Value Net

A possible improvement of SIMNAV is to leverage a better navigation metric in each navigation step. We propose using a neural value net for this purpose. We refer to the strategy as NEURALNAV which follows the same procedure as the SIMNAV except for an additional value net, that is, we replace $d(x, u)$ with the output of the value net. Formally, the value net for navigation is $f : V \times V \times V \rightarrow \mathbb{R}$, where V stands for the entity set of the knowledge graph.

The value net is a neural network with a $(3 \times d)$-dimensional input layer and a single-neuron output layer, where d is the embedding dimension. The output neuron gives a score for guiding the navigation. Long short-term memory (LSTM) is widely reported for the good performance on modeling sequence data [13]. Thus we use it as the first layer to model the path of $[v, u, x]$, followed by several fully connected layers. The input layer consists of the embeddings of nodes (v, u, x), and its target value is calculated as follows:

$$f(v, x, u) = \lambda \cdot \frac{l(v, x, u)}{l(v, u)} + (1 - \lambda) \cdot d(x, u) \tag{2}$$

where x is a neighbor of v, $l(v, x, u)$ is the length of the shortest path from v to u through x, $l(v, u)$ is the length of the shortest path from v to u, $d(x, u)$ is the semantic distance between x and u, and $\lambda \in [0, 1]$ is a hyper parameter which can be chosen by performing grid search in the interval at a step size 0.1. More details about the value net are given in Section 4.1.

Compared with the semantic distance metric, this value benefits the navigation from three aspects:

- It considers the information of the source node v;
- A lower value of $\frac{l(v,x,u)}{l(v,u)}$ indicates that x is on some shorter paths connecting v and u;
- It can not only help pick semantically closer nodes at each step but also guide the navigation to get more reachable and shorter paths.

Each individual part of Equation 2 is not the best guidance for the navigation as will be illustrated in the experiments later. Thus we use a neural network to learn a better model.

3.3 MCTSNAV: Navigation with Tailored Monte Carlo Tree Search

Given a pre-trained value net for NEURALNAV, it is still possible to reach a dead end. We designed a Monte Carlo tree search based algorithm to tackle this problem. We refer to this approach as MCTSNAV. During navigation, the probability of selecting x is calculated using the following formula base on the value net:

$$p(x) \propto \frac{1}{f(v, x, u) \cdot (1 + n(x))}, \tag{3}$$

where $n(x)$ is the total visit count of the nodes in the subtree rooted at x.

Similar to a typical Monte Carlo tree search algorithm, each round of MCTSNAV consists of four steps: selection, expansion, simulation, and backpropagation, as shown in Algorithm 2. In the selection step (line 5), a node is selected according to Equation 3. Once a node is selected, say x, the unexpanded neighbor of x which is closest to the

Algorithm 2 Navigation via Monte Carlo Tree Search

```
1: procedure MCTSNAV(v, u)
2:     Initialize V ← {v}                          ▷ Candidates for selection
3:     while visited node number less than a threshold do
4:         If V is empty, return failed
5:         Choose v' in V with probability p(v') (Eq. 3)
6:         N ← the neighbors of v' that are not expanded
7:         Add x to V  s.t.  x = arg min_{z∈N} f(v', z, u)
8:         NEURALNAV(x, u)
9:         If u is reached, return path {v ⇝ v'} ∘ {x ⇝ u}
10:        Update n(z) for each z routing v to v'
11:        Remove v' from V if all neighbors of v' are expanded
12:    end while
13:    Return failed
14: end procedure
```

target node u is chosen for expansion (line 7) and simulation/unrolling (line 8). The simulation step is to perform NEURALNAV which unrolls to the target node from the expanded node. If the simulation fails, the backpropagation step is performed (line 10). In this step, the visit count of each node on the path from v to x increases by 1. Different from the typical Monte Carlo tree search, Algorithm 2 will stop once the target node is found.

