AN APPROACH TO COMPARING CONTROL FLOW GRAPHS BASED ON BASIC BLOCK MATCHING

Hyun-il Lim
Dept. of Computer Engineering, Kyungnam University,
Changwon, Gyeongsangnam-do 51767, South Korea
hilim@kyungnam.ac.kr

Abstract - The control flow graph is widely used data structure to understand the characteristics of software. The comparisons of control flow graphs is effectively applied in areas of software development and analysis, such as code optimization, common module or software plagiarism detection and malware detection. In this paper, we present an effective method for identifying the similarity of control flow graphs of software. We find a match of similar basic blocks between control flow graphs concerning the syntactic information in basic blocks and their control flow edges. From the match result, we identify the similarity between two control flow graphs. We evaluate the proposed method with Java programs that have different execution structures. In the experimental results, we can confirm that the proposed matching and identifying method can be effectively applied in analyzing the similarity of the structures of control flow graphs of software.

***Keywords*:** Control flow analysis; Control flow graph; Graph comparison; Software analysis.

1. Introduction

A control flow graph is a basic and important data structure that can show static structures and characteristics of programs. Besides, it can be applied in various areas, such as code optimization, detection of similar codes, or plagiarized codes. For example, a comparison of control flow graphs can be used as a preliminary method for identifying similarity between programs. Recently, the control flow graph is applied in detecting malicious software through the matching of the structures [1,2,3,4]. So, the comparison methods of control flow graphs are widely applied in several areas of applications and software analysis.

The control flow graph of a program is a primitive data structure that can effectively describe the structural characteristics of a program utilizing basic blocks with instructions and control flows between the blocks in the program. Because the control flow graph can show various aspects of the software's structural characteristics, it is widely used in static or dynamic software analysis to figure out the features of the software. So, the control flow graph has been applied to identify and understand software in various application areas, such as software plagiarism detection or malware detection. The effective representation of control flow graph can also be used as data in various application areas, such as big data and machine learning.

In this paper, we present a matching method for control flow graphs of software to identify the similarity between software. The method can accomplish the goal through an analysis of static structures of basic blocks and control flow in the software. To compare and match two different control flow graphs of programs, we propose a method for analyzing and comparing features of basic blocks and control flow edge. For a successful matching of control flow graphs, we consider instructions and syntactic information contained in basic blocks of control flow graphs to effectively distinguish the structures and characteristics of programs. Besides, we also consider control flows that are connected with incoming or outgoing edges between basic blocks. To identify the structural similarity between control flow graphs of software, we propose a method for finding a match between basic blocks of two control flow graphs according to the analyzed information. From the match of basic blocks, we can identify similarity according to the matching information of control flow graphs of software. To evaluate the effectiveness of the proposed method, we experiment on Java programs that have two different execution structures, which are iterative and recursive versions of programs. From the experimental results, we confirm that the proposed matching method can identify the similarity of control flow graphs with high reliability.

This paper is organized as follows. In section 2, we summarize previous related work on comparing control flow graphs. In section 3, we present a method for comparing and identifying the similarity of control flow graphs of software. In section 4, we describe the experimental environment and show experimental results with the proposed method. In section 5, we describe several discussions to improve the proposed method and describe future work. Finally, we conclude this paper in section 6.

2. Related Work

A graph is a widely used data structure for storing and managing various types of data. It also can be used to represent the structures of data in various ways. So, in comparing the features of data that are represented as a form of graph data structure, it is required to compare the characteristics of the graph data structures.

In comparing graph data structures, the specific features of the graphs structures should be considered for effective results. Because the labels of graphs nodes can denote important features for representing the specific characteristics of data, graph comparison methods can be classified into two approaches according to considerations of the labels on nodes. If the nodes of graphs have no labels on, the nodes themselves are considered as having identical characteristics. So, the nodes should be compared with the edges connecting between nodes. On the other hand, if the nodes of graphs have distinguishable labels, the labels on nodes can be important information for identifying or distinguishing the similarity of the structures of graphs. So, the two different types of graph data structures need to be considered when comparing the structural characteristics of graphs.

