An Iterative Embedding-based Framework for Detecting Similar Mobile Applications

No Author Given

No Institute Given

Abstract. With the popularity of smart phones and mobile devices, large amounts of mobile applications (a.k.a."app") have been developed and published. Detecting similar apps from a large pool of apps is a fundamental and important task because it has many benefits for various applications (e.g., app recommendation). There exist several works that try to combine different metadata of apps in application markets to measure the similarity between apps in a principled way and obtain encouraging results. However, few methods pay attention to the roles (e.g., normal users, cybersecurity analysts) of this service. On the other hand, existing methods do not distinguish the characters of contents in these metadata (e.g., structured labels, unstructured texts). It is hard for them to capture fine-grained correlations among apps and obtain their accurate semantic representations. In this paper, we propose a novel framework by knowledge graph (KG) techniques and embedding models to fill these gaps. For the construction of KG, we further design a lightweight ontology tailored for the service of cybersecurity analysts. Benefited from a well-defined schema, more linkages can be shared among apps. To detect similar apps, we divide the relations in KG into structured and unstructured ones according to their related content. Then, TextRank algorithm is employed to extract important tokens from unstructured texts and transform them into structured triples. In this way, the representations of apps in our framework can be iteratively learned by KG embedding methods and network embedding models for improving the performance of similar apps detection. Preliminary results indicate the effectiveness of our framework comparing to several existing methods in terms of reciprocal ranking and minimum ranking.

Key words: Similar Apps Detection, KG Embedding, Network Embedding, Iterate Framework, Domain Ontology

1 Introduction

With the popularity of smart phones and mobile devices, the number of mobile applications (a.k.a. "app") has been growing rapidly, which provides great convenience to users for online shopping, education, entertainment, financial management etc. [1]. According to a recent report[1], as of August 2018, there were over 9.8 and 4.5 million apps available on Google Play and App Store, respectively, and global downloads of mobile apps have exceeded 194 billion. With large amounts of apps, if a specific app is given as a query, it is difficult to find all other apps that are similar to the query one.

Detecting semantically similar apps from a large pool of apps is a basic and important task because it has many benefits for different stakeholders in the mobile app ecosystem [2]. For example, it is helpful for app platforms to improve the performance

[1] https://www.appannie.com/cn/insights/market-data/the-state-of-mobile-2019/

of their app recommendation systems and enhance the user experience of app search engines. For app developers, detecting similar apps can be useful for various purposes such as identifying direct competing apps, assessing reusability (if open source) and so on. Meanwhile, lots of apps also become the hotbeds for cybercriminals such as thieving private data, propagating false news and pornography, online-scam. Therefore, it is essential for cybersecurity analysts to supervise apps and prevent potential cybercriminals related to them.

However, it is a nontrivial and difficult problem for detecting similar apps. One of the key challenges is how to explore and combine different modalities of data in app markets to measure the similarity between apps in a principled way. Previous studies provided solutions based on bag of words [3] or topic models [4, 5] to calculate the similarity of apps, which depended on description texts, titles and user reviews of apps. Recently, Chen et al. [2] and Lin et al. [6] proposed hybrid frameworks to achieve this service. The authors defined kernel functions and decision trees to integrate different metadata for improving the performances of similar apps detection.

Although existing methods have obtained some encouraging results, they still suffer from two limitations. Firstly, different objects (e.g., users, developers) expect different results of this service [7]. Therefore, it may not be suitable to directly utilize their algorithms to provide the service of similar apps detection for cybersecurity analysts, who pay more attention to the sensitive apps rather than the entertainment ones. Secondly, existing works focus on basic features of metadata, whereas they do not distinguish the characters of contents in these metadata (e.g., structured labels, unstructured texts). It is hard for them to capture fine-grained correlations among apps and obtain their accurate semantic representations.

To fill above gaps, in this paper, we present a novel framework for detecting similar apps using knowledge graph techniques and embedding models. We focus on one kind of apps, namely sensitive apps, that own more conditions or plausibility than normal apps that become the hotbeds for cybercriminals. We define a lightweight ontology including basic classes and properties from the view of cybersecurity analysts and construct the knowledge graph (KG) of sensitive apps. Benefited from a well-defined schema, more linkages can be shared among apps. To detect similar apps, we divide the relations in KG into structured and unstructured ones according to their related content. Then, TextRank algorithm [8] is employed to extract important tokens from unstructured texts and transform them into structured triples. In this way, the representations of apps in our framework can be iteratively learned by KG embedding methods [9] and network embedding models [10] for improving the performance of similar apps detection.

The main contributions of our work are summarized as follows.