3.4 Post-Processing

The path found using the above approach may be further improved. To derive an even better path, we devised an optimization method as a post-processing step. Let P be the nodes visited in the previous navigation procedure. Let $G[P]$ be the vertex-induced subgraph by the vertices on P. We run a post-processing step on $G[P]$ starting from v to u. The post-processing step enumerates all paths from v to u in $G[P]$ to get the best one. We then return this best path as the final result of the navigation.

This post-processing step can always find a path that is not worse than the path found in the previous step. As will be shown in our experiments, the post-processing can greatly reduce the path length. Because $G[P]$ is in general quite small, the costs of the post-processing is usually negligible. In practice, post-processing is a practical optimization method for deriving a better path.

4 EXPERIMENTAL EVALUATIONS

We have conducted extensive experiments to answer the following questions: 1) Can the neurally-guided semantic navigation produce short enough paths? 2) Are the paths found by semantic navigation meaningful? 3) Is semantic navigation efficient?

4.1 Datasets and Experiment Settings

We use five datasets for our experiments: FB15K, WN18, WikiGraph, Wikispeedia[1], and Clickstream [35]. These datasets are widely used in the studies of knowledge graph embedding [7]. FB15K is a relatively dense subgraph of Freebase [6], while WN18 is a subset of WordNet [19]. We use the extracted wiki graph from the dump file of the English Wikipedia [23] and the prepared TF-IDF values of the words as semantics representation for the Wikipedia articles. Wikispeedia consists of a small Wiki graph and human navigation paths on it. Each path is a clicked page sequence navigating from a start page to an explicitly given target by following only the hyperlinks provided on the current page. While the Wikispeedia dataset is used for constrained navigations, the behaviors of humans may be affected by the environments [22], we also study an unconstrained navigation dataset named Clickstream[2], which consists of (referer, resource) pairs and their counts extracted from the request logs of Wikipedia. For the Clickstream, we link it to the WikiGraph. The details of these datasets are listed in Table 1. We use *TransE* to train the graph embedding for FB15K and WN18 and set the dimension of the embedding as 50. All the experiments are conducted on a server with eight 3.00 GHz Intel(R) Core(TM) i7-5960X CPUs, each of which has two threads, and a Titan X GPU card.

TABLE 1: **Dataset Overview**

Datasets	#Nodes	#Edges	#Relations
FB15K	14,951	483,143	1,345
WN18	40,943	141,443	18
Wikispeedia	4,604	119,882	1
WikiGraph	5,946,517	99,812,932	1

4.1.1 Details of the Value Net

The value net for NEURALNAV and MCTSNAV is a neural network that consists of a recurrent layer (LSTM, $3 \times d$) followed by five fully connected layers, where d is the dimension of the embedding. The neuron counts of each layer are $(3 \times d)$, $3d$, $3d$, d, d, and 1 respectively, where $(3 \times d)$ is the input shape of the LSTM layer. For example, the neuron counts of each layer are (3×50), 150, 150, 50, 50, and 1 for FB15K. The activation functions are all set to Tanh. The target values of the output layer are calculated using Equation 2.

The value nets are trained using the datasets FB15K and WN18. The training data are sampled from the graph datasets. For each dataset, we sample n nodes from its entity set. For each node v, we use $\{x_1, x_2, \ldots, x_p\}$ to denote its neighbors. We then sample another p nodes $\{u_1, u_2, \ldots, u_p\}$ that are different to v or x_i. All the (v, x_i, u_j) combinations, where $1 \leq i, j \leq p$, are chosen as the training data and their target values $f(v, x_i, u_j)$ are calculated according to Equation 2. The target value for an unreachable (v, x, u) triple is set to a large positive number 100. For FB15K and WN18, we use about $265K$ and $3.5M$ of $\langle \mathbf{v}, \mathbf{x}, \mathbf{u}, f(v, x, u) \rangle$ samples for training respectively.

In the experiments, λ of Equation 2 varies from 0.0 to 1.0 with step size 0.1, an individual model is trained for each λ value. The value nets are trained using mini-batches with batch size 128. The loss function is *mean absolute error* and Adam is used as the optimizer with learning rate 0.001 with decay 0.0001. In the training, 10% data is used as the validation data set. The max training epoch is 100.