Because the graph is a basic data structure, there have been various approaches to compare the structures of graphs. In comparing general graph structures, the maximum common subgraph algorithm is used to solve the subgraph isomorphism problem [5]. In the approach, however, the problem of finding maximum common subgraphs between graphs is an NP-Hard problem. So, the approach is inefficient for practical applications. To mitigate such an efficiency problem, there have been several approaches to approximation methods [6,7] to efficiently solve maximum common subgraphs although the approaches may not find the optimal solutions. Besides, there have been various researches on comparing graph data structures, such as graph edit distances [8] or statistical similarity [9].

Unlike comparing general structures of graph data, comparisons of control flow graphs can be more effective if the characteristics of control flow graphs for representing the structures of software are considered. In comparing control flow graphs of different software versions, dynamic matching [10] has been applied by using execution histories to produce mappings between instructions in the two versions of control flow graphs. In the other approach, the dynamic matching algorithm for control flow graphs was applied to function call graphs for effective matching between control flow graphs to get reliable results [11]. However, the matching requires additional dynamic information that can be collected from the execution of software. So, these approaches require additional execution environments for programs to compare control flow graphs and need extra works to make comparisons. In this paper, we propose an effective method for comparing the similarity of the structures of control flow graphs of software in a static way.

3. Comparing Control Flow Graphs

3.1. *Control Flow Graph*

The control flow graph [12] of software is widely used in software analysis to represent the static structures of programs. Generally, a control flow graph is defined as a graph $G = (N, E)$, where N and E denote the set of nodes and edges in a graph G, respectively. Each node of a control flow graph represents a basic block of a program, and it contains syntactic information or instructions that are executed during program execution. Each edge of a control flow graph represents a control flow between basic blocks and it shows a possible execution flows between basic blocks during program execution.

Control flow graphs are useful in various software analysis approaches because they are typical data structures that can statically express the syntactic features of the software. Besides, they can show possible execution flows of software execution that can abstract runtime features of the software. So, the control flow graph can also be used as analysis data in several areas of data processing, such as big data or machine learning approaches on software.

3.2. *Main Idea*

The control flow graph of software is a special form of graph data structure because it is represented with sets of nodes and edges, those are basic blocks and control flows, respectively. So, it can be compared through general graph comparison methods. Because such a typical graph comparison method does not reflect the specific characteristics of control flow graphs of software, it is inappropriate to directly apply the methods to control flow graphs.

As a feature of a control flow graph, basic blocks of a control flow graph include important information of software, such as instruction sequences and syntactic structures of software. If such information is used to distinguish or identify similar basic blocks, it can be helpful to achieve better results. So, in this paper, we employ the information of basic blocks and control flows as key features for identifying similar basic blocks between control flow graphs. In the previous approaches, labels on graph nodes are used only for finding or matching nodes with identical labels between graphs. In the approaches, comparisons of nodes, which contain partially similar contents, do not properly reflect partial similarities in comparing control flow graphs. So, directly applying the previous method to control flow graphs of software may have limitations in achieving good results.

The main idea of this paper is as follows. At first, partially similar basic blocks between control flow graphs are identified according to the characteristics of basic blocks. The pairs of highly similar basic blocks are identified by measuring similarities between basic blocks by considering instructions and syntactic information of basic blocks and their control flow edges. After that, the pairs of similar basic blocks are matched to each other to construct a similarity relationship between two control flow graphs. Finally, we identify the similarity of structures between control flow graphs by analyzing the conformity of the match between control flow graphs.

3.3. *Comparing Features of Basic Blocks*

In the first step, we consider instructions and syntactic information contained in basic blocks to find pairs of similar basic blocks between control flow graphs. At this step, the features of basic blocks are described as sets of syntactic information and instructions, and they are compared to calculate the similarity between basic blocks. The similarity is measured by calculating the number and ratio of common elements between basic blocks.

For example, let $\{P_1, P_2, \cdots, P_m\}$ and $\{Q_1, Q_2, \cdots, Q_n\}$ be the set of basic blocks in two control flow graphs, respectively. Then, the similarity between the *i*-th and the *j*-th basic blocks P_i and Q_j of the control flow graphs, $BBlock(P_i, Q_j)$, is measured as the following equation:

$$BBlock(P_i, Q_j) = \frac{|B(P_i) \cap B(Q_j)|}{|B(P_i) \cup B(Q_j)|}, \qquad (1)$$

where $B(P_i)$ denotes the set of instructions and syntactic information contained in the basic block P_i, and $|A|$ denotes the number of elements in the set A.