1. We study the problem of detecting similar apps serviced for cybersecurity analysts. To the best of our knowledge, this is the first work that focuses on this problem;
2. We present a novel framework to tackle this problem, in which we construct a knowledge graph based on defined ontology such that more linkages can be shared among apps. Moreover, KG embedding methods and network embedding models are utilized to iteratively learn the representations of apps, which can further improve the performance of similar apps detection;
3. We construct a new dataset based on the constructed knowledge graph for evaluation. Compared with several existing methods, the preliminary result indicates that our approach for detecting similar apps of a new one can obtain better performances in terms of reciprocal ranking and minimum ranking.

2 Related work

2.1 Detecting Similar Mobile Applications

Detecting semantically similar apps from a large pool of apps is a fundamental and important problem, as it is beneficial for various applications, such as app classification and app recommendation, app search, etc.

Bhandari et al. [3] linked the title, description and user reviews of an app as one document, and then built the vector using the TD-IDF weighting scheme. They also used cosine similarity to calculate the pairwise similarity.

Yin et al. [4] treated the description of an app as a document and applied LDA to learn its latent topic distribution. In this way, each app was represented as a fixed-length vector. Then, the similarity between two apps was computed as the cosine similarity of their vectors.

Chen et al. [2] proposed a framework called SimApp that detected similar apps by constructing kernel functions based on multi-modal heterogeneous data of each app (e.g., description texts, images, user reviews) and learned optimal weights for the kernels.

Park et al. [5] exclusively leveraged text information such as reviews and descriptions (written by users and developers, respectively) and designed a topic model that could bridge the vocabulary gap between them to improve app retrieval.

Lin et al. [6] developed a hybrid framework that integrated a variety of app-related features and recommendation techniques, and then identified the most important indicators for detecting similar app. The authors employed a gradient tree boosting model as the core to integrate the scores by using user features and app metadata as additional features for the decision tree.

Although existing methods have obtained some encouraging results, it may not be suitable for these algorithms to provide the same service for cybersecurity analysts, who pay more attention to the sensitive apps rather than the entertainment ones. Because different objects expect different results of this service [7]. On the other hand, existing methods do not distinguish the characters of contents in these metadata (e.g., structured labels, unstructured texts). It is hard for them to capture fine-grained correlations among apps and obtain their accurate semantic representations. Relatively, we define a lightweight ontology in view of cybersecurity analysts, and utilize existing metadata of apps to construct a knowledge graph to achieve this goal. Moreover, we employ KG embedding methods and network embedding models to iteratively learn the representations of apps, which can further improve the performance of similar apps detection.

2.2 Knowledge Bases for Mobile Applications

Many interesting insights can be learned from data on application markets and aggregations of that data, which gain a remarkable attraction from academia and industry [11].

Drebin [12] provides a considerate number (5,560) of malware to the public with specific malicious behaviors inside, which were identified and classified by a machine learning method. These samples were categorized into 149 families in terms of contained malicious behaviors.

Commercial databases have mirror metadata from Google Play and other app markets and sell access to this information (e.g., appannie.com, appbrain.com and appszoom.com [13]). Most of these commercial databases contain comprehensive metadata of millions of apps but they lack links to other resources.

AndroZoo++ [14] is an ongoing effort to gather executable Android applications from as many sources as possible and make them available for analysts. In addition, the authors figured out 20 types of app metadata to share with the research community for relevant research works.

AndroVault [15] is a knowledge graph of information on over five million Android apps. It has been crawled from diverse sources, including Google Play and F-Droid since 2013. AndroVault computes several attributes for each app based on downloaded android application package, in which entities can be heuristically clustered and correlated by attributes.

Commercial databases and AndroZoo++ mainly focus on the scale of apps. The goal of their collecting apps is to sell their access and share them with the research community. Relatively, Drebin and AndroVault dedicate to provide abundance apps for malware detection. All of these knowledge bases pay little attention to similar apps detection serviced for cybersecurity analysts, but it is essential for them to supervise apps and prevent potential cybercriminals. To the best of our knowledge, our work is the first step towards employing a knowledge graph techniques and embedding models to tackle this problem.

3 Detecting Similar Mobile Applications

At the beginning of presenting our framework, we give a formalized definition that models the similarity of apps.