4.1.2 Compared Methods

For comparison, we also implemented a RANDOMNAV algorithm, which is similar to SIMNAV but randomly selects an unvisited neighbor in each step. Different regression based navigation methods are compared since they are also able

1. http://snap.stanford.edu/data/wikispeedia.html

2. https://figshare.com/articles/Wikipedia_Clickstream/1305770

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2805363, IEEE Transactions on Big Data

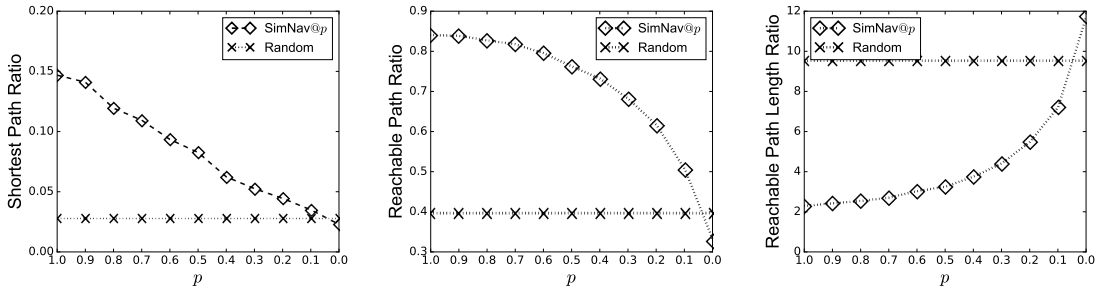

Fig. 3: **Probability of Finding Short Reachable Paths.** SIMNAV@p denotes a navigation procedure with probability p to choose the closest unvisited neighbor to the target node in each step. Obviously, the higher p is, more likely a short path can be found.

to predict the value for given (v, x, u). The embeddings of the same sampled (v, x, u) data set are used to train these regression models. In detail, we built LINEARNAV, LASSONAV, RIDGENAV, ELASTICNETNAV, and SGDREGNAV using linear, lasso, ridge, Elastic Net, and SGD regression algorithms. In addition, we take degree-based navigation into consideration, especially, we implemented DEGREENAV and DEGSIMNAV. DEGREENAV chooses an unvisited neighbor with largest degree in each step, while DEGSIMNAV selects the neighbor x with largest $d(x, u)^{-deg(x)}$ value, where $deg(x)$ is the out-degree of x.

4.1.3 Evaluation Metrics

In our experiments, we randomly select $10,000$ entity pairs that are reachable from each other. Here we define a few evaluation metrics which will be used in our experimental evaluation, specifically, *RPR*, *SPR*, and *RPLR*.

RPR (*reachable path ratio*) is the ratio of reachable paths our navigation can find to the total sampled pairs. *SPR* (*shortest path ratio*) is the ratio of shortest paths our navigation can find to the total sampled pairs. Besides *RPR* and *SPR*, we are also interested in the ratio between the lengths of the paths found by our navigation to the lengths of the shortest paths. For this purpose, we define *RPLR* (*reachable path length ratio*) as $\frac{\sum_i q_i^{-1}}{\sum_i p_i^{-1}}$, where $1 \leq i \leq 10000$. Specifically, for the i-th entity pair, let p_i be the length of the path discovered by our navigation and q_i be the length of its corresponding shortest path which can be calculated via a BFS (breadth-first search) procedure. When our navigation cannot find a path, we set p_i as ∞.

Instead of proposing an explicit metric to measure the meaningfulness of the found paths, we use three different ways to evaluate their meaningfulness. We first analyze how our human beings behave in the freely clicked navigation logs. Then, we study two cases of the navigated paths and compare them with the paths found by other navigation methods. Lastly, we statistically compare different navigation methods.

To evaluate the efficiency, we are interested in the number of entities that the navigation visited before it find a path. For this purpose, we define a metric called *VER* (*visited entity ratio*). For each pair of the sampled 10,000 entity pairs, we count the number of entities that are visited in our navigation procedure. The number is denoted by n_i. For comparison, we also calculate the number of the visited entities in a BFS procedure. The number is denoted by m_i. *VER* then can be calculated using equation $VER = \frac{\sum_i n_i}{\sum_i m_i}$.

