An Android Malware Detection Approach Based on SIMGRU

HANXUN ZHOU\textsuperscript{1}, XINLIN YANG\textsuperscript{1}, HONG PAN\textsuperscript{2}, AND WEI GUO\textsuperscript{3}

\textsuperscript{1}School of Information, Liaoning University, Shenyang 110036, China
\textsuperscript{2}School of Economics, Liaoning University, Shenyang 110036, China
\textsuperscript{3}School of Computer, Shenyang Aerospace University, Shenyang 110136, China

Corresponding author: Hong Pan (panhong@lnu.edu.cn)

This work was supported in part by the National Key Research and Development Program under Grant 2019YFB1406002, in part by the National Science Foundation of China under Grant 51704138, in part by Liaoning Education Department Grant Nos. JYT19053, and in part by National Natural Science Foundation of Liaoning Nos. 2020-MS-239.

\textbf{ABSTRACT} With the rapid development of the Internet era, the number of malware has reached an unprecedented peak, and therefore malware is threatening global network security seriously. In this article, we propose an Android malware detection approach based on SIMGRU, which belongs to the static detection approach. The similarity of clustering is widely used in static analysis of android malware, so we introduce the similarity to improve Gated Recurrent Unit (GRU), and obtain three different structures of SimGRU: InputSimGRU, HiddenSimGRU, and InputHiddenSimGRU. The experiment shows that InputSimGRU, HiddenSimGRU, and InputHiddenSimGRU outperform the conventional GRU model and other methods.

\textbf{INDEX TERMS} Network security, android malware detection, GRU, SIMGRU, deep learning.

I. INTRODUCTION

Nowadays, a large number of mobile software is released constantly into the application market, which provides favorable conditions for malicious software spread. As a result, newer and more complicated Android malware are emerging, along with the rising number of malicious Android apps\textsuperscript{[1]}. Malware targeting the Android platform is continuously improving in performance as well as applying new techniques to thwart analysis and to avoid detection. Therefore, it can be seen that Android malware on mobile phones is active and is jeopardizing the security of the mobile terminal persistently.

To detect malware effectively, experts proposed many approaches to identify malware\textsuperscript{[2]–[9]}. They can be classified as static detection, dynamic detection, and hybrid detection. The static detection decompiles the application program by reverse engineering and extracts critical features to judge whether there is malicious code. Static analysis is based on matching analysis, which can detect known malware quickly and effectively. The dynamic detection approach is to identify malicious behavior while the application is running in the simulated environment. The hybrid detection method combines the dynamic and static detection. This article mainly studies the static detection approach of Android malware. Sun \textit{et al.}\textsuperscript{[2]} presented a feature extraction method in JAVA source code. And SVM is applied to make the system to accommodate the function of new malicious samples to detect new malicious software and existing malware. Wang \textit{et al.}\textsuperscript{[3]} designed a novel method to extract permission patterns on the differences between Android benign apps and malware and used these differences to detect Android malware. Talha \textit{et al.}\textsuperscript{[4]} presented a permission-based Android malware detection system that applies static analysis to classify Android applications as benign or malicious. Fan \textit{et al.}\textsuperscript{[5]} proposed a malware family detection system based on frequent subgraphs. Android malware is automatically classified based on frequent subgraphs representing the features of the same malware family. Shang \textit{et al.}\textsuperscript{[6]} proposed an Android malware detection model based on improved naive Bayes classification. Allix \textit{et al.}\textsuperscript{[7]} devised several machine learning classifiers that rely on a set of features built from applications’ CFGs. Wu \textit{et al.}\textsuperscript{[8]} proposed an Android malware detecting system that provides accurate classification and sensitive data analysis. The study adopted a machine learning approach that leverages the use of dataflow application program interfaces (APIs) as classification features.
to detect Android malware. Hou et al. [9] constructed the weighted directed graphs and then applied a deep learning framework resting on the graph-based features for unknown Android malware detection.