Definition 1 *(Mobile App Similarity Modeling) [2]. Given a collection of mobile apps* $\mathcal{A}$*, the objective of mobile app similarity modeling problem is to learn a function* $f : \mathcal{A} \times \mathcal{A} \rightarrow R^+$*, such that* $f(a_i, a_j)$ *can measure the semantic similarity for any two apps* $a_i, a_j \in \mathcal{A}$*.*

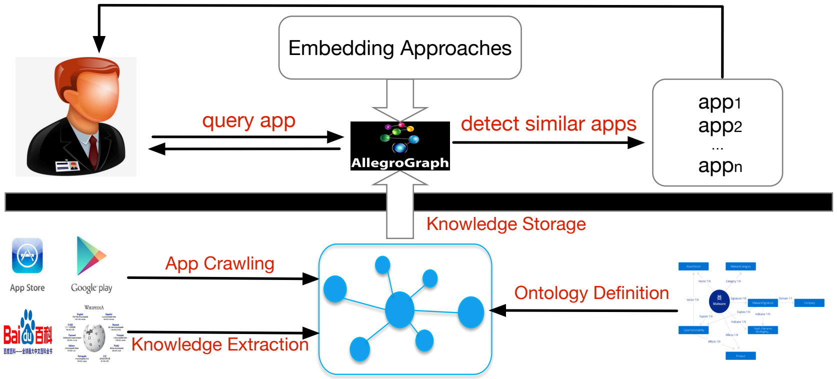

Fig. 1: The framework of detecting similar mobile applications

Fig. 1 presents the framework for detecting similar apps. After we extracted the metadata of apps from application markets and external resources, we further construct a knowledge graph tailored for the service of cybersecurity analysts, in which a lightweight ontology is defined to formalize the basic classes and properties. Benefited from a well-defined schema, more linkages can be shared among apps. To detect similar apps, the underlying idea is to divide the relations in KG into structured and unstructured ones according to their related content. Then, we employ TextRank algorithm to extract important tokens from unstructured texts and transform them into structured triples. In this way, the representations of apps can be iteratively learned by KG embedding methods and network embedding models for improving the performance of similar apps detection. Next, we will illustrate each part in detail.

3.1 The Construction of Mobile Application Knowledge Graph

Ontology definition To model a well-defined schema of apps according to their sensitivity, we discuss with analysts worked on the China Academy of Industrial Internet, and discover that the vast majority of conceptualizations (e.g., function point, interaction mode) described for sensitive apps are not available online. Therefore, we choose appropriate terms based on the survey of existing conceptualizations in view of sensitivity, and define a set of properties by protégé[2] to cover the sensitivity of mobile.

Fig. 2 shows an overview of the light-weight ontology, where red edges and blue ones represent *subclassof* and *rdfs:type* relations, respectively, which are two basic relations. The green ones represent the object properties, and violet ones represent data properties. Overall, we define 30 basic concepts and properties in the ontology. Benefited from a well-defined schema, it not only can make apps to present more comprehensive properties to analysts, but also can generate more shared linkages among apps.

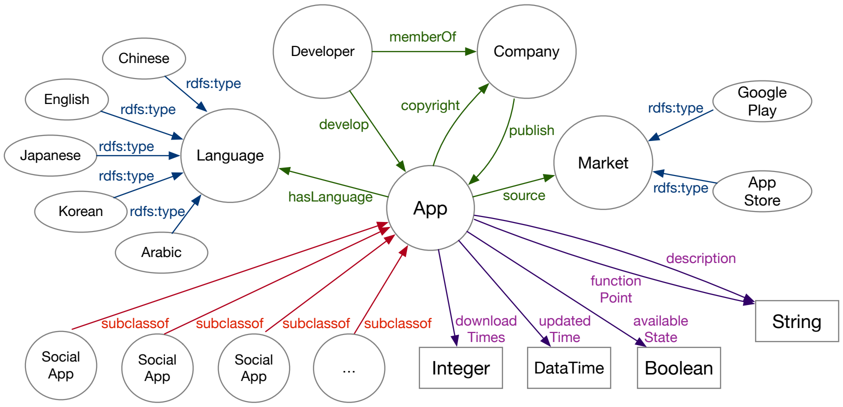

Fig. 2: The overview of lightweight ontology

App crawling With the help of scrapy framework, we crawl the descriptive information of apps published from Google Play and App Store. Note that we do not download application packages of them because we focus on supervising the sensitive apps

[2] https://protege.stanford.edu/

rather than detecting malicious codes of them. To achieve this goal, we design dozens of heuristic principles to guide sensitive apps crawling. The main four principles are listed as follows.

- If the state of one app is not available (e.g., off the shelf), it may be a sensitive app.
- If download times of one app are more than one thousand, it may be a sensitive app.
- If the description of one app contains sensitive tokens (e.g., belle, lottery), it may be a sensitive app.
- If one app shares the same companies or developers with sensitive apps, it may be a sensitive app.

Knowledge extraction Crawling the data of apps from application markets is the most direct way to build KG. However, the known labels often inadequately cover the value of properties in our designed ontology, which impedes the discovery of the shared linkage among apps. Therefore, we try to extract related web pages from external resources (e.g., Baidu Baike[3] and Wikipedia[4]) to fill the lacked value of these properties.