4.2 Can Semantic Navigation Produce Short Enough Paths?

Whether our navigation can produce short paths depends on how likely the guidance for navigation can choose the closest nodes to the destination. Since all navigation algorithms share the same guidance, namely semantic distance, we only need to verify this in the most basic version. Let $p \in [0, 1]$ be the probability that the navigation algorithm selects the closest unvisited neighbor in each step. We plot the evaluation metrics introduced in section 4.1 against p for the FB15K data set. The results are shown in Figure 3. For comparison, we also give the results of random navigation as the baseline. The random navigation randomly selects an unvisited neighbor with a uniform distribution. It is clear that all three metrics have a strong correlation with p, that is, selecting closest neighbors with a larger probability certainly leads to a greater chance to find a short path or even the shortest path.

We run SIMNAV, NEURALNAV, MTCSNAV, and MTCSNAV with post-processing (denoted as MTCSNAV+) on the FB15K and WN18 datasets. We measure *SPR*, *RPR*, *RPLR*, and *VER* for each algorithm. The results for the first three metrics are shown in Figure 4.

In general, SIMNAV is much better than RANDOMNAV. For the WN18 data set, RANDOMNAV almost cannot find reachable paths. NEURALNAV behaves almost the same as SIMNAV when λ is 0. The performance of NEURALNAV becomes much better than SIMNAV while λ is between $[0.2, 0.6]$, but becomes close to or even worse than SIMNAV when λ approaches 1. Take $\lambda = 0.4$ as an example, the number of shortest paths found by NEURALNAV is twice the number of the ones found by SIMNAV for FB15K. For reachable paths, NEURALNAV outperforms SIMNAV by 10% in FB15K and 25% in WN18. Nevertheless, the results of NEURALNAV are much better than those of RANDOMNAV.

When λ is 1, the value net is trained solely by the topology structure. But interestingly, it is not the best for finding short paths. The conclusion is that the best navigation guidance is not derived from one single factor but a mixture of semantics and topology distances.

Compared with NEURALNAV, MTCSNAV gets almost the same results for *SPR* and *RPLR* (Figure 4). However,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2805363, IEEE Transactions on Big Data