Although many machine learning approaches are proposed to detect android malware, there are still some problems. Firstly, the effectiveness of these methods, except deep learning approaches depend on the features extracted manually. Secondly, although there are many deep learning approaches for android malware detection, most of them only apply existing deep learning technology to detect malware and do not associate characteristics of malware detection with deep learning.

This article presents an Android malware detection approach based on SimGRU, which belongs to the static malware detection. Due to the wide applicability of the similarity in the static malware detection, we introduce the similarity principle to the GRU cell and establish a new Android malware detection model-SimGRU. Furthermore, we classify the model into three types: InputSimGRU, HiddenSimGRU, and InputHiddenSimGRU. The experiment shows that the proposed approach has a better performance than the traditional GRU model and the other three methods.

The rest of this article is organized as follows. In section II, we review the work related to Android malware detection. In section III, we introduce the traditional GRU model. In section IV, we explain how to improve GRU to SimGRU in detail. Section V is the experiment. Finally, we summarize the article in Section VI.

II. RELATED WORK

With the rapid development of malware and the emergence of various polymorphic and deformation techniques, traditional detection methods have become inefficient. In recent years, people’s gradual attention to network security has accelerated the research on network security, which has led to the rapid development of Android malware detection.

Junaid et al. [10] presented a static analysis framework called Dexteroid, which uses reverse engineer life cycle models to capture the behavior of Android components accurately. Aafer et al. [11] introduced a lightweight classifier to detect malicious software—Droid APIMiner, which captures the relevant features of malicious software. Fan et al. [12] proposed the malicious application detection method (DAPASA) to detect Android malicious applications by sensitive subgraph analysis. DAPASA generates sensitive subgraphs, which extract five features from the sensitive subgraph to describe the different calling modes of the application’s sensitive API. As an essential feature of Android apps, the function call graph can describe the behavior of Android applications very well. Wang et al. [13] extracted a very large number of features from each app and categorized them into two groups, app-specific, and platform-defined features. These features would be fed into four classifiers (i.e., Logistic Regression, linear SVM, Decision Tree, and Random Forest) for the malware detection. Li et al. [14] proposed an Android malware detection system based on feature fusion to improve the detection efficiency of Android malicious applications.

Shang et al. [15] performed malware detection by the similarity of the function call graphs. In terms of code obfuscation, function call graphs are more robust than traditional feature codes. Xu et al. [16] proposed a method for identifying malware variants based on function call graphs. Firstly, the malicious program is disassembled to propose a function call graph, and then the graph coloring technique is used to calculate the similarity function between two function call graphs. Finally, the similarity function is used to identify variants of known malware. Utku and Doğru [17] developed a permission-based detection system based on Android malware’s machine learning methods. Song et al. [18] proposed a static detection framework, which consists of four layers of filtering mechanisms, the message digest values, the combination of malicious permissions, the dangerous permissions, and the dangerous intention. Rastogi et al. [19] studied some of the repackaging detection techniques in detail.

III. GRU

GRU [20] is a special RNN model that solves the gradient disappearance problem of native RNN and is widely applied in classification and other fields. There are two gates in the structure of GRU: the reset gate and the update gate. The structure of two gates can learn from the hidden state of the input and the output of the previous GRU unit adaptively. The main structure of the GRU cell is shown as figure 1.

At the time step t, the learning process is calculated as follows:

\[ h_t = z_t e h_{t-1} + (1-z_t) e \tilde{h}_t \]  
\[ z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \]  
\[ \tilde{h}_t = \tanh(W_h x_t + r_t (U_h h_{t-1} + b_h)) \]  
\[ r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \]

where \( x_t \) is the input vector at the time step t, \( \sigma \) is the sigmoid activation function, \( W_z, W_h, \) and \( W_r \) are the mapping matrices, \( U_z, U_h, \) and \( U_r \) are the weight matrices, and b is the bias.
As can be seen from figure 1, \( x_t \) is a vector of the original input, and \( h_t \) is the hidden state learned from the hidden state of GRU in the previous step \( t-1 \) and the input vector \( x_t \) at the time step \( t \). \( r_t \) is the output of the reset gate, and \( z_t \) is the output of the update gate. Since the input vector and the hidden state represent different information, they control the information transferring process jointly in the GRU unit. As a result, the reset gate and the update gate perform similar calculations, making it possible to memorize as much as possible in the GRU structure.