We mainly consider the following strategies to parse the web pages of sensitive apps and populate their property value.

- **String matching method**. It is a mainstream method of knowledge extraction. With the defined mappings between properties (e.g., *Market*) in ontology and attributes (e.g., *Platform*) in inforbox, we can complete the value of several properties.
- **Template-based method**. For one property such as *function point*, we design several templates tailored for textual descriptions in web-pages to obtain the lacked functions of apps.
- **Named entity recognition**. For the concepts *Developer* and *Company*, we utilize the parsed pos of named entity recognition [16](e.g., LSTM+CRF) to capture the related value.

In addition, we make use of the triples asserted in zhishi.me [17] to further complete our KG as many as possible. It is one of the largest knowledge bases that cover three Chinese encyclopedias.

Knowledge storage After we utilized the crawled data to instantiate the properties in our designed ontology and finish knowledge extraction, we transform them into RDF triples $\{(s, p, t)\}$ by Jena[5]. For knowledge storage, we employ AllegroGraph[6] to store the transformed triples, which is one of the efficient graph bases for storing the RDF triples and supporting SPARQL query[7] seamlessly. Benefited from SPARQL query and inference rules implied in ontology, it can present comprehensive information of apps for analysts. To keep the knowledge graph in sync with the evolving apps, we will periodically update the descriptive information of apps by crawling above sources and record the updated logs.

[3] https://baike.baidu.com

[4] http://en.wikipedia.org/wiki/Wiki

[5] http://jena.apache.org/

[6] https://allegrograph.com/

[7] https://www.w3.org/2001/sw/wiki/SPARQL

3.2 Similar Apps Detection based on Iterative Embedding Approaches

To better detect similar apps, we propose an iterative embedding-based framework as shown in Fig. 3. Given the descriptive information of apps, we divide various relations stored in KG into structured and unstructured ones according to their contents. Then, TextRank algorithm is employed to extract important tokens from unstructured texts and transform them into structured triples. Then, we employ KG embedding methods [9] and network embedding (NE) models [10] successively to learn the representations of apps. The learning of the process is iterative. If loss of function is converged or the number of iteration is beyond the default value, this process is terminated.

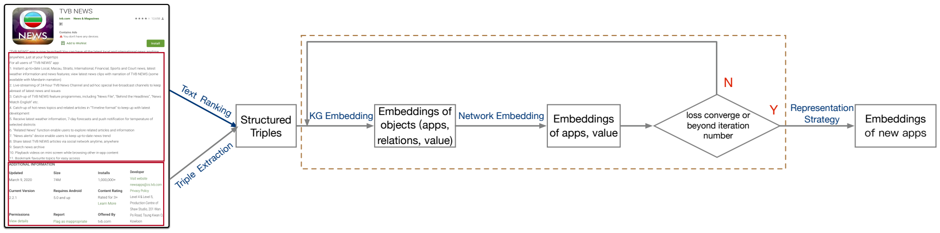

Fig. 3: The workflow of similar apps detection based on embedding approaches

Extracting the structure triples from unstructured texts Note that, KG embedding methods and NE models can not make use of the description texts of apps to enhance the potential correlations of apps. To address this problem, we extract the important tokens of app description texts by TextRank algorithm [8], which is a graph-based ranking model for text processing. The corresponding formula is defined as follows.

$$Sim(S_i, S_j) = \frac{|\{t_k | t_k \in S_i \cap t_k \in S_j\}|}{log(N_i) + log(N_j)}, \quad (1)$$

where $S_i = t_i^1, t_i^2, ..., t_i^{N_i}$ and $S_j = t_j^1, t_j^2, ..., t_j^{N_j}$ are two sentences in the description text of one app, N_i and N_j are the number of tokens in S_i and S_j, t_k is one shared token between two sentences. After iteratively calculated the text-rank value of each token, we can obtain several important tokens by a threshold θ to represent the description text of this app. Then, we introduce a new relation *relatedTo* tailored for these tokens and generate new triples.

Iterative Embedding based on structured triples For structured triples, we try to employ KG embedding methods and network embedding (NE) models to learn the representations of apps iteratively. KG embedding aims to effectively encode a relational knowledge graph into a low dimensional continuous vector space and achieves success on relational learning tasks like link prediction and triple classification [9]. Network embedding can effectively preserve the network structure and capture higher-order similarities among entities [10]. Although they are suitable to model the structured triples, the merits of them are different. KG embedding can learn the representations of entities and relations in KG simultaneously. Relatively, network embedding sacrifices the semantics of edges for capturing higher-order semantic similarities among entities.