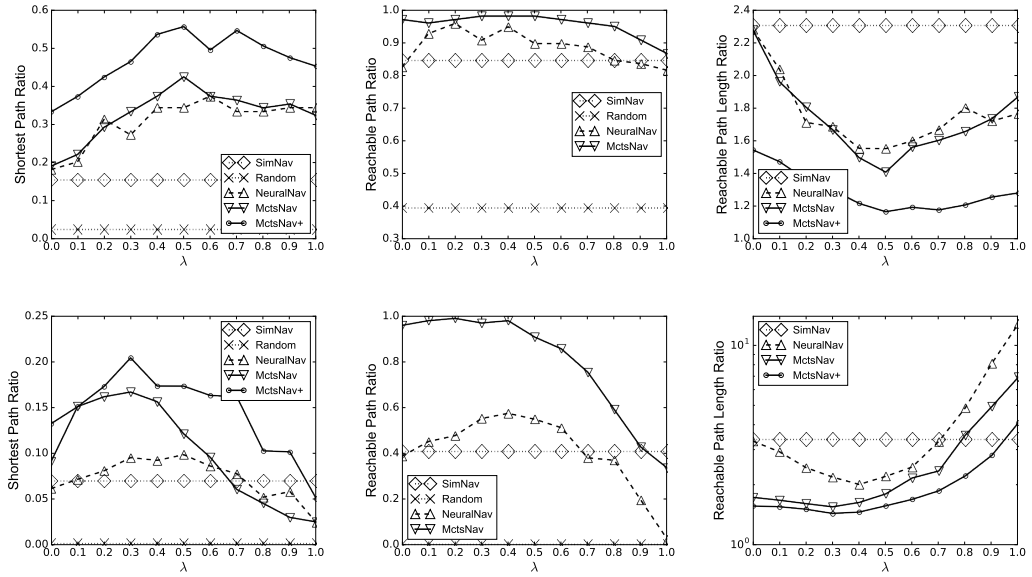

Fig. 4: **Navigation Performance on FB15K and WN18.** The first row shows the results for FB15K and the second row shows that for WN18. Larger values of *SPR* and *RPR* are better, indicating more shortest and reachable paths are found; while smaller values of *RPLR* are better, indicating the found paths are shorter. MCTSNAV+ denotes for MCTSNAV with post-processing. For RANDOM, the *RPLR* values are not plotted since they are very large, which are 11 and 145 respectively.

TABLE 2: **Comparison between Different Navigation Methods ($\lambda = 0.5$)**

Methods	$[\mathbf{v}, \mathbf{x}, \mathbf{u}]$						$\mathbf{u} - \mathbf{x}$					
	FB15K			WN18			FB15K			WN18		
	SPR	RPR	RPLR	SPR	RPR	RPLR	SPR	RPR	RPLR	SPR	RPR	RPLR
LINEARNAV	1.5%	46.4%	14.3	0.3%	3.2%	65.0	2.0%	9.7%	19.4	0.3%	0.5%	97.0
LASSONAV	2.7%	40.6%	10.4	0.2%	0.5%	117.8	2.7%	40.6%	10.4	0.2%	0.5%	117.8
RIDGENAV	1.5%	46.4%	14.3	0.3%	3.2%	65.0	2.0%	9.7%	19.4	0.3%	0.5%	97.0
ELASTICNETNAV	2.7%	40.6%	10.4	0.2%	0.5%	117.8	2.7%	40.6%	10.4	0.2%	0.5%	117.8
SGDREGNAV	2.7%	40.6%	10.4	0.2%	0.5%	117.8	2.7%	40.6%	10.4	0.2%	0.5%	117.8
RANDOMNAV	2.0%	39.4%	11.1	0.2%	0.5%	144.9	2.0%	39.4%	11.1	0.2%	0.5%	144.9
DEGREENAV	17.4%	90.4%	2.7	0.5%	13.1%	30.8	17.4%	90.4%	2.7	0.5%	13.1%	30.8
DEGSIMNAV	3.3%	24.3%	9.0	1.7%	9.6%	13.0	3.3%	24.3%	9.0	1.7%	9.6%	13.0
SIMNAV	15.5%	84.7%	2.3	7.0%	40.9%	3.4	15.5%	84.7%	2.3	7.0%	40.9%	3.4
NEURALNAV	38.3%	93.7%	1.5	8.5%	54.8%	2.7	38.3%	93.7%	1.5	8.5%	54.8%	2.7

TABLE 3: **Comparison on WikiGraph**

	SPR	RPR	RPLR
RANDOMNAV	0.0%	0.3%	7064.6
LASSONAV	0.0%	0.0%	-
DEGREENAV	0.0%	41.7%	259.6
SIMNAV	0.0%	26.8%	24.8
MCTSNAV+	1.8%	65.5%	2.5

MTCSNAV can discover more reachable paths as expected, especially when λ is small. Meanwhile, Figure 4 shows that MTCSNAV+ can greatly improve *SPR* and *RPLR* compared with MTCSNAV.

In summary, the pre-trained value net is a much better guidance for navigation than just using semantic similarity for most hyper parameter values. Based on it, MTCSNAV is able to find more reachable paths. For a small λ value for both datasets, it can find reachable paths for almost all node pairs. After applying post-processing for MTCSNAV, the *SPR* and *RPLR* metrics are greatly boosted further.

We also compared these navigation methods with regression based and degree based approaches as shown in Table 2. Two sets of inputs are considered, namely, $[\mathbf{v}, \mathbf{x}, \mathbf{u}]$ and $\mathbf{x} - \mathbf{u}$. As we can see, the regression based results are slightly better than RANDOMNAV, however, much worse than SIMNAV and NEURALNAV. The degree based methods are good at finding paths compared to the regression based ones, but result in much longer paths indicated by their *RPLR* values. Similar results can be observed in the experiments on WikiGraph, which is a larger dataset, and the results are presented in Table 3.