**IV. SIMGRU**

We introduce the principle of similarity to GRU to improve the performance of android malware detection due to the similarity function that other works utilize to detect malware. We name the proposed model as the SIMGRU model.

**A. BASIC PRINCIPLES**

Android malware, which belongs to the same family, usually calls common sensitive functions because these functions often implement similar functions. As a result, there are similarities in the calling sequence of sensitive functions invoked by the same family malware. Therefore, similarity functions are often utilized to represent the similarity.

There are many similarity functions. Euclidean distance, which expresses the similarity between two objects by calculating the angle between two vectors, is also widely used in malware detection. The calculation formulas to calculate the angle between two vectors, which is usually used to calculate the distance between them, is one of the most commonly used functions. The Cosine function is usually used to calculate the angle between two vectors, which is also widely used in malware detection. The calculation formulas of Euclidean distance and Cosine function are as follows:

\[
\text{sim}_{\text{Em}}(h_{t-1}, h_t) = \frac{[(h_{t-1} - h_t) \cdot (h_{t-1} - h_t)]^{\frac{1}{2}}}{\|h_{t-1}\| \|h_t\|} \tag{5}
\]

\[
\text{sim}_{\text{Cos}}(h_{t-1}, h_t) = \frac{h_{t-1} \cdot h_t}{\|h_{t-1}\| \|h_t\|} \tag{6}
\]

Thus, we introduce similarity functions to GRU to detect Android malware due to the performance of similarity functions in malware detection, which we name as SimGRU. Furthermore, we classify the models into three types: InputSimGRU, HiddenSimGRU, and InputHiddenSimGRU. The static features of both benign apps and malware are converted to the vector \( x_t \), which is input to the SimGRU model. The static features include permissions, API calls, and network addresses, and so on.

**B. INPUTSIMGRU**

The input vector \( x_t \) of GRU is a vectorized representation of the original data. The similarity function can be represented by \( s = \text{sim}(x_{t-1}, x_t) \). The similarity between \( x_{t-1} \) and \( x_t \) can be calculated by \( \Delta x = (1-s)x_{t-1} \). Therefore, we introduce the similarity function to GRU, which is named InputSimGRU. The basic cell unit is shown in figure 2.

The position pointed by the wide arrow is calculated based on the similarity function of the input \( x_t \). The similarity between the inputs at two adjacent time steps and the current input is input to the reset gate and update gate, which determines whether to trigger the gates. Therefore, the gates are computed as follows:

\[
z_t = \sigma(W_z x_t + U_z (1 - \text{sim}(x_{t-1}, x_t))x_{t-1} + U_z h_{t-1} + b_z) \quad \tag{7}
\]

\[
r_t = \sigma(W_r x_t + U_r (1 - \text{sim}(x_{t-1}, x_t))x_{t-1} + U_r h_{t-1} + b_r) \quad \tag{8}
\]

The input \( x_t \) is still selected as the input to candidate states \( h_t \) to memorize the input at the current time step:

\[
h_t = z_t e h_{t-1} + (1 - z_t) e \tilde{h}_t \quad \tag{10}
\]

**C. HIDDENSIMGRU**

The hidden state \( h_t \) of GRU simulates the process of human neurons, so it is the representation of GRU cells, which may help detect Android malware more effectively. As a result, the similarity of the hidden states instead of inputs can be calculated as \( s = \text{sim}(h_{t-2}, h_{t-1}) \), and the similarity between two adjacent hidden states is \( \Delta h = (1-s)h_{t-2} \). The proposed GRU is named HiddenSimGRU, whose cell is shown in figure 3.
The position pointed by the wide arrow is the similarity function calculated based on the hidden state. The similarity of the hidden state $h_{t-2}$ and $h_{t-1}$ is calculated at the current time step $t$. Then the similarity between two adjacent hidden states and the current hidden state is input to the reset gate and update gate. The formula is calculated as follows:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + U_{\Delta h}(1 - \text{sim}(h_{t-2}, h_{t-1})))h_{t-2} + b_r$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + U_{\Delta h}(1 - \text{sim}(h_{t-2}, h_{t-1})))h_{t-2} + b_z$$

Other states are computed as the same as those in the cell of InputSimGRU.