Precisely, KG-embedding methods are utilized to pre-train the vector representations of objects such that the semantics of properties can be encoded to some extend. Then, we treat these pre-trained embedding as initial vectors for NE models such that fine-grained semantic representations among apps can be learned. Notice that, our framework is iterative. Hence, learned representations of apps and their value in NE models can be also treated as inputs for KG embedding methods, which is helpful to adjust the vector presentations of properties in KG. Finally, If loss of function is converged or the number of iteration is beyond the default value, this process is terminated. The loss functions[8] of KG embedding methods and NE models are defined in Eq. 2 and Eq. 3.

$$\mathcal{L}_{KGE_{(k+1)}} = \sum_{(h,r,t)\in\xi} \sum_{(h',r,t')\in\xi'} [f_{KGE_{(k)}}(h,r,t) - f_{KGE_{(k)}}((h',r,t')) + \gamma]_+, \quad (2)$$

where ξ is a set of structured triples of KG, ξ' is a negative one generated by negative sampling [18], $[x]_+ \overset{\triangle}{=} max(x,0)$, $f_{KGE_{(l)}}$ represents the score function of KG embedding methods employed in the kth iteration. For the vector representations $\mathbf{h},\mathbf{r},\mathbf{t} \in \mathbb{R}^d$ of each triple (h,r,t), $\mathbf{h}$ and $\mathbf{t}$ need to be replaced with $\mathbf{v}_{NE_{(k)}}, \mathbf{u}_{NE_{(k)}} \in \mathbb{R}^d$ that are the vector representations of nodes trained by NE models in the kth iteration. $\mathbf{r}$ is replaced with $\mathbf{r}_k$, which is trained by KG embedding methods in the kth iteration.

$$\mathcal{L}_{NE_{(k+1)}} = \sum_{i\in V} \sum_{j\in N(v_i)} f_{NE_{(k)}}(v_i, v_j), \quad (3)$$

where V is a set of nodes in network for NE models, $N(i)$ is a set of out-neighbors of node v_i, $f_{NE_{(k)}}$ represents the score function of NE models employed in the kth iteration. For the vector representations $\mathbf{v_i}, \mathbf{v_j} \in \mathbb{R}^d$ of nodes $v_i, v_j \in V$, $\mathbf{v_i}$ and $\mathbf{v_j}$ need to be replaced with $\mathbf{h}_{KG_{(k)}}, \mathbf{t}_{KG_{(k)}} \in \mathbb{R}^d$ that are the vector representations of entities trained by KG embedding methods in the kth iteration.

Notice that, joint learning [19] of above embedding techniques is an alternative framework, including merging pre-training models (e.g. BERT [20]). Nevertheless, the experimental result indicates that the performances of KG embedding methods and pre-training models are not well for detecting similar apps (Section 4.2). Therefore, it may not be suitable to combine them with NE models together for joint learning.

The representations of new apps With helpful of embedding techniques, the similarities between apps can be calculated based on cosine measure. However, it is still challenging for KG embedding methods and NE models to obtain accurate embeddings for new apps because these apps are not fed into the training process.

Existing NE models try to utilize the related information (or entities) of these apps to calculate their similarity. Arithmetic mean [21] and property concatenation [22] are two common strategies to represent the embeddings of new apps. Nevertheless, these

[8] In this paper, KG embedding methods are employed by translated-based models, and NE models mainly consider the effects of out neighbors of nodes in the network.

two strategies ignore the semantics of properties in triples, which assume related information and entities of new apps have the same contributions. Hence, it may not reflect the reality embedding representations for new apps. To address this problem, we further optimized the property concatenation strategy based on entropy. Intuitively, this strategy can utilize the value or entities in each property to measure the importance of itself. Given one new app v_{n+1} and its related information (or entities) denoted by $\{(v_{n+1}, r_k, v_k)\}$, we formalize our strategy by Eqs. 4.

$$\mathbf{v_{n+1}} = w_1\mathbf{v_1} \oplus w_2\mathbf{v_2} \oplus ... \oplus w_m\mathbf{v_m}, \text{ s.t. } v_1, v_2, ..., v_m \in V, \tag{4}$$

$$w_i = \frac{H(p_i)}{\sum_1^l H(p_t)}, \tag{5}$$

where $\mathbf{v_{n+1}}$ is the embedding representation of a new app. $\mathbf{v_1}, \mathbf{v_2} ..., \mathbf{v_m} \in \mathbb{R}^d$ are embedding representations of related information (or entities) $v_1, v_2, ..., v_m$, which belong to a set of nodes V in the network. $\oplus$ is a concatenate operation $\mathbb{R}^{a\times d} \oplus \mathbb{R}^{b\times d} \rightarrow \mathbb{R}^{(a+b)\times d}$, w_i is a weight calculated by all the entropy of app properties, $H(p_i)$ is an entropy of all the value and entities of property p_i, l is the number of them.