The similarity of basic blocks is measured by calculating the number and ratio of common instructions. Intuitively, the more same contents between basic block pairs, the higher the similarity values. The resulting values are used to identify pairs of similar basic blocks and find a match of basic blocks between control flow graphs.

We also consider the characteristics of edges, which denote control flows between the basic blocks, because the control flow information can provide syntactic information for distinguishing basic blocks of a control flow graph. Similarly as comparing basic blocks themselves, we measure similarities between incoming and outgoing edges, respectively. In measuring the similarity of outgoing edges of two basic blocks, we calculate the similarity according to features of basic blocks that are followed by outgoing edges from the two basic blocks.

In matching similar basic blocks of control flow graphs, it is important to consider the relationship with their neighbors of the matching basic blocks. The relationships of incoming and outgoing control flows of basic blocks can reflect the status of the basic block within the overall structure of a control flow graph. So, considering the control flows of basic blocks can be an effective approach for finding a proper match of similar basic blocks.

For example, for two basic blocks P_i and Q_j, the similarity of outgoing control flows of the basic blocks P_i and Q_j, $BBlock_{out}(P_i, Q_j)$, is measured as follows:

$$BBlock_{out}(P_i, Q_j) = \frac{|\bigcup_{X \in OUT(P_i)} B(X) \cap \bigcup_{Y \in OUT(Q_j)} B(Y)|}{|\bigcup_{X \in OUT(P_i)} B(X) \cup \bigcup_{Y \in OUT(Q_j)} B(Y)|}, \qquad (2)$$

where $OUT(P)$ denotes the set of basic blocks reachable through outgoing edges from the basic block *P*.

The similarity value is calculated by the number and ratio of common instructions or syntactic information that is executed after the basic blocks through outgoing edges. Intuitively, the more common instructions or syntactic information after executing the basic block, the higher the similarity values. These figures can stabilize a match of similar basic blocks because the values can reflect the syntactic structures adjacent to the basic blocks in control flow graphs.

Similarly, the similarity of incoming control flows of basic blocks P_i and Q_j, $BBlock_{in}(P_i, Q_j)$, is measured as follows:

$$BBlock_{in}(P_i, Q_j) = \frac{|\bigcup_{X \in IN(P_i)} B(X) \cap \bigcup_{Y \in IN(Q_j)} B(Y)|}{|\bigcup_{X \in IN(P_i)} B(X) \cup \bigcup_{Y \in IN(Q_j)} B(Y)|}, \qquad (3)$$

where $IN(P)$ denotes the set of basic blocks reachable to the basic block *P* through incoming edges of the basic block *P*.

When we compare basic blocks, $BBlock(P_i, Q_j)$ provides a measure for calculating a similarity between basic blocks themselves. Besides, $BBlock_{out}(P_i, Q_j)$ and $BBlock_{in}(P_i, Q_j)$ describe the similarity of control flows of the basic blocks in the comparison. By combining the three measures in comparing basic blocks, we can reflect the syntactic structures and the execution order of basic blocks in overall structures of control flows as well as instructions contained in the basic blocks themselves. Therefore, we consider the three measures together to effectively find a stable match of similar basic blocks between control flow graphs.

To balance the three measures, the matching similarity of basic blocks P_i and Q_j, $S(P_i, Q_j)$ is calculated as follows:

$$S(P_i, Q_j) = \frac{2 \times BBlock(P_i, Q_j) + BBlock_{out}(P_i, Q_j) + BBlock_{in}(P_i, Q_j)}{4}. \qquad (4)$$

This value represents the similarity of basic blocks of control flow graphs by combining three similarity measures of the basic blocks and their control flows. The similarity value reaches 1.0 if both the basic blocks and their control flows are identical. On the other hand, if both basic blocks and their control flows are different from each other, the similarity will be the value of 0.0. So, the value of $S(P_i, Q_j)$ remains between 0.0 and 1.0 according to the degree of similarity between basic blocks. This value is used to find a match of similar basic blocks for identifying the similarity between two control flow graphs.

In this approach, every pair of basic blocks between two control flow graphs should be compared to get their similarities for finding a match of similar basic blocks between the two control flow graphs. In other words, for two control flow graphs, which have *m* and *n* basic blocks, respectively, the similarities of $m \times n$ pairs of basic blocks between the control flow graphs should be calculated. At the next step, the similarity values are used to match similar basic blocks to identify the similarity of control flow graphs.