**D. INPUTHIDDEN SIMGRU**

Since the input and the hidden state can help detect Android malware from different perspectives, so the similarity of the input and the hidden state can help detect Android malware adaptively. The learning result is merged through the full connection layer and predicted by the SoftMax layer to determine whether the Android application is malicious or not.

**Algorithm 1 Training**

*Initialize: data, label*

for $t = 1, ......, T$ do:

- calculate the similarity $s = \text{sim}(x_{t-1}, x_t)$ or $s = \text{sim}(h_{t-2}, h_{t-1})$ or both;
- calculate the reset gate and the update gate as (15-16);
- calculate hidden layer status $h_t$;

End

Predict = Softmax(Output);
Loss = CrossEntropy(Predict, Label);
GradientDescent(Loss);
predict = f(output);

**V. EXPERIMENT**

To evaluate the performance of Android malware detection based on SIMGRU, we used the DREBIN dataset [21] for the experiment. We performed 10-fold cross-validation due to the limited malicious samples. The dataset was partitioned into 10 subsets, one subset was used as the testing set, and another one was used as the validation set, and the rest were used for the training set. Furthermore, we conducted the experiment 10 times so that each subset was selected once as the test dataset. The effectiveness of different SIMGRU structures on Android malware detection is analyzed.

**A. DATASET**

There are 123,453 benign applications, which were collected from Google play, Chinese markets, Russian markets, and other sources in DREBIN dataset [21], and also a collection of 5560 malicious samples from 179 different malware families collected between 2010 and 2012. All samples
Three metrics were selected as the metrics to evaluate the performance of the algorithm as follows:

Accuracy can be calculated as (19):

\[
\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \tag{19}
\]

Precision is calculated as follows:

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{20}
\]

Recall is computed as follows:

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{21}
\]

The true positives (TP) is the number of malicious applications classified as malicious applications. The true negatives (TN) is the number of benign applications classified as benign applications. The false positives (FP) is the number of benign applications classified as malicious applications. The false negatives (FN) is the number of malicious applications classified as benign applications.

in the Android malware genome project [22] are included. This dataset is one of the well-known android malware datasets to evaluate the detection approach.

| GRU structure       | Accuracy | Precision | Recall  |
|---------------------|----------|-----------|---------|
| GRU                 | 98.03%   | 98.14%    | 99.57%  |
| InputSimGRU         | 98.95%   | 98.47%    | 99.40%  |
| HiddenSimGRU        | 99.11%   | 98.61%    | 99.51%  |
| InputHiddenSimGRU   | 99.30%   | 98.79%    | 99.50%  |

FIGURE 5. Accuracy of both GRU and SIMGRU in the training process.

TABLE 1. Detection results.
malware. As a result, it can capture more static features to identify than GRU due to the Sim function calculated in the SimGRU SimGRU can focus on the similarity of malware much more detect malware better than in either layer. That is to say, because the sim function in both layers can help SimGRU -InputHiddenSimGRU outperforms the other two models. The combination of InputSimGRU and HiddenSimGRU can capture the similarity of malware better than GRU. That might be the introduction of Sim functions, which include that SimGRU has great superiority over GRU.

As can be seen from Table 1, the accuracy of SimGRU is better than GRU by about 1%. Furthermore, InputHiddenSimGRU has the highest accuracy, whereas HiddenSimGRU is better than InputSimGRU. The precision follows almost the same principals with accuracy. However, there is little difference in the recall rate.

The accuracy curves, representing the learning ability of different GRU structures for Android malware detection in the training process, are shown in figure 5. It can be seen that when epoch ∈ [0,4), the training accuracy increases dramatically; when epoch ∈ [4,60), accuracy gradually becomes steady after a tremendous increase.

