Attributed Heterogeneous Information Network Embedding for Code Retrieval

Lingwei Chen[1], Shifu Hou[2], Yanfang Ye[*2], and Shouhuai Xu[3]
[1]College of Information Sciences and Technology, The Pennsylvania State University, PA, USA
[2]Department of Computer and Data Sciences, Case Western Reserve University, OH, USA
[3]Department of Computer Science, University of Colorado Colorado Springs, CO, USA

ABSTRACT

Modern software developers tend to engage in social coding platforms (e.g., Stack Overflow) to reuse code snippets for expediting the development process. To address code retrieval issue based on natural language queries over these platforms, in this paper, we introduce an attributed heterogeneous information network (AHIN) to model the corresponding data. Based on the constructed AHIN, we design an interaction schema *meta-tree* for the first time to search both local and global relatedness in AHIN, and elaborate a novel network embedding model *metatree2vec* to seamlessly fuse network structure and node attributes for embedding. This new embedding paradigm can better bridge the gap between the semantics of code snippets and queries. Comprehensive experiments on the data collection from Stack Overflow are conducted to validate the effectiveness of our code retrieval method.

CCS CONCEPTS

• **Computing methodologies** → *Learning latent representations*; • **Information systems** → *Web searching and information discovery.*

KEYWORDS

Social Coding, Code Retrieval, Heterogeneous Network Embedding

ACM Reference Format:
Lingwei Chen[1], Shifu Hou[2], Yanfang Ye[*2], and Shouhuai Xu[3]. 2018. Attributed Heterogeneous Information Network Embedding for Code Retrieval. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In order to expedite software development process, developers tend to engage in collaborative social coding platforms to reuse codes. Despite the apparent benefits, it still takes effort for them to locate the specific code snippets from searching over millions that reside on these platforms. As such, an effective code retrieval engine requires a higher-level semantic matching between code snippets and natural language queries (e.g., intent descriptions) [8]. Recent studies [2, 8, 10, 12, 24] have shown some promising research results in code retrieval based on natural language queries. Though leveraging embeddings [8] and collaborations among learning models [24] can improve the code retrieval performance, the existing approaches are still faced with limitations in code understanding: all the information is completely learned from individual code snippets without considering any potential network structure and semantics. Generally, social coding platforms are characterized by user communications through different threads, which provide a rich source of descriptive and comprehensive information to depict code snippets. In other words, such information can be exploited to better bridge the gap between the high-level intent of queries and low-level implementation of source code snippets [2].

To address the above challenge, an effective paradigm elaborated by this work is to leverage social coding properties for code retrieval based on natural language queries. We consider Stack Overflow as a case study for our approach, since it is the largest online programming discussion platform. To utilize these properties, in this paper, we introduce an attributed heterogeneous information network (AHIN) [5, 17] as an abstract representation to model Stack Overflow data. Based on the constructed AHIN, we design an interaction schema *meta-tree* for the first time to search both local and global relatedness in AHIN, and accordingly formulate a novel network embedding model *metatree2vec* for representation learning. Our metatree2vec makes the node embedding as two-step implementation: (1) we first instantiate a meta-tree to guide the generation of the semantic units among tightly inter-connected nodes, and utilize a graph auto-encoder model to preserve the local topology structure and semantics; (2) we then further devise another meta-tree to guide the generation of the node sequences, and adopt a seq2seq model to capture the long-range similarity over nodes, where the final representations of nodes (e.g., code snippets in our application) can thus be obtained. This allows a refined architecture to cope better with network structure and node attributes over AHIN in a local and global fashion for embedding, and thus build better lexical connections between the representations of code snippets and natural language queries.

2 PROBLEM STATEMENT

Given a set of natural language queries Q and code snippets C, the problem of code retrieval can be stated in the form of $f : Q \rightarrow C$ which learns a similarity measure model f to score the matching degrees between Q and C in order to retrieve the k ($k \geq 1$) highest scoring code snippets $c_k \in C$ to an input query $q \in Q$:

$$c_k = \begin{cases} \text{argmax}_{\hat{c} \in C} \ f(q, \hat{c}) & k = 1 \\ \text{argmax}_{\hat{c} \in C, f(q,\hat{c}) < f(q, c_{k-1})} \ f(q, \hat{c}) & k > 1 \end{cases} \quad (1)$$

* Corresponding author: yanfang.ye@case.edu.