4 Evaluation

In this section, we report the statistic of constructed KG for mobile applications, called MAKG, and verify the effectiveness of our proposed framework for detecting similar apps. Our approach[9] can be downloaded together with the datasets.

4.1 Statistics of the constructed knowledge graph

Statistics of MAKG are listed in Table 1, in which more than 241 thousand apps are collected and they are divided into four categories, including Tools, Social, News and Newspaper&Magazine. The last column lists the whole number of apps, entities, relations in MAKG. Due to the defined schema of apps, the number of relations in each category is the same. Notice that MAKG is a multilingual knowledge graph because the language of their names and descriptions includes Chinese, English, Japanese, Korean and Arabic.

Table 1: The detailed statistics of MAKG

Category	Tools	Social	News	Newspaper&Magazine	Total
♯Apps	129,730	70,948	36,325	4,422	241,425
♯Relations	30	30	30	30	30
♯Entities	235,333	146,598	70,386	4,369	445,028

Datasets To evaluate the effectiveness of our method, we select some apps with Chinese and English in MAKG and build a benchmark dataset named MAKG-E listed in Table 2.

[9] https://github.com/zbyzby11/MAKG4Embedding

Table 2: Statistics of datasets for evaluation

Dataset	Train			Test
	♯ Apps	♯ Nodes	♯ Edges	♯ Apps
MAKG-E	61773	126817	432411	100
MAKG-E^+	61773	165838	628458	100

For the test set, we invite several experienced analysts to randomly select 100 new apps that are separated from the training set. For each app, analysts select 20 most similar apps in the training as a standard set, which are from the candidate apps generated by TF-IDF algorithm based on their textual descriptions. MAKG-E^+ is an enhanced one that has integrated important tokens and corresponding relationships of apps based on TextRank algorithm, in which the threshold θ for selecting important tokens is set to 0.5.

Metrics According to the built benchmarks, we introduce two metrics from the field of information retrieval to evaluate ranking methods that are formally defined as follows.

$$RR = \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{1}{Rank_{ij}} \qquad Rank_{min} = \frac{1}{n}\sum_{i=1}^{n}\arg\min_{l} Rank_{il}.$$

The first metric is reciprocal rank, written RR, which is defined as the sum of the reciprocal of $Rank_{ij}$. $Rank_{ij}$ indicates the jth similar apps in descending order for the ith tested app. If the jth similar app belongs to the standard set, then $Rank_{ij} = j$. Otherwise, the rank value is 0. The second metric, written Rank$_{min}$, is defined as the minimum rank of similar apps in descending order for each given app. The larger RR is, the closer of the similar search list is to the ideal one. Relatively, the smaller Rank$_{min}$ is, the earlier people can see similar apps. Notice that, as similar apps in the standard set are not unique, we do not employ AUC (Area Under Curve) as a metric, which is one of the important indicators to evaluate the effectiveness of classification models.

4.2 Evaluation of Similar Apps Detection

Implementation details For structured triples, we employ TransE [18], TransH [23], TransD [24] by OpenKE platform[10] to train them and obtain the vector representations of apps by the average strategy in [25]. The network embedding models are implemented based on DeepWalk [26], LINE [27], Node2Vec [28] by OpenNE platform[11]. For our iterative framework, we employ TransE as KG embedding method for pre-training the vector representations of apps, and utilize LINE as NE model to learn the embeddings of apps and their value. Because both of them are efficient for large-scale representation learning. The number of iteration in our framework is set to 3 as default.

In addition, we employ a feature matching method (abbreviated as FM) and the pre-training model BERT [20] as baselines[12] to verify the effectiveness of our framework.

[10] https://github.com/thunlp/OpenKE

[11] https://github.com/thunlp/OpenNE

[12] As TF-IDF algorithm is selected for generating candidate apps for the test set, we do not employ it as a baseline.

FM is implemented by calculating the overlapping entities related to apps based on Jaccard similarity[13]. Relatively, we transform all the triples into textual descriptions and feed them into BERT[14] for detecting similar apps of new ones.

To ensure a fair comparison, we fine-tune the hyperparameters (e.g., dimension, learning rate, negative sampling number) of above models to obtain the best results.