3.4. *An Example of Comparing Basic Blocks*

Figure 1 shows an example of basic blocks P and Q from two different control flow graphs. The two basic blocks have control flow edges connecting to other basic blocks through incoming and outgoing edges. In the example, the characters written on basic blocks denote instructions or syntactic information contained in basic blocks.

In the example, the basic block *P* and *Q* have five instructions $\{a,b,c,f,i\}$ and $\{a,b,c,e,f\}$, respectively. From Eq. (1), the similarity between the two basic blocks, $BBlock(P, Q)$, is calculated as follows:

$$BBlock(P_i, Q_j) = \frac{|\{a,b,c,f,i\} \cap \{a,b,c,e,f\}|}{|\{a,b,c,f,i\} \cup \{a,b,c,e,f\}|} = \frac{|\{a,b,c,f\}|}{|\{a,b,c,e,f,i\}|} = \frac{4}{6}.$$

From Eq. (2) and (3), the similarities of the control flows, $BBlock_{out}(P, Q)$ and $BBlock_{in}(P, Q)$, are calculated as follows:

$$BBlock_{out}(P, Q) = \frac{|\{a,b,c,e,f,g,h\} \cap \{a,b,d,f,g,i\}|}{|\{a,b,c,e,f,g,h\} \cup \{a,b,d,f,g,i\}|} = \frac{|\{a,b,f,g\}|}{|\{a,b,c,d,e,f,g,h,i\}|} = \frac{4}{9}.$$

$$BBlock_{in}(P, Q) = \frac{|\{a,b\} \cap \{a,b,c,d\}|}{|\{a,b\} \cup \{a,b,c,d\}|} = \frac{|\{a,b\}|}{|\{a,b,c,d\}|} = \frac{2}{4}.$$

From Eq. (4), the matching similarity between the two basic blocks P and Q, $S(P, Q)$, is as follows:

$$S(P, Q) = \frac{2 \times BBlock(P, Q) + BBlock_{out}(P, Q) + BBlock_{in}(P, Q)}{4} = \frac{2 \times \frac{4}{6} + \frac{4}{9} + \frac{2}{4}}{4} = 0.569.$$

So, the matching similarity between the basic blocks *P* and *Q* shown in Figure 1 is about 0.569. As this example, the similarity for every pair of basic blocks between two control flow graphs is calculated to find a match of similar basic blocks between control flow graphs.

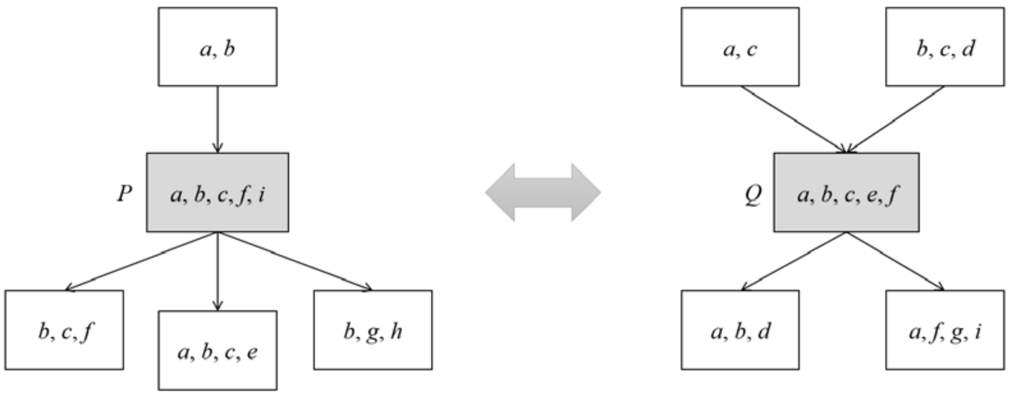


Fig. 1. An Example of Basic Blocks *P* and *Q* with their adjacent basic blocks.