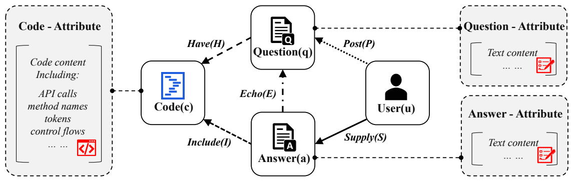


Figure 1: Network schema for AHIN in our application.

We assume that developers may provide natural language descriptions (NL_{des}) with or without keywords (NL_{word}) such as method names for code retrieval; thus, the space of possible queries Q is concatenated as $Q = [\mathrm{NL}_{des}; [\mathrm{NL}_{word}]]$ where NL_{word} is optional.

Since the query q and code snippet c is heterogeneous, we need to jointly embed them into a unified vector space so that we can reasonably measure the semantical similarity between them. Formally, the joint embeddings of q and c can be formulated as [8]:

$$q \xrightarrow{\mathcal{F}_q(q)} \mathbf{q} \to f(\mathbf{q}, \mathbf{c}) \leftarrow \mathbf{c} \xleftarrow{\mathcal{F}_c(c)} c \tag{2}$$

where $\mathcal{F}_q : q \to \mathbf{q} \in \mathbb{R}^d$ and $\mathcal{F}_c : c \to \mathbf{c} \in \mathbb{R}^d$ are embedding functions to map query q and code c into d-dimensional vector space respectively. In this paper, we employ cosine similarity for measuring $f(\mathbf{q}, \mathbf{c})$, where the higher the cosine similarity, the more matched the code snippet is to the query.

3 PROPOSED METHOD

3.1 Meta-tree Formulation

We first construct an AHIN $G = (V, E, \mathbf{X})$ to model Stack Overflow data, where V is node set, E is the edge set, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is node attribute matrix. To avoid introducing unexpected noises into network, we intuitively extract the most significant entities (i.e., *code snippet*, *question*, *answer*, *user*) and their relations (i.e., *question-have-code*, *answer-include-code*, *user-post-question*, *user-supply-answer*, *answer-echo-question*) from Stack Overflow for AHIN construction. Attribute vectors attached on nodes are learned using *doc2vec* [15] over their corresponding contents. Considering different types of nodes and edges in AHIN, we further map V and E to their types through functions ϕ: $V \to \mathcal{V}$ and ψ: $E \to \mathcal{E}$ respectively, and thus enable a network schema [19] $\mathcal{N}_G = (\mathcal{V}, \mathcal{E})$ to abstract the AHIN, which is shown in Figure 1.

Based on the constructed AHIN, meta-paths [20] a
used to characterize the relationships among nodes,
been shown to be useful in different applications. Ho
can only express simple pairwise relationships betwee
target objects [11], while fail to capture more complex r
For example, a meta-path is not able to depict the inter
among a question and all its answers with code snip
is an individual and indecomposable semantic unit. D
it into pairwise-node relations will enforce some sen
mation loss [6, 22]. Therefore, here we design a new
schema *meta-tree* to encapsulate both local topology an
relatedness over AHIN nodes, which can be defined a

Definition 3.1. **Meta-tree**. A meta-tree $\mathcal{T}$ is an intera
of tree structure defined on the network schema $\mathcal{N}_G$ =
mally, in meta-tree $\mathcal{T} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, \mathcal{V}_{id})$, $\mathcal{V}_{id} \in \mathcal{V}_\mathcal{T}$ is t
to represent the type of task identifier, and each bran

parent node and child node $(\mathcal{V}_p, \mathcal{V}_c) \in \mathcal{E}_\mathcal{T}$ represents the type of object connection, where $\mathcal{V}_\mathcal{T} \subseteq \mathcal{V}$, and $\mathcal{E}_\mathcal{T} \subseteq \mathcal{E}$.