The evaluation results Table 3 lists comparison results of different embedding methods in terms of RR and Rank$_{min}$. From the table, we can observe that:

- Benefited from TextRank algorithm applied in MAKG-E^+, FM and NE-based models can gain significant improvements compared with the original MAKG-E. Because these important tokens and relations can enrich the contexts of apps. It is helpful to capture more semantic correlations among apps.
- NE-based models (e.g., LINE, Node2Vec, our method) outperform other models in both two datasets. It indicates that NE-based models can obtain fine-grained correlations among apps by the constructed network in their models. Notice that our method is slightly better than LINE and Node2Vec because it has been incorporated the semantics of relations by TransE.
- The performances of KG embedding methods and BERT are not well for detecting similar apps. We analyze that the inherent characters (e.g., multilingual textual descriptions, insufficient triples) of constructed datasets may affect them to capture accurate representations of apps.

Table 3: Comparison results in terms of RR and Rank$_{min}$

Methods	MAKG-E		MAKG-E^+	
	RR	Rank$_{min}$	RR	Rank$_{min}$
FM	93.60	5.14	165.10	2.37
BERT	0.92	20.47	3.03	20.00
TransE	22.83	17.30	12.10	18.44
TransH	22.86	17.38	11.90	18.51
TransD	23.15	17.31	11.35	18.86
DeepWalk	83.55	9.07	117.68	4.18
Line	99.38	**5.05**	188.93	2.17
Node2vec	98.28	6.24	173.13	2.21
Our method	**100.40**	5.98	**190.80**	**1.87**

The results of different representation strategies of new apps Table 4 and Table 5 show the results of different representation strategies of new apps. Overall, NE-based models with the concatenation strategy based on entropy are better than the original one and arithmetic mean. It indicates that our optimized strategy can solve the representation problem of new apps to some extent. Our method can obtain better performances

[13] https://wiki2.org/en/Jaccard_index

[14] https://github.com/hanxiao/bert-as-service

than other models in terms of RR and Rank$_{min}$ because the fine-grained semantics of relations are encoded in our iterative framework. Nevertheless, Node2Vec equipped with our strategy does not perform satisfactorily in MAKG-E^+. We discover that the test apps corresponding to good results equipped with arithmetic mean are different from the ones with our strategy. It makes sense to combine these two strategies for detecting similar apps in our future work.

Table 4: Comparison results of different representation strategies in terms of RR

Dataset	Arithmetic Mean ($*$)				Concatenation ($\star$)				Concatenation based on Entropy ($\diamond$)			
	DeepWalk	LINE	Node2Vec	Our method	DeepWalk	LINE	Node2Vec	Our method	DeepWalk	LINE	Node2Vec	Our method
MAKG-E	83.2	71.0	93.7	75.7	68.0	49.0	51.2	65.9	83.6	99.4	98.3	**100.4**
MAKG-E^+	106.4	170.0	173.1	177.1	88.6	73.9	94.7	47.3	117.7	188.9	118.9	**190.8**

Table 5: Comparison results of different representation strategies in terms of Rank$_{min}$

Dataset	Arithmetic Mean ($*$)				Concatenation ($\star$)				Concatenation based on Entropy ($\diamond$)			
	DeepWalk	LINE	Node2Vec	Our method	DeepWalk	LINE	Node2Vec	Our method	DeepWalk	LINE	Node2Vec	Our method
MAKG-E	9.41	7.28	6.08	7.62	8.38	10.29	9.90	8.33	9.07	**5.05**	6.24	5.98
MAKG-E^+	5.28	2.89	2.21	2.85	6.92	8.85	5.71	14.25	4.18	2.17	4.53	**1.87**

Case study Table 6 lists three real cases of similar apps detection tested on MAKG-E$^+$. Overall, our method outperforms FM and other embedding techniques. For *17 LIVE* and *9CHAT*, the minimum ranks of them in FM are beyond the 12th and 8th, respectively. Relatively, these values in our method are significantly improved and obtain the ideal ranking. In terms of RR, the performances of NE-based models are better than the ones of FM and other embedding techniques. It indicates that more apps in the standard set can be detected.

Table 6: Three real cases of similar apps detection tested on MAKG-E$^+$

Name of app	Fast News		17 LIVE		9CHAT	
	RR	Rank$_{min}$	RR	Rank$_{min}$	RR	Rank$_{min}$
FM	1.42	**1**	0.08	12	0.13	8
TransE	0.43	8	0.00	20+	0.00	20+
TransH	0.36	13	0.00	20+	0.00	20+
TransD	0.67	3	0.00	20+	0.00	20+
DeepWalk$^\diamond$	1.27	**1**	**1.00**	**1**	0.10	10
Line$^\diamond$	2.54	**1**	0.31	4	**3.00**	**1**
Node2vec*	1.85	2	**1.00**	**1**	2.48	**1**
Our method$^\diamond$	**2.81**	**1**	0.56	2	2.66	**1**

5 Conclusion and Future Work

In this paper, we presented a novel framework by knowledge graph techniques and embedding models, which is suitable for cybersecurity analysts to find similar apps. We further designed a light-weight ontology for the construction of knowledge graph, which can present more comprehensive information of apps to analysts and generate more shared linkages among apps. To obtain accurate representations of new apps, we employed TextRank algorithm to enhance the structured information and optimized the concatenation strategy for network embedding models. Moreover, KG embedding methods and network embedding models are employed to iteratively learn the representations of apps, which can further improve the performance of similar apps detection. The preliminary result indicated the effectiveness of our framework comparing to several existing methods in terms of reciprocal ranking and minimum ranking.