3.5. *Matching and Identifying Similarity of Control Flow Graphs*

Let *m* and *n* be the numbers of basic blocks in two control flow graphs *A* and *B*, respectively. Then, among the $m \times n$ similarities of basic blocks between control flow graphs, the pairs of similar basic blocks are matched in descending order of matching similarity S. So, a match between control flow graphs *A* and *B*, $Match(A, B)$, is a set of $\min(m, n)$ pairs of similar basic blocks between the control flow graphs. The overall similarity of control flow graphs is calculated by summing the matching similarities of the matched pairs of basic blocks. The similarity between the control flow graphs *A* and *B*, $Sim(A, B)$, can be calculated as follows:

$$Sim(A,B) = \frac{\sum_{(P_i,Q_j)\in Match(A,B)} S(P_i,Q_j)}{\min(m,n)}, \qquad (5)$$

where $Match(A, B)$ denotes a set of matched pairs of basic blocks between *A* and *B*. To calculate an overall similarity, we sum up similarities of the matched pairs of basic blocks. And then, we divide the value with the number of matched pairs to normalize the similarity. So, the resulting value is located between 0 and 1 according to the degree of similarity of the structures of control flow graphs.

Intuitively, the set $Match(A, B)$ contains the pairs of basic blocks, and it represents how many basic blocks are similar in terms of the structural features of basic blocks and their control flows. If control flow graphs have many similar structures, they can share many more pairs of similar basic blocks with high matching similarity. So, the similarity can effectively reflect the structural characteristics of control flow graphs.

3.6. *An Example of Identifying Similarity of Control Flow Graphs*

Let $\{P_1, P_2, P_3, P_4, P_5\}$ and $\{Q_1, Q_2, Q_3, Q_4\}$ be the set of basic blocks of two control flow graphs *A* and *B*, respectively. The matching similarities of basic block pairs between the control flow graphs are calculated by the equations presented in the previous section. For example, Table 1 shows an example of the matching similarities of basic block pairs between the control flow graphs.

Table 1. An example of matching similarities of basic blocks between control flow graphs.

	P_1	P_2	P_3	P_4	P_5
Q_1	**0.83**	0.74	0.38	0.18	0.36
Q_2	0.24	**0.69**	0.53	0.39	0.61
Q_3	0.54	0.28	0.73	0.43	**0.88**
Q_4	0.17	0.39	**0.92**	0.57	0.47

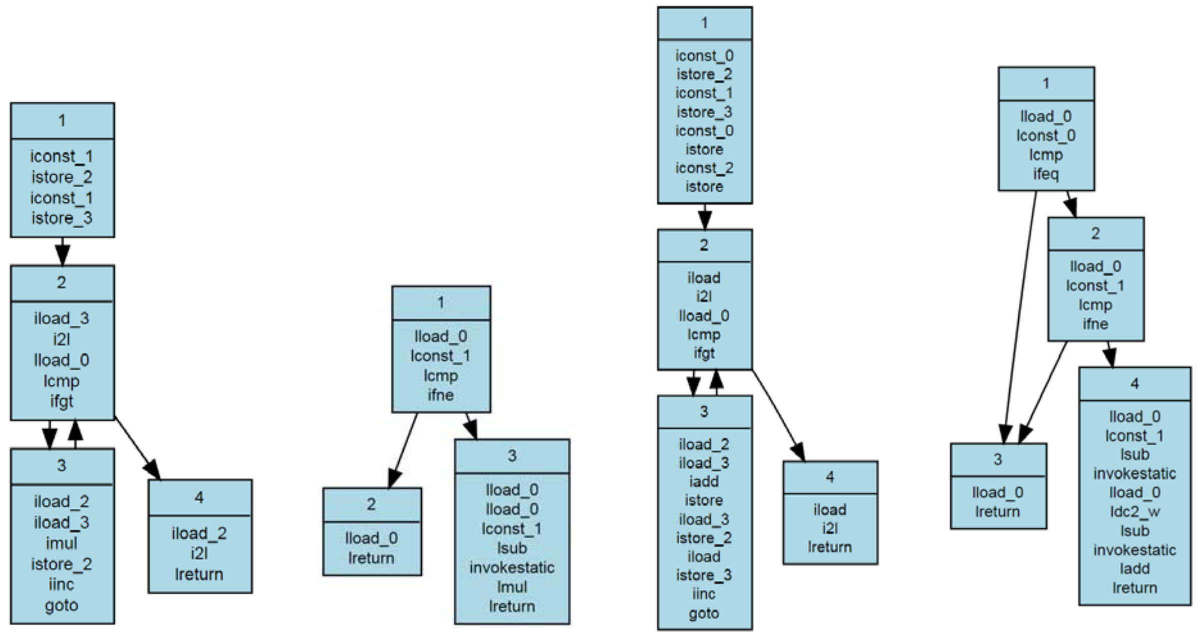
The procedure for finding a match and calculating the similarity between control flow graphs is as follows. At first, similar basic blocks are one-to-one matched in descending order of matching similarity. For example in Table 1, the pair of basic blocks (P_3, Q_4) has the highest matching similarity with 0.92. So, the pair is firstly matched. After that, among the remaining basic blocks, the pair with the highest similarity is (P_5, Q_3), and its matching similarity is 0.88. Similarly, the next matching pair of basic blocks is (P_1, Q_1), and its similarity is 0.83. Finally, P_2 and Q_2 are matched and its similarity value is 0.69. So, the match set of basic blocks between the control flow graphs A and B, $Match(A, B)$ is {(P_3, Q_4), (P_5, Q_3), (P_1, Q_1), (P_2, Q_2)}. From Eq., the similarity between control flow graphs is calculated as follows:

$$Sim(A,B) = \frac{\sum_{(P_i,Q_j)\in Match(A,B)} S(P_i,Q_j)}{\min(m,n)} = \frac{0.92+0.88+0.83+0.69}{\min(5,4)} = \frac{3.32}{4} = 0.83 .$$

Therefore, the similarity between the two control flow graphs is measured as 0.83. From the high similarity, we can confirm that the two control flow graphs are analyzed to have common similar structures.

4. Experimental Results

In this paper, we have proposed a method for identifying the similarity of control flow graphs by finding a match of similar pairs of basic blocks. The proposed method can be applied to compare control flow graphs of software or binary programs.



(a) The factorial of number with iteration. (b) The factorial of number with recursion. (c) The Fibonacci numbers with iteration. (d) The Fibonacci numbers with recursion.

Fig. 2. The control flow graphs of the factorial and Fibonacci numbers with iteration or recursion.

To evaluate the effectiveness of the proposed approach, we have implemented the proposed method in Python 2.7 [13] and experimented with four Java programs. The four Java programs contain different execution structures, and they are iterative and recursive program versions for calculating factorial and Fibonacci numbers, respectively. In the experiment, we first extracted Java bytecode from class files compiled from the four Java programs. Then, we constructed control flow graphs from the extracted Java byte code of the four benchmark programs. Finally, we applied the proposed method to identify similarities of structures between control flow graphs of the benchmark programs.

Figure 2 shows the control flow graphs constructed from the two different program versions for calculating factorial and Fibonacci numbers, respectively. In the iterative program versions (a) and (c), we can check backward control flow edges for going to previous basic blocks to iterate instruction sequences of the programs. On the other hand, in the recursive program versions (b) and (d), there is no backward control flow edges in the control flow graphs while the programs have recursive calls to calculate the numbers.

Table 2 shows the results of the similarities between the four control flow graphs. The similarities are calculated after finding match sets of basic block pairs as described in the previous section. In comparing identical control flow graphs, all the similarities were 1.0, because each basic block was perfectly matched to its identical basic block with the similarity value of 1.0. In comparing different control flow graphs, comparisons of control flow graphs that have the same execution structures had higher similarities than the others. In this experiment, we compared the two different execution structures of iterative and recursive versions.

The similarity between the recursive versions of the factorial and Fibonacci numbers was 0.77, and it showed that the two control flow graphs had a high structural similarity. The similarity between the iterative versions of the factorial and Fibonacci numbers was 0.58, and the value was higher than the reference point of 50%. On the other hand, in comparing control flow graphs that have different execution structures, the similarities were lower than 50%. The similarity between the iterative and the recursive version for calculating factorial numbers was 0.25. The similarity between the Fibonacci number with iteration and recursion was 0.21, which was the lowest similarity among all comparisons of control flow graphs.

Table 2. The results of similarities between the control flow graphs with iteration and recursion.

	Factorial (iteration)	Factorial (recursion)	Fibonacci (iteration)	Fibonacci (recursion)
Factorial (iteration)	1.00	0.25	0.58	0.21
Factorial (recursion)		1.00	0.25	0.77
Fibonacci (iteration)			1.00	0.21
Fibonacci (recursion)				1.00

From the experimental results, the comparisons of control flow graphs with similar execution structures had high similarity values. On the other hand, the comparisons of control flow graphs with different execution structures had a low similarity, although they performed identical tasks. From the evaluation result, we confirm that the proposed method can effectively identify the similarity between control flow graphs.