Clearly, meta-tree is a prefix tree (or search tree), where each path from root to any internal or leaf node forms a meta-path; therefore, it can be comprehensively instantiated to depict the local topology structure, and higher-order relatedness over nodes in AHIN. That is to say, guided by different meta-tree schemas, we can dynamically search the nodes, and generate not only the semantic units with strong relationships, but also variable-length paths with long-range proximity. Although there are a variety of approaches for network embedding [4, 6, 7, 9, 18, 21–23], none of these approaches has a focus on simultaneously dealing with the local and global structural and attributed information over AHIN. In this respect, effective embedding method for AHIN is in need.

3.2 Metatree2vec

The AHIN embedding task is to learn a function $\mathcal{F} : V \to \mathbb{R}^d$ that maps each node $v \in V$ to a vector in a d-dimensional space ($d \ll |V|$) that is capable of encoding network structure and node attributes. To this end, we propose a two-step model *metatree2vec* to learn node representations in AHIN as follows.

Embedding over local semantic units. For our code retrieval in Stack Overflow, the task identifier can be specified by the code snippets. The important insight here is that code snippets are always semantically related to question and answer threads that provide them. The semantic unit is accordingly defined as the affiliation of each code snippet, with the additional information of the questions and answers. Since user information does not necessarily have a strong and direct relationship with code semantics, we exclude it from semantic units while leveraging it for next step. In this regard, a meta-tree can be instantiated as semantic-unit search schema to visualize relationships among the task identifier and participating nodes, and guide semantic unit generation for each task identifier. As such, based on the network schema shown in Figure 1, starting with code snippet, we traverse all the distinctive paths that interact with code snippet, question, and answer, and formulate a meta-tree as semantic unit search schema (shown in Figure 2) with three types: the first is code snippet being directly provided in questions and answers; the second is code snippet being associated with

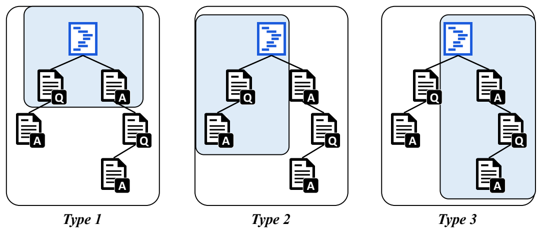


Figure 2: Meta-tree built for semantic unit search schema.

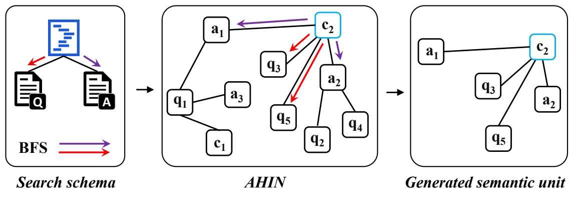


Figure 3: BFS for semantic unit generation by search schema.

of a node to nodes which are immediate neighbors of the source [7]. Specifically, let $\mathcal{V}_i$ be the current type node (i.e., root node) in search schema, and v_i be the current node in AHIN of $\mathcal{V}_i$ type. We first generate $\mathcal{V}_i$'s neighborhood $N_{\mathcal{V}_i}$; for each $\mathcal{V}_j \in N_{\mathcal{V}_i}$, we then traverse the AHIN to enumerate v_i's neighborhood N_{v_i} of $\mathcal{V}_j$ type, and merge all the nodes and the corresponding links into the node set and edge set of the semantic unit respectively; afterwards, update the current nodes to $\mathcal{V}_j \in N_{\mathcal{V}_i}$ and $v_j \in N_{v_i}$ ($\phi(v_j) = \mathcal{V}_j$) and repeat the same traversal until all type nodes in search schema have been visited or no nodes are available in AHIN. For instance, as shown in Figure 3, given a search schema and an AHIN, the generated semantic unit has node set $\{c_2\} \cup \{q_3, q_5\} \cup \{a_1, a_2\}$ and edge set $\{(c_2, q_3), (c_2, q_5)\} \cup \{(c_2, a_1), (c_2, a_2)\}$ where c, q, and a denote code, question, and answer respectively. As each semantic unit is an indirect graph, while intuitively, nodes in the same semantic unit should have significant impact on their neighborhood, a graph learning model is feasible for embedding over local semantic units.