One example of the detection process is shown in figure 6 to represent the capacity of SimGRU to detect malware as the time step increases. The process of the detection is from ‘‘normal’’ to ‘‘malicious’’, which shows the ability of the models to identify malware. SimGRU is faster than the traditional GRU in detecting malware. Furthermore, InputHiddenSimGRU is the fastest, and HiddenSimGRU is faster than InputSimGRU in identifying malware.

Taking all the factors into account, we can conclude that SimGRU has great superiority over GRU. That might be the introduction of Sim functions, which can capture the similarity of malware better than GRU. The combination of InputSimGRU and HiddenSimGRU—InputHiddenSimGRU outperforms the other two models because the sim function in both layers can help SimGRU detect malware better than in either layer. That is to say, SimGRU can focus on the similarity of malware much more than GRU due to the Sim function calculated in the SimGRU cell. As a result, it can capture more static features to identify malware.

### TABLE 2. Test results of GRU and SimGRU.

| Method      | Accuracy | Precision | Recall |
|-------------|----------|-----------|--------|
| Random Forest | 96%      | 97%       | 95%    |
| SVM         | 94%      | 94%       | 94%    |
| KNN         | 92%      | 90%       | 96%    |

### C. COMPARISON WITH OTHER METHODS

To test the feasibility of the proposed model, we compare SimGRU with the other three methods [23], including Support Vector Machine, Random Forest, KNN. For the sake of fair comparison, we conducted the experiments the same as the proposed models.

We reported the results in table 2. The SimGRU model outperforms the three methods in Accuracy, Precision, and Recall rate, whereas the GRU model also works better than them. Thus, there is the competitive performance of the SimGRU model compared with both GRU and other three approaches. That means that it holds great superiority in detecting Android malware.

### VI. CONCLUSION

We propose an Android malware detection approach based on SIMGRU. This method uses static analysis to analyze Android malware. Due to the wide applicability of similarity in Android malware detection, we improve the traditional GRU model to SimGRU which is based on the similarity principle, and propose three different GRU structures: InputSimGRU, HiddenSimGRU and InputHiddenSimGRU. The experiment shows that SimGRU has a better performance than GRU and the other three methods.

### REFERENCES

[1] F. Jaafar, G. Singh, and V. Zavarsky, ‘‘An analysis of Android malware behavior,’’ in Proc. IEEE. Int. Conf Software Qual., Rel. Secur. Companion, Lisbon, Portugal, Jul. 2018, pp. 337–341.

[2] J. Sun, K. Yan, X. Liu, C. Yang, and Y. Fu, ‘‘Malware detection on Android smartphones using keywords vector and SVM,’’ in Proc. IEEE/ACIS 16th Int. Conf. Comput. Inf. Sci. (ICIS), Wuhan, China, May 2017, pp. 833–838.

[3] C. Wang, Q. Xu, X. Lin, and S. Liu, ‘‘Research on data mining of permissions mode for Android malware detection,’’ Cluster Comput., vol. 22, no. S6, pp. 13337–13350, Nov. 2019.

[4] K. A. Talha, D. I. Aliper, and C. Aydin, ‘‘APK auditor: Permission-based Android malware detection system,’’ Digit. Invest., vol. 13, pp. 1–14, Jun. 2015.

[5] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, ‘‘Android malware familial classification and representative sample selection via frequent subgraph analysis,’’ IEEE Trans. Inf. Forensics Security, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[6] F. Shang, Y. Li, X. Deng, and D. He, ‘‘Android malware detection method based on Naive Bayes and permission correlation algorithm,’’ Cluster Comput., vol. 21, no. 8, pp. 1–12, 2017.
148410

H. Zhou et al.: Android Malware Detection Approach Based on SIMGRU

K. Aliki, T. F. Bissyandé, Q. Jérome, J. Klein, and Y. Le Traon, “Empirical assessment of machine learning-based malware detectors for Android,” *Empirical Softw. Eng.*, vol. 21, no. 1, pp. 183–211, 2016.