In future work, we will collect more apps to enrich MAKG, and employ techniques to achieve the alignment of multilingual texts, which is helpful to improve the performances of similar apps detection.

References

1. Guozhu Meng, Matthew Patrick, Yinxing Xue, Yang Liu, and Jie Zhang. Securing Android App Markets via Modeling and Predicting Malware Spread Between Markets. *IEEE Trans. Information Forensics and Security*, 14(7):1944–1959, 2019.
2. Ning Chen, Steven C. Hoi, Shaohua Li, and Xiaokui Xiao. SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning. In *WSDM*, pages 305–314, 2015.
3. Upasna Bhandari, Kazunari Sugiyama, Anindya Datta, and Rajni Jindal. Serendipitous Recommendation for Mobile Apps Using Item-Item Similarity Graph. In *AIRS*, pages 440–451, 2013.
4. Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation: a contest between satisfaction and temptation. In *WSDM*, pages 395–404, 2013.
5. Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In *SIGIR*, pages 533–542, 2015.
6. Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. Scrutinizing mobile app recommendation: Identifying important app-related indicators. In *AIRS*, pages 197–211, 2016.
7. Afnan A. Al-Subaihin, Federica Sarro, Sue Black, and Licia Capra. Empirical comparison of text-based mobile apps similarity measurement techniques. *Empirical Software Engineering*, 24(6):3290–3315, 2019.
8. Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Text. In *EMNLP*, pages 404–411, 2004.
9. Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Trans. Knowl. Data Eng*, 29(12):2724–2743, 2017.
10. Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A Survey on Network Embedding. *IEEE Trans. Knowl. Data Eng*, 31(5):833–852, 2019.
11. Franz-Xaver Geiger and Ivano Malavolta. Datasets of Android Applications: a Literature Review. *CoRR*, abs/1809.10069, 2018.
12. Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *NDSS*, 2014.
13. Daniel E. Krutz, Mehdi Mirakhorli, Samuel A. Malachowsky, Andres Ruiz, Jacob Peterson, Andrew Filipski, and Jared Smith. A dataset of open-source android applications. In *MSR*, pages 522–525, 2015.

14. Li Li, Jun Gao, Médéric Hurier, Pingfan Kong, Tegawendé F. Bissyandé, Alexandre Bartel, Jacques Klein, and Yves Le Traon. AndroZoo++: Collecting Millions of Android Apps and Their Metadata for the Research Community. *CoRR*, abs/1709.05281, 2017.
15. Guozhu Meng, Yinxing Xue, Jing Kai Siow, Ting Su, Annamalai Narayanan, and Yang Liu. AndroVault: Constructing Knowledge Graph from Millions of Android Apps for Automated Analysis. *CoRR*, abs/1711.07451, 2017.
16. Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *CoRR*, abs/1812.09449, 2018.
17. Xing Niu, Xinruo Sun, Haofen Wang, Shu Rong, Guilin Qi, and Yong Yu. Zhishi.me - weaving chinese linking open data. In *ISWC*, pages 205–220, 2011.
18. Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.
19. Yang Gao, Yue Xu, Heyan Huang, Qian Liu, Linjing Wei, and Luyang Liu. Jointly learning topics in sentence embedding for document summarization. *IEEE Trans. Knowl. Data Eng.*, 32(4):688–699, 2020.
20. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, pages 4171–4186, 2019.
21. Jian Tang, Meng Qu, and Qiaozhu Mei. PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks. In *SIGKDD*, pages 1165–1174, 2015.
22. Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *SIGKDD*, pages 839–848, 2018.
23. Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*, pages 1112–1119, 2014.
24. Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *ACL*, pages 687–696, 2015.
25. Meng Wang, Ruijie Wang, Jun Liu, Yihe Chen, Lei Zhang, and Guilin Qi. Towards empty answers in sparql: Approximating querying with rdf embedding. In *ISWC*, pages 513–529, 2018.
26. Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *SIGKDD*, pages 701–710, 2014.
27. Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *WWW*, pages 1067–1077, 2015.
28. Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *SIGKDD*, pages 855–864, 2016.