4.3 Are the Paths Found Meaningful?

To study human navigation with the semantics under unconstrained scenarios, we link the Clickstream dataset to the WikiGraph, where the Clickstream dataset is collected from real-life browsing log. We plot the semantic similarity of the (referer, resource) pairs against their click counts as shown in Figure 5. An obvious trend is that people click semantically closer links more frequently, which confirms that it is reasonable to choose a semantically closer neighbor in each step during navigation.

Because there are no metrics that can strictly measure the meaningfulness of a path, here we first use several real cases to study the explainability of the paths discovered by semantic navigation. Then, we compare the statistical

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2805363, IEEE Transactions on Big Data

metrics for the paths found by human and the ones found by semantic navigation.

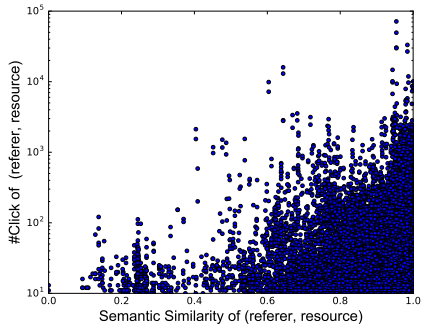

Fig. 5: **Click Count vs. Semantic Similarity in Clickstream.** Semantically closer pairs are clicked much more frequently.

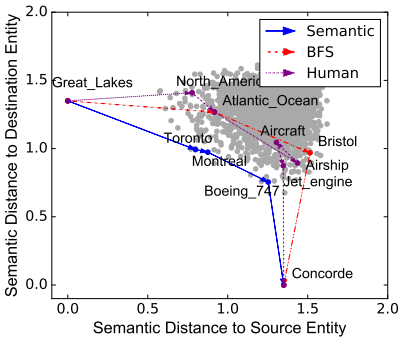

Fig. 6: **Semantic Distance to Source and Destination.** The gray points are entities that are not on the selected paths.

A few representative paths found by human and the semantic navigation algorithm are given in Figure 6. For each intermediate node on a path, we plot its semantic distance to the source node and the distance to the destination node. In general, the paths found by human and our navigation algorithm can both be easily explained. However, the paths found by the algorithm is generally shorter than the ones found by human.

Let us look at a concrete example on the Wikispeedia data set, from *ElephantBird* to *TheLordOfTheRings*, BFS finds a shortest path {*ElephantBird* → *Ostrich* → *Mythology* → *TheLordOfTheRings*}. Despite of the short length, there is a clear semantic gap between the two intermediate entities (i.e. *Ostrich* → *Mythology*). Humans can hardly associate these two entities in an intuitive meaningful way. In contrast, the path found by humans is longer but can be easily understood: {*ElephantBird* → *UnitedKingdom* → *BritishEmpire* → *NewZealand* → *TheLordOfTheRingsFilmTrilogy* → *TheLordOfTheRings*}. Compared with the path found by humans, the path found by our semantic navigation algorithm is not only shorter but also easily explainable: {*ElephantBird* → *UnitedKingdom* → *J.R.R.Tolkien* → *TheLordOfTheRings*}. It is interesting that the same entity (*UnitedKingdom*) is chosen by both semantic navigation and humans as the first entity; the second one (*J.R.R.Tolkien*) chosen by the algorithm is even semantically closer to the destination entity (*TheLordOfTheRings*) than the one chosen by humans.

To quantitatively study the characteristics of the paths found by semantic navigation, let us compare some statistical metrics for the paths found by human and the ones found by semantic navigation. Specifically, we studied three statistical metrics: *Shortest*, *AvgLen* and *AvgVisited*. Metric *Shortest* measures the shortest path percentage of the found paths. Metric *AvgLen* measures the average length of the found paths. Metric *AvgVisited* measures the average number of nodes visited for finding a path. To calculate these metrics, we conducted a set of experiments on the Wikispeedia data set which contains paths navigated by humans. We collected the source and destination node pairs for these paths. For these pairs, we run SIMNAV, DEGREENAV, DEGSIMNAV, BFS, and random search.