Graph auto-encoder. We propose to exploit graph auto-encoder model [14] to implement this step, which encodes graph data to a compact representation, and then reconstructs the topological structure from such representation. Since graph convolutional network (GCN) has shown its effectiveness to learn graph structure and node attributes [13], we devise a two-layer GCN as an encoder to learn the compact representation:

$$\mathbf{Z} = \tilde{\mathbf{A}}\,\mathrm{ReLU}\,(\tilde{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}^0)\,\mathbf{W}^1, \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix for a semantic unit, $\tilde{\mathbf{A}}$ is a symmetric normalization of $\mathbf{A}$ with self-loop, $\mathbf{X}$ is the node attribute matrix, and $\mathbf{W}^l$ denotes weight matrix for the l-th layer. Afterwa
we train a decoder to reconstruct graph adjacency structure, wh
aim is to minimize a reconstruction loss $\mathcal{L}$:

$$\mathcal{L} = -\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{A}_{ij} \log(\sigma(\mathbf{z}_i^T \mathbf{z}_j)),$$

where $\sigma(x)$ is a sigmoid function, and batch gradient descent me
ods can be used to optimize parameters in $\mathcal{L}$. Generally, there
multiple meta-trees that can be instantiated as search schema (e
three in Figure 2). Let search schema number be K. We can apply
erage pooling on $\mathbf{Z}_i, i = 1, 2, ..., K$ to compute the node embeddi
over semantic units.

Embedding over long-range node sequences. Local seman
units only preserve similarities among task identifiers and their c
or two-hop neighborhoods, which is not sufficient due to the sp
sity of networks [22]. Higher-order relatedness over task identifi
also entails explicit or implicit similarities to further enhance AH
embedding [25]. In real-world scenarios, the typical meta-pa
based sequence formulation is apparently not ideal to thoroug
capture intrinsic node correlations as it overlooks the comp
interactions among different types of nodes. A remedy to this formulation is to use meta-tree to guide node sequence generation. Thus, we first enumerate all the meaningful meta-paths from network schema (Figure 4 (left)), and design a meta-tree to further integrate them for characterizing relatedness over code snippets (Figure 4 (right)). This allows us to flexibly interpolate among different meta-paths during sequence generation, which reflects a real and natural affinity for different code snippets in each sequence.

Sequence generation. To increase AHIN sampling rate and decrease the potential noises, we develop an attribute-aware random walk guided by meta-tree to explore the neighborhoods in a top-down fashion. The transition probability between two nodes is decided by not only the meta-path selection probability but also the attribute similarity. Specifically, we put a biased random walker to traverse the AHIN and then set the transition probability at step i as follows:

$$P(v_{i+1}|v_i, \mathcal{T}) = \begin{cases} \frac{p(\mathcal{V}_{t+1}|\mathcal{V}_t)\,\mathrm{sim}(\mathbf{x}_{\hat{v}}, \mathbf{x}_{v_{i+1}})}{\sum_{\tilde{v}\in N(v_i,\mathcal{V}_{t+1})} \mathrm{sim}(\mathbf{x}_{\hat{v}}, \mathbf{x}_{\tilde{v}})} & \\ & \text{if } (v_{i+1}, v_i) \in E, \phi(v_{i+1}) = \phi(\hat{v}) = \mathcal{V}_{t+1} \\ \frac{p(\mathcal{V}_{t+1}|\mathcal{V}_t)}{|N(v_i,\mathcal{V}_{t+1})|} & \\ & \text{if } (v_{i+1}, v_i) \in E, \phi(v_{i+1}) = \mathcal{V}_{t+1}, \hat{v} = \varnothing \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where $\mathrm{sim}(\mathbf{x}_{\hat{v}}, \mathbf{x}_{v_{i+1}})$ is the similarity between two nodes' attribute vectors, $\hat{v}$ denotes the latest node visited with the same type of v_{i+1}, $N(v_i, \mathcal{V}_{t+1})$ is $\mathcal{V}_{t+1}$ type of neighborhood of node v_i, and $p(\mathcal{V}_{t+1}|\mathcal{V}_t)$ is the probability of choosing $\mathcal{V}_{t+1}$ meta-path branch conditioned on the current $\mathcal{V}_t$ type guided by meta-tree $\mathcal{T}$. Let $l(\cdot)$ be the number of meta-paths from the current type to the leaves, $p(\mathcal{V}_{t+1}|\mathcal{V}_t)$ can be calculated as $l(\mathcal{V}_{t+1})/l(\mathcal{V}_t)$. After removing nodes whose types are not code, the remaining ones form a needed sequence, represented by embedding from semantic units $(\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_l)$. As such, embedding over node sequences can be viewed as a sequence modeling task.