5. Discussion and Future Work

In this paper, we proposed a method for identifying the similarity of control graphs through matching similar basic blocks according to the characteristics of basic blocks. To improve the accuracy of the proposed method, we investigated the evaluation results, and we have the following future work.

At first, in matching similar basic blocks between control flow graphs, we have considered the features of the basic block as well as incoming and outgoing edges. In deciding the match of basic blocks, it is important to balance the three factors for effectively representing the similarity of control flow graphs. In the proposed approach, we applied the ratio of the basic block itself and control flow edges as one to one. Because each basic block has generally more than one control flow edges, the ratio may not be an optimal one for describing the structural characteristics of control flow graphs. So, in future work, we plan to perform experiments to customize the ratio of basic block and control flow edges to find a stable match for identifying structural similarity.

Secondly, we have performed experiments with four Java benchmark programs to evaluate the proposed method. Although the programs have different execution structures, the experimental results were limited to small benchmark programs. To improve the reliability and accuracy of the proposed method, we plan to perform experiments with a benchmark set of large programs.

6. Conclusion

The control flow graph of software is widely used in static and dynamic software analyses because it can effectively represent the static or dynamic features and the structures of software. In this paper, we have presented an effective method for identifying the similarity of control flow graphs of software. The proposed method compares and matches the characteristics of basic blocks and control flow edges between basic blocks with considering the structures of control flow graphs. The similarity is calculated by finding a match of similar basic blocks between control flow graphs. The proposed method was evaluated with four Java benchmark programs, which had two different execution structures. In the experimental results, the similarities of control flow graphs with the same execution structures had higher similarities than those with different execution structures. With the evaluation, the proposed method is evaluated to be an effective method for identifying the similarity of control flow graphs.

Recently, analysis of control flow graphs is applied in various areas, such as code optimization, detecting similar or plagiarized code, and detecting malicious code. The proposed method can also be applied to various data analysis applications to analyze and understand the characteristics of software.

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Education) (No. NRF-2017R1D1A1B03034769).

References

[1] Bruschi, D.; Martignoni, L.; Monga, M. (2006): Detecting Self-Mutating Malware Using Control-Flow Graph Matching, In Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2006), LNCS 4064, pages 129-143.
[2] A. Kapoor, S. Dhavale, (2016): Control Flow Graph Based Multiclass Malware Detection Using Bi-normal Separation, in Defence Science Journal, Vol. 66. No. 2, pp. 138-145, 2016.
[3] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, D. Nyang, and A. Mohaisen, (2019): Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach, in Proceedings of NDSS Symposium, San Diego, California.
[4] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, (2019): Analyzing and Detecting Emerging Internet of Things Malware: A Graph-Based Approach, in IEEE Internet of Things Journal, Vol. 6, No. 5, pp. 8977-8988.
[5] Raymond, J. W.; Gardiner, E. J.; Willett, P. (2002): RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs, The Computer Journal Vol. 45, No. 6, pp. 631-644.
[6] Raymond, J. W.; Willett, P. (2002): Maximum common subgraph isomorphism algorithms for the matching of chemical structures, Journal of Computer-Aided Molecular Design, 16: 521–533.
[7] Abu-Khzam, F. N.; Samatova, N. F.; Rizk, M. A.; Langston, M. A. (2007): The Maximum Common Subgraph Problem: Faster Solutions via Vertex Cover, In IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007), pp. 367-373.
[8] Zeng, Z.; Tung, A. K. H.; Wang, J.; Feng, J.; Zhou, L. (2009): Comparing Stars: On Approximating Graph Edit Distance, International Conference on Very Large Data Bases (VLDB 2009).
[9] Zager, L. (2005): Graph similarity and matching, MS Thesis, Dept of EECS, MIT.
[10] Zhang, X. and Gupta, R. (2005): Matching Execution Histories of Program Versions, 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 197-206, 2005.
[11] Nagarajan, V.; Gupta, R; Zhang, X.; Madou, M., Sutter, B. D. (2007): Matching Control Flow of Program Versions, 2007 IEEE International Conference on Software Maintenance.
[12] Wilhelm, R.; Maurer, D. (1995): Compiler Design, Addison-Wesley.
[13] Python Programming Language, http://www.python.org/.