The results of the experiments are listed in Table 4. The *AvgLen* metric of the paths found by SIMNAV is close to the ones found by humans. Similar results can be observed for the *AvgVisited* metric. In contrast, the random search as well as degree based searches find much longer paths with much more nodes visited. As expected, BFS can 100% finds the shortest paths but with significantly more nodes visited. The results suggest that the semantics navigation algorithm is the closest navigation method to human navigation in terms of the statistical metrics considered here.

TABLE 4: **Statistical Comparison between Different Navigation Methods**

	Shortest	*AvgLen*	*AvgVisited*
Human(Avg)	22%	4.9	6.4
Human(Best)	34%	4.2	5.5
SIMNAV	42%	5.9	5.9
RANDOMNAV	6%	75.6	75.6
DEGREENAV	9%	156.0	156.0
DEGSIMNAV	22%	14.5	14.5
BFS	100%	2.8	293.7

4.4 Is Semantic Navigation Efficient?

The above experimental evaluation shows that the semantic navigation method can very likely find not only short but also meaningful paths. Now let us investigate how much cost is spent to find such paths.

We first examine the costs spent by the value nets, including those for training and inference. For the FB15K and WN18 datasets, the training procedures take about 200 and 3000 seconds respectively. The inference costs and the total costs are listed in Table 5. As we can see, the inference cost dominates the whole navigation cost for NEURALNAV since it just requires a few graph node visits besides the value prediction. While for MCTSNAV+, the inference cost becomes comparable to the remaining part. Each of them takes less than a second during navigation.

TABLE 5: **Time Cost of the Value Net during Navigation.** The values in each cell are 'value net cost/total cost (ratio)'.

[unit: sec]	FB15K	WN18
NEURALNAV	0.487/0.503 (96.8%)	0.064/0.065 (98.4%)
MCTSNAV+	0.551/1.311 (42.0%)	0.623/1.818 (34.2%)

To evaluate how efficient the navigation is, we recorded the *VER* values for all navigation methods we studied. The results are shown in Table 6. semantic navigation methods visit a very small portion of nodes compared to BFS, most of which are even less than 1%. The results suggest that these semantic navigations are very efficient for path finding compared with BFS.

TABLE 6: **Ratios of Visited Nodes to BFS for FB15K and WN18.** Compared with BFS, semantic navigation methods visit much less nodes for path finding.

Algorithm	FB15K	WN18
SIMNAV	0.10%	0.03%
NEURALNAV	0.09% − 0.25%	0.01% − 0.04%
MCTSNAV	0.30% − 1.69%	0.38% − 3.41%

Discussion. As many knowledge graph applications could tolerate some data update latency, the proposed approach could also work in the scenarios where model rebuilding on data snapshots are permitted. In practice, to minimize the model offline time, a two-model approach is usually adopted: when one model is used for online serving, the other one will be rebuilt.

5 CONCLUSION

In this paper, we proposed a neurally-guided semantic navigation method for navigating knowledge graphs. The basic idea is to leverage the semantic knowledge graph embeddings to guide the navigation. Keeping the unique characteristics of knowledge graph in mind and inspired by AlphaGo, we designed an efficient semantic navigation method based on a tailored Monte Carlo tree search algorithm with a neural value net as the navigation guidance. Experimental results show that the proposed method is not only effective but also efficient for finding short and meaningful paths in real-life knowledge graphs.

Acknowledgement. This work was partially supported by grants from the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61325010), the National Natural Science Foundation of China (Grant No. U1605251), and the National Key Foundation for Exploring Scientific Instrument of China (Grant No. 61727809).

REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.
[2] N. M. Beckage, M. Steyvers, and C. T. Butts. Route choice in individualssemantic network navigation. In *CogSci*, pages 108–113, 2011.
[3] Y. Berchenko and M. Teicher. Graph embedding through random walk for shortest paths problems. In *SAGA*, pages 127–140. Springer, 2009.
[4] A. Beutel, L. Akoglu, and C. Faloutsos. Graph-based user behavior modeling: From prediction to fraud detection. In *SIGKDD*, pages 2309–2310. ACM, 2015.
[5] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia-a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.
[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250. ACM, 2008.
[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.
[8] A. Bordes, J. Weston, R. Collobert, and Y. Bengio. Learning structured embeddings of knowledge bases. In *Conference on Artificial Intelligence*, number EPFL-CONF-192344, 2011.
[9] J. Borge-Holthoefer and A. Arenas. Navigating word association norms to extract semantic information. In *CogSci*, 2009.
[10] J. A. Capitán, J. Borge-Holthoefer, S. Gómez, J. Martinez-Romo, L. Araujo, J. A. Cuesta, and A. Arenas. Local-based semantic navigation on a networked representation of information. *PloS one*, 7(8):e43694, 2012.
[11] A. Dallmann, T. Niebler, F. Lemmerich, and A. Hotho. Extracting semantics from random walks on wikipedia: Comparing learning and counting methods. In *ICWSM*, 2016.
[12] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.
[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[14] R. Jenatton, N. L. Roux, A. Bordes, and G. R. Obozinski. A latent factor model for highly multi-relational data. In *NIPS*, pages 3167–3175, 2012.
[15] D. Lamprecht, M. Strohmaier, D. Helic, C. Nyulas, T. Tudorache, N. F. Noy, and M. A. Musen. Using ontologies to model human navigation behavior in information networks: A study based on wikipedia. *Semantic Web*, 6(4):403–422, 2015.
[16] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
[17] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu. Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379*, 2015.
[18] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187, 2015.
[19] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.
[20] A. Neelakantan, B. Roth, and A. McCallum. Compositional vector space models for knowledge base completion. 2015.
[21] T. Niebler, M. Becker, C. Pölitz, and A. Hotho. Learning semantic relatedness from human feedback using metric learning. *arXiv preprint arXiv:1705.07425*, 2017.
[22] T. Niebler, D. Schlör, M. Becker, and A. Hotho. Extracting semantics from unconstrained navigation on wikipedia. *KI-Künstliche Intelligenz*, 30(2):163–168, 2016.
[23] R. Nogueira and K. Cho. End-to-end goal-driven web navigation. In *Advances in Neural Information Processing Systems*, pages 1903–1911, 2016.
[24] A. Passant. Measuring semantic distance on linking data and using it for resources recommendations. In *AAAI Spring Symposium: linked data meets artificial intelligence*, volume 77, page 123, 2010.
[25] A. T. Scaria, R. M. Philip, R. West, and J. Leskovec. The last click: Why users give up information network navigation. In *WSDM*, pages 213–222. ACM, 2014.
[26] 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*, 529(7587):484–489, 2016.
[27] P. Singer. Understanding, leveraging and improving human navigation on the web. In *WWW*, pages 27–32. ACM, 2014.
[28] P. Singer, T. Niebler, M. Strohmaier, and A. Hotho. Computing semantic relatedness from human navigational paths: A case study on wikipedia. *IJSWIS*, 9(4):41–70, 2013.
[29] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.
[30] C. Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4):45, 2014.
[31] I. Sutskever, J. B. Tenenbaum, and R. R. Salakhutdinov. Modelling relational data using bayesian clustered tensor factorization. In *NIPS*, pages 1821–1828, 2009.
[32] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119. Citeseer, 2014.
[33] R. West and J. Leskovec. Automatic versus human navigation in information networks. In *ICWSM*, 2012.
[34] R. West, J. Pineau, and D. Precup. Wikispeedia: an online game for inferring semantic distances between concepts. In *IJCAI*, pages 1598–1603. Morgan Kaufmann Publishers Inc., 2009.
[35] E. Wulczyn and D. Taraborelli. Wikipedia clickstream. *Retrieved February*, 17, 2015.
[36] W. Zhang and R. E. Korf. Depth-first vs. best-first search: New results. In *AAAI*, pages 769–775, 1993.