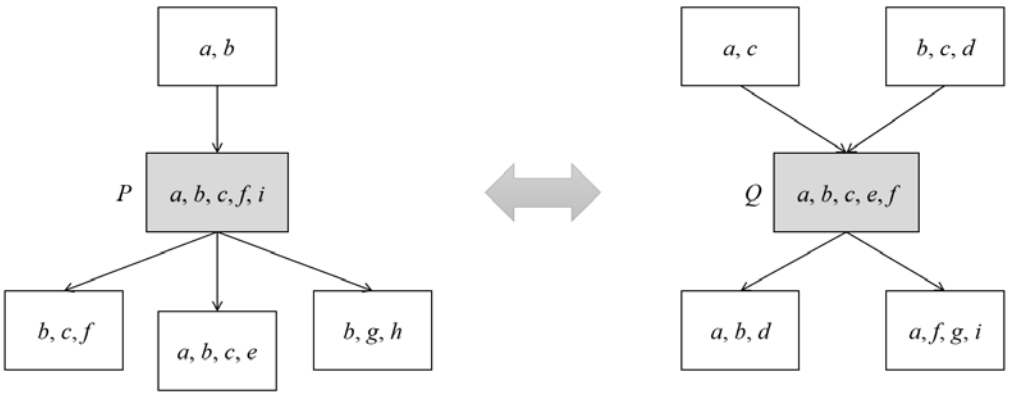


Fig. 1. An Example of Basic Blocks *P* and *Q* with their adjacent basic blocks.

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

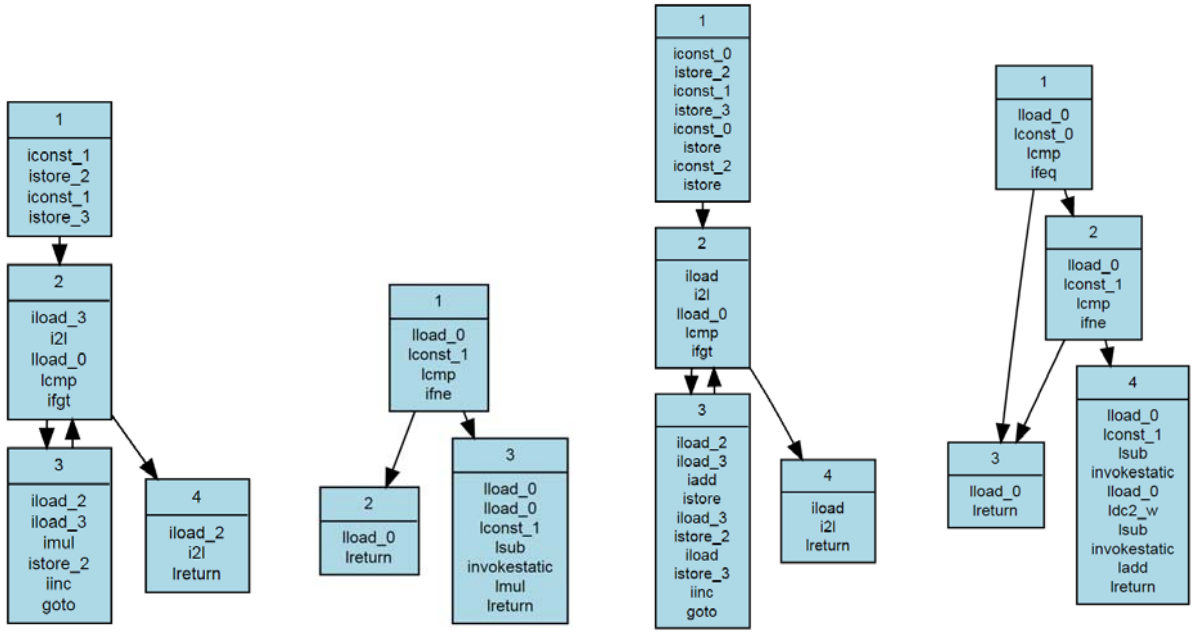
The procedure for finding a match and calculating the similarity between control flow graphs is as follows. At first, similar basic blocks are one-to-one matched in descending order of matching similarity. For example in Table 1, the pair of basic blocks (P_3, Q_4) has the highest matching similarity 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/.