Sequence modeling with GRU. As Gated Recurrent Unit (GRU) [3] has shown significant improvement in language modeling [1, 3],

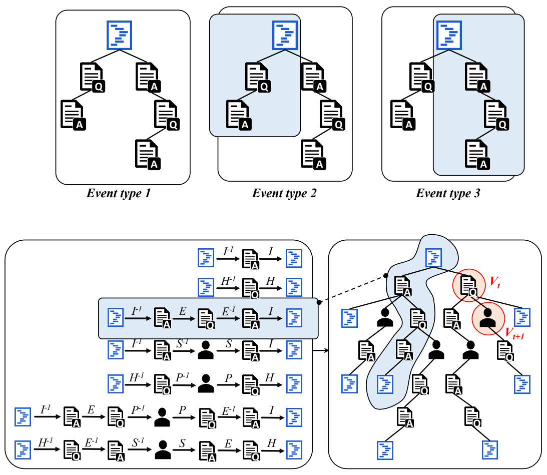


Figure 4: Meta-tree built for sequence generation.

As intermediate states, hidden vectors are extracted as the embedding for the corresponding nodes in the input sequence. Since each code snippet may appear in multiple sequences, by doing avg pooling, we can obtain final embedding for each code snippet.

3.3 Code Retrieval Model Training

Using metatree2vec, we can embed code snippets C into a vector space $\mathbf{C}$. As for queries Q, since each one is a natural language sentence, we embed them into vectors $\mathbf{Q}$ during model training through a GRU encoder layer. Afterwards, embeddings of each code snippet and query pair are fed to a multilayer perceptron (MLP) and the similarity between them is measured. The code retrieval model is trained by minimizing a ranking loss with a triple of $\langle q, c+, c\text{-}\rangle$ as training instance [24], where $c+$ answers query q while c- doesn't. Accordingly, the ranking loss is defined as [24]:

$$\mathcal{L}(\theta) = \sum_{\langle q,c+,c-\rangle \in D} \max(0, \epsilon - f(\mathbf{q}, \mathbf{c}+) + f(\mathbf{q}, \mathbf{c}\text{-})), \tag{7}$$

where θ denotes model parameters, D denotes training dataset, and ϵ is a constant margin. Minimizing the ranking loss $\mathcal{L}(\theta)$ enables the cosine similarity between q and $c+$ to be greater than that between q and c-, and thus the trained model can facilitate retrieving the relevant code snippets with respect to the given queries.

4 EXPERIMENTAL EVALUATIONS

4.1 Experimental Setup

Dataset. We test our model on a Stack Overflow data collection [25]: 429,523 question threads, 623,746 answer threads, 213,560 users, and 737,215 code snippets. 5,120 pairs of title and code snippet provided in the best answer thread are annotated as positive examples (q, $c+$), while 5,120 pairs of title and random code snippet (except for positive $c+$) are used as negative examples (q, c-). We further extract tokens from each code snippet as keywords for evaluation.

Performance measure. To quantitatively validate the effectiveness of different methods, we use FRank, SuccessRate@k (SR@k), and Mean Reciprocal Rank (MRR) as the performance measures [8, 16]. For a pair of (q, c), the FRank is the rank of c for query q in the evaluation list of K samples (including c itself) in the ascending order. Following the common settings in [2], we set K as 50 in our experiments. The SR@k is the percentage of queries whose paired code snippet exists in the top k ranked samples, which can be computed by:

$$\text{SR@k} = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \sigma(\text{FRank}_q \le k), \tag{8}$$

where $\sigma(\cdot) = 1$ if $\text{FRank}_q \le k$; otherwise, $\sigma(\cdot) = 0$. MRR is the average of the inverse FRanks for a set of query Q:

$$\text{MRR} = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{1}{\text{FRank}_q}. \tag{9}$$

The lower the FRank value, the higher SR@k and MRR, and the better the code retrieval performance.