S. Wu, P. Wang, X. Li, and Y. Zhang, “Effective detection of Android malware based on the usage of data flow APIs and machine learning,” *Inf. Softw. Technol.*, vol. 75, pp. 17–25, Jul. 2016.

S. Hou, A. Saas, L. Chen, and Y. Ye, “Deep4MalDroid: A deep learning framework for Android malware detection based on Linux kernel system call graphs,” in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Workshops (WIW)*, Omaha, NE, USA, Oct. 2016, pp. 104–111.

M. Junaid, D. Liu, and D. Kung, “Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models,” *Comput. Secur.*, vol. 59, pp. 92–117, Jun. 2016.

Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, Sep. 2013, pp. 86–103.

M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, “DAPASA: Detecting Android piggybacked apps through sensitive subgraph analysis,” *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1772–1785, Aug. 2017.

X. Wang, W. Wang, Y. He, J. Liu, Z. Han, and X. Zhang, “Characterizing Android apps' behavior for effective detection of malware at large scale,” *Future Gener. Comput. Syst.*, vol. 75, pp. 30–45, Oct. 2017.

J. Li, Z. Wang, T. Wang, J. Tang, Y. Yang, and Y. Zhou, “An Android malware detection system based on feature fusion,” *Chin. J. Electron.*, vol. 27, no. 6, pp. 1206–1213, Nov. 2018.

S. Shang, N. Zheng, J. Xu, M. Xu, and H. Zhang, “Detecting malware variants via function-call graph similarity,” in *Proc. 5th Int. Conf. Malicious Unwanted Softw.*, Oct. 2010, pp. 113–120.

M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren, and N. Zheng, “A similarity metric method of obfuscated malware using function-call graph,” *J. Comput. Virol. Hacking Techn.*, vol. 9, no. 1, pp. 35–47, Feb. 2013.

A. Utku and I. A. Dogru, “Malware detection system based on machine learning methods for Android operating systems,” in *Proc. 25th Signal Process. Commun. Appl. Conf. (SIU)*, Antalya, Turkey, May 2017, pp. 1–4.

J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, “An integrated static detection and analysis framework for Android,” *Pervas. Mobile Comput.*, vol. 32, pp. 15–25, Oct. 2016.

S. Rastogi, K. Bhushan, and B. B. Gupta, “Android applications repackaging detection techniques for smartphone devices,” *Procedia Comput. Sci.*, vol. 78, pp. 26–32, Mar. 2016.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” 2014, arXiv:1406.1078. [Online]. Available: https://arxiv.org/abs/1406.1078.

D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of Android malware in your pocket,” in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.

A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark Android malware datasets and classification,” in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2018, pp. 1–7.

H. Zhang, S. Luo, Y. Zhang, and L. Pan, “An efficient Android malware detection system based on method-level behavioral semantic analysis,” *IEEE Access*, vol. 7, pp. 69246–69256, 2019.

**HANXUN ZOU** was born in 1981. He received the master’s and Ph.D. degree in computer application technology from Northeastern University, in 2006 and 2009, respectively. He is currently an Associate Professor with Liaoning University. His research interest includes network security, especially malware.

**XINLIN YANG** was born in 1996. He graduated from the Shandong University of Technology, in 2019. He is currently pursuing the master’s degree with Liaoning University. Follow Dr. Zhou Hanxun to learn the direction of network security based on machine learning, etc.

**HONG PAN** was born in 1979. He received the Ph.D. degree in economics from Liaoning University, in 2016. He is currently an Associate Professor with Liaoning University. His research interests include digital economy, big data management, distributed data management, and uncertain data management.

**WEI GUO** was born in China, in 1983. She received the B.S., M.S., and Ph.D. degree from the Northeast University, Shenyang, Liaoning, China, in 2006, 2008, and 2011, respectively. Since her graduation, she has been worked at Shenyang Aerospace University, China, where she is currently an Associated Professor. Her research interests include artificial intelligence, machine learning, and image processing.

* * *