An Average Case Time Complexity Estimator for Black-box Functions

Duncan Yung, Bill Laboon, Shikuo Chang

Department of Computer Science, University of Pittsburgh
duncanyung@cs.pitt.edu,bill.laboon@pitt.edu,chang@cs.pitt.edu

***Abstract*— Average case time complexity is widely used to evaluate the efficiency of an algorithm [2]. Given a black-box function, if a tester wants to know the average case time complexity, he/she has to analyze the source code and make input assumption so as to know the average case of the function. Although that is feasible, it is time consuming and that makes the testing process no longer automatic. In this paper, we propose an approach for estimating the average case time complexity of a given black-box function without analyzing source codes. Experimental results show that our approach can accurately estimate the average case time complexity without reading the source code.**

I. INTRODUCTION

The worst case time complexity is always used to evaluate the efficiency of an algorithm. The worst case time complexity of an algorithm is based on an extreme input which maximize the execution time of the algorithm. However, such extreme input may not always appear. Hence, average case time complexity can be a better representation of efficiency of an algorithm [2].

Given a black-box function, if a tester wants to know the average case time complexity, he/she has to analyze the source code and make input assumption so as to know the average case of the function. Although that is feasible, it is time consuming and that makes the testing process no longer automatic. Furthermore, manual source code analysis is not always reliable as it is not rare that human error occurs (That is one of the reason to have software testing.).

In this paper, we propose an approach for estimating the average case time complexity of a given black-box function without analyzing source codes. It is a useful tool for testers and programmers to analyze their source codes. The challenges of building such tools lie in two-fold:

1) Without reading the source code, we cannot get any hint from the implementation. The only easy thing that we can do is to measure execution time of the black-box function.
2) It is too time consuming to measure the execution time of all possible inputs so as to obtain the average execution time of the function and estimate the average case time complexity.

Related work study is in Section II. In Section III-A, we propose the baseline approach. Based on the baseline approach, we develop an advanced approach in Section III-C. The accuracy and efficiency of different approaches are evaluated in Section IV.

II. RELATED WORK

The related work of time complexity analysis mainly fall into two areas-static time complexity analysis and measurement-based time complexity analysis. However, they all focus on worst-case time complexity. In this paper, we focus on average case time complexity.

A. Static Time Complexity Analysis

Static time complexity analysis approaches[9], [7] derive bounds for the execution time of a program without actually executing the program. Usually, the program is not treated as a black-box function and analysis of source codes is needed. In this paper, we propose a approach that can estimate the time complexity of black-box functions.

B. Measurement-based Time Complexity Analysis

Measurement-based approaches measures execution time of programs. As the number of possible inputs increase with the complexity of the program, exhaustive measurements becomes impossible. There exists solutions (e.g. [6] and [8]) that partition a program into parts for measuring execution time of worst case. These approaches try to bring the system into a worst-case state before taking measurements, e.g. by clearing the cache. However, this assumption may not hold for complex processor architectures that can exhibit timing anomalies. Kirner et al [8] proposed to generate inputs so that all paths of the program are taken. However, Kirner et al's approach is not a purely measurement-based time complexity analysis approach. Bernat et al [4], [3] proposed to determine the probability distribution of execution time. Their approach does not derive a bound for the execution time.

III. METHODOLOGY

In this paper, we propose to use execution time of the black-box function (Section III-A.1) and regression to estimate the average case time complexity (Section III-A.2) of a black-box function.

(DOI reference number: 10.18293/SEKE2015-213)

A. Baseline Approach

In this paper, we assume that the average case time complexity of a function is analyzed based on the uniform input assumption (Definition 1). That is the probability of the appearance of an input is always uniformly distributed over all possible inputs. For example, if the input is a size 3 integer array and only 1, 2, and 3 (domain size=3) are valid values for each element of the array, the probabilities that {1,1,1},{1,1,2},{1,1,3},...,{3,3,3} (there are 27 possible inputs) will be the input are the same and sum of the probabilities is 1.

Definition 1: **[Uniform Input]** Let x be an input. x satisfies: $\forall x \in \{0,1\}^*$, $prob(input = x) = \frac{1}{|\{0,1\}^*|}$, where $\{0,1\}^*$ is all possible bit patterns and $|\{0,1\}^*|$ is the size of the set $\{0,1\}^*$.

1) Data Collection Phase: Based on the uniform input assumption, we estimate the average execution time of an input size by measuring the average execution time of all possible inputs. Theoretically, we can generate all possible inputs and estimate the average execution time for n=1,...,∞. By doing that, we can obtain data point of the average execution time of all input sizes.

2) Prediction Phase: Non-linear regression can be used to fit different curves to the data points. The best fit curve is the one with the least means square error. We adopt the best fit curve to be the average case time complexity of the function. For example, we use non-linear regression to fit the data points to functions $a+bn, a+blog_2n, a+bnlog_2n, a+bn^2, a+bn^3$, and $a+bn^4$, where n is the input size, and a and b are coefficients. For each function, we can obtain the mean square error. The function with the least mean square error is chosen to be the average case time complexity estimation of the black-box function.

B. Bottleneck in Data Collection Phase

The number of all possible inputs increases exponentially with with input size. For example, the number of all possible inputs of a size 10 integer array is 10^{10}, given that the value of each element of the array can only be 1,2,...,10. Therefore, it is impossible to measure the execution time when input size is large. Although it is still possible to measure execution time of a function when input size is small (e.g. 5), the estimation of the average case time complexity is not accurate. (Figure 1, 2, and 3, Sol_3).

C. Sampling Approach with Majority Voting

In this paper, we propose to use uniform sampling of input to represent all possible inputs so as to improve efficiency of the model and majority voting technique to reduce variance of the estimation result.

1) Efficient Sampling: For each input size n, we draw samples uniformly from all possible inputs. The samples can be used as an estimation of all possible inputs as the probability of each input being drawn is the same.

2) Majority Vote: For each run, we run the model for k times, where k is a user-defined parameter. Then, we pick the majority prediction result as the result of that run. Suppose k is 10. There 6 times the prediction result is $a+bnlog_2n$ and 4 times the prediction result is $a+bn^2$. Then, the prediction result of that run is $a+bnlog_2n$.

In this setting, the majority voting is the same as bagging in machine learning field. It is well-known that bagging can successfully improve stability of models [5].

IV. Experiment

In this section, we evaluate the accuracy and efficiency of seven different approaches using 14 algorithms (black-box functions) which have integer array as inputs. We assume that the domain size of each integer is 1 to maximum integer in Java. Algorithm 1 to 10 are well-known algorithms. The definitions of algorithm 11 to 14 can be found in [1]. Experiments are implemented in Java and run in an Intel Core i5 2.5GHz laptop with 4G memory. All source codes can be found in $https://github.com/duncanyung/cs1699Fall14\_deliverable4.git$. We compare accuracies and execution time of different approaches.

A. Comparison of Different Approaches

The settings of each approach are as below:

1) Sampling Approach with Majority Voting (Sol_1)
   - input size n=1,2,5,10,50,100
   - sample size for each n=5000 (sample size for algorithm 5 is 1000)
   - majority voting of 50 predictions
   - 50 runs

2) Sampling Approach **without** Majority Voting (Sol_2^a)
   - input size n=1,2,5,10,50,100
   - sample size for each **n=5000**
   - 50 runs

3) Sampling Approach **without** Majority Voting (Sol_2^b)
   - input size n=1,2,5,10,50,100
   - sample size for each **n=20000**
   - 50 runs

4) Sampling Approach **without** Majority Voting (Sol_2^c)
   - input size n=1,2,5,10,50,100
   - sample size for each **n=80000**

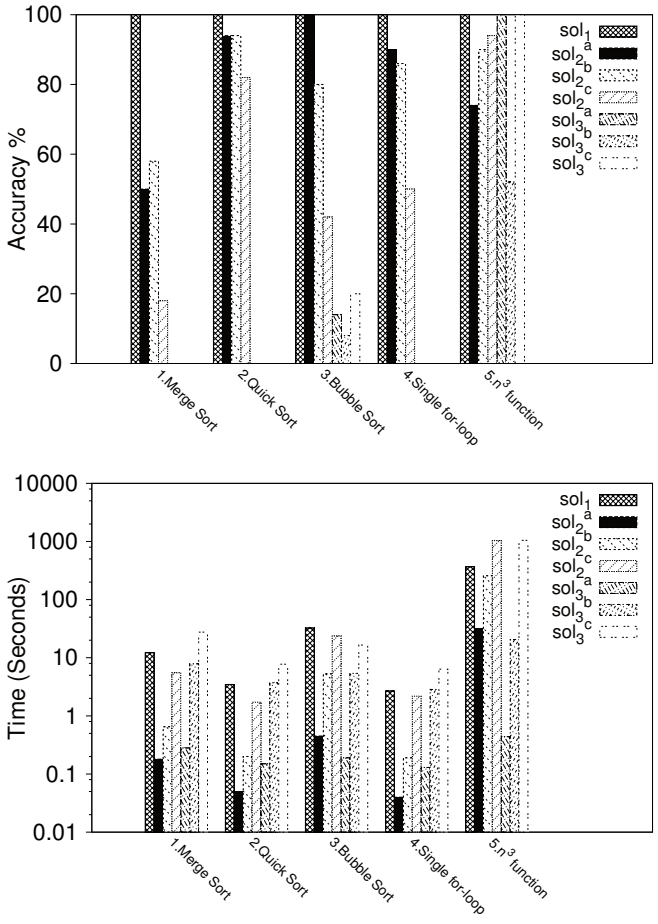

Fig. 1. Algorithm 1 - 5

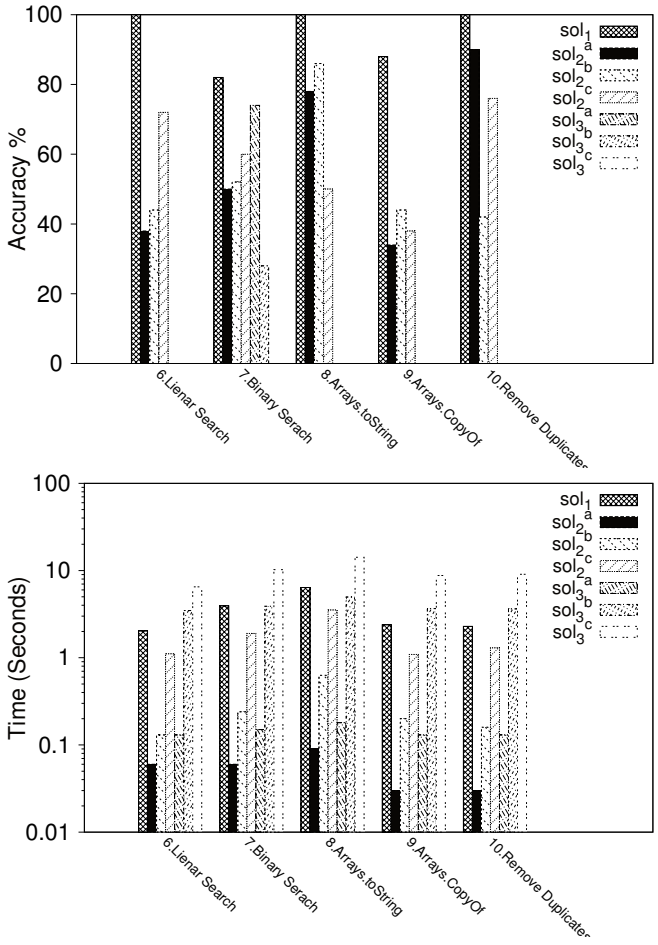

Fig. 2. Algorithm 6 - 10

- 50 runs

5) Baseline Approach **with** Majority Voting (Sol_3^a)
   - input size n=1,2,3,4,5,6,7
   - majority voting of 50 predictions
   - 50 runs

6) Baseline Approach **without** Majority Voting (Sol_3^b)
   - input size n=1,2,3,4,5,6,7,**10**
   - majority voting of 50 predictions
   - 50 runs

7) Baseline Approach **with** Majority Voting (Sol_3^c)
   - input size n=1,2,3,4,5,6,7,**10**
   - majority voting of