Baselines. We validate the performance of our proposed method by comparisons with different groups of baselines. (1) Features: code content (original code attribute using doc2vec), semantic unit embedding (the first step of metatree2vec), and code sequence embedding (the second step of metatree2vec). (2) Embedding methods: DeepWalk [18], LINE [21], and metapath2vec [4] (vector dimension $d = 100$, walks per node $r = 10$, walk length $l = 50$). For DeepWalk and LINE, we ignore the heterogeneous property and directly feed the network for embedding. Since these three baselines are incapable of dealing with node attributes, we simply concatenate the node embedding with attribute vectors to represent a code snippet. (3) code retrieval models: DeepCS (feed code tokens to LSTM to learn the code representations) [8] and CoaCor (score the matching degrees between code snippets and their annotations, which are incorporated with the original scores between code snippets and queries to distinguish relevant code snippets from others) [24].

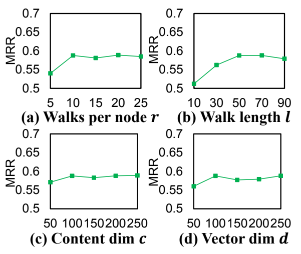
Table 1: Comparisons of different baselines

Baselines	MRR	SR@1	SR@5	SR@10
(a) Different types of features				
Code content	0.4556	0.3418	0.5703	0.6718
Semantic unit embedding	0.5600	0.4316	0.7089	0.8378
Code sequence embedding	0.5413	0.4043	0.6875	0.8359
(b) Different embedding methods				
DeepWalk	0.5128	0.3710	0.6738	0.7988
LINE	0.5168	0.3886	0.6503	0.8203
metapath2vec	0.5346	0.4023	0.6816	0.8203
(c) Different code retrieval models				
DeepCS	0.5145	0.3691	0.6621	0.8671
CoaCor	0.5395	0.3984	0.7148	0.8593
metatree2vec	0.5878	0.4609	0.7207	0.8867

4.2 Comparisons and Analysis

Comparisons of features. The results are illustrated in Table 1(a). We can observe that different features show different performances. (1) Compared to code content, network embedding is beneficial for code retrieval, which improves MRR from 0.4556 to greater than 0.55, and makes SR@1, SR@5 and SR@10 reach to a new level. (2) Semantic unit embedding outperforms code sequence embedding. The reason behind this could be that questions and answers that directly provide code, or connect to code through the same threads are likely to better describe code semantics than the ones related through long-range interactions; (3) metatree2vec achieves the best result by leveraging local and global structure over AHIN.

Comparisons of embedding methods. The comparisons are shown in Table 1(b). We can see that metatree2vec outperforms all baselines in terms of MRR and SR@k. DeepWalk, LINE, and metapath2vec merely embed the AHIN structure without considering any node attributes, while the brute-force concatenation of such node embedding and attribute vectors fails to capture the intrinsic correlations between them. By contrast, metatree2vec learns better node representations in AHIN. The success of metatree2vec lies in: (1) the proper consideration and accommodation of the heterogeneous property of AHIN, (2) the flexibility of meta-tree interaction schema to guide embedding, and (3) the advantage of graph auto-encoder



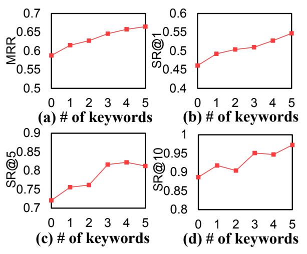


Figure 6: Keywords.

and GRU sequence modeling to incorporate both local and global structure and node attributes for representation learning.

Comparisons of code retrieval models. In this experiment, we slightly modify the original DeepCS to directly embed code snippets using LSTM. Table 1(c) shows the evaluation results, which demonstrate that our method produces more relevant results than DeepCS and CoaCor, and also higher MRR and SR@1. For DeepCS, it significantly relies on the pairwise correlations between code snippets and queries to bridge their semantic gaps without any help from other supportive information. For CoaCor, code annotations provide a rich source of supportive information, but it may also generate irrelevant or noisy descriptions to annotate the code snippets such that the code retrieval model may be mistrained.

Evaluation of parameters. In this experiment, we assess how different choices of parameters will affect the performance of our method. From Figure 5(a) and 5(b), we can observe that the balance between computational cost (r and l in x-axis) and efficacy (MRR in y-axis) can be achieved when $r \geq 10$ and $l \geq 50$. As shown in Figure 5(c), the performance tends to be stable once c reaches around 100; similarly, Figure 5(d) shows that the performance inclines to be stable when d increases to around 100. Overall, our method is not strictly sensitive to these parameters, and can reach high performance under a cost-effective parameter choice.

Evaluation of keywords. Though keyword-based code retrieval methods fail to match semantically relevant code snippets [2], we would still like to evaluate if the keywords can be integrated with natural language queries to further improve the performance of code retrieval. We extract tokens from each code snippet. Since the keywords have no sequential relationships, we embed them separately as vectors which are then fed to the fully connected layer to be fused with queries. Figure 6 illustrates the results, where the keywords increasingly added into queries indeed improve the code retrieval performance. Considering that developers may have knowledge about tokens in code snippets, code retrieval based on natural language queries and keywords may yield great value on social coding platforms.

5 CONCLUSION

In this paper, we leverage social coding properties for code retrieval based on natural language queries on social coding platforms. To model such data, we first introduce AHIN for abstraction and propose a new interaction schema meta-tree to capture both local and global semantics in AHIN. Guided by meta-tree, we elaborate a novel model metatree2vec to seamlessly embed AHIN structure and attributes for representation learning and build better lexical connections between the intent of queries and implementation of code snippets. metatree2vec is a generic framework which is able to learn desirable node representation in AHIN and thus can be further applied to various network mining tasks. The experimental results on Stack Overflow data demonstrate our method outperforms alternative approaches in code retrieval.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.
[2] Qingying Chen and Minghui Zhou. 2018. A neural framework for retrieval and summarization of source code. In *ASE*.
[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *arXiv:1406.1078*.
[4] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*.
[5] Yujie Fan, Yiming Zhang, Shifu Hou, Lingwei Chen, Yanfang Ye, Chuan Shi, Liang Zhao, and Shouhuai Xu. 2019. iDev: Enhancing Social Coding Security by Cross-platform User Identification Between GitHub and Stack Overflow. In *IJCAI*.
[6] Guoji Fu, Bo Yuan, Qiqi Duan, and Xin Yao. 2019. Representation Learning for Heterogeneous Information Networks via Embedding Events. *arXiv preprint arXiv:1901.10234* (2019).
[7] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.
[8] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *ICSE*. 933–944.
[9] Huan Gui, Jialu Liu, Fangbo Tao, Meng Jiang, Brandon Norick, Lance Kaplan, and Jiawei Han. 2017. Embedding learning with events in heterogeneous information networks. *TKDE* (2017).
[10] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *ICPC*. ACM.
[11] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. 2016. Meta structure: Computing relevance in large heterogeneous information networks. In *KDD*.
[12] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *ACL*, Vol. 1.
[13] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[14] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[15] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. 1188–1196.
[16] Xuan Li, Zerui Wang, Qianxiang Wang, Shoumeng Yan, Tao Xie, and Hong Mei. 2016. Relationship-aware code search for JavaScript frameworks. In *SIGSOFT*.
[17] Xiang Li, Yao Wu, Martin Ester, Ben Kao, Xin Wang, and Yudian Zheng. 2017. Semi-supervised clustering in attributed heterogeneous information networks. In *WWW*.
[18] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.
[19] Yizhou Sun and Jiawei Han. 2012. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery* 3, 2 (2012), 1–159.
[20] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.
[21] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*.
[22] Ke Tu, Peng Cui, Xiao Wang, Fei Wang, and Wenwu Zhu. 2018. Structural deep embedding for hyper-networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[23] Tao yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *CIKM '17 Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1797–1806.
[24] Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning. In *Proceedings of the 28th International Conference on World Wide Web*. 2203–2214.
[25] Yanfang Ye, Shifu Hou, Lingwei Chen, Xin Li, Liang Zhao, Shouhuai Xu, Jiabin Wang, and Qi Xiong. 2018. ICSD: An Automatic System for Insecure Code Snippet Detection in Stack Overflow over Heterogeneous Information Network. In *Proceedings of the 34rd Annual Computer Security Applications Conference (ACSAC)*. 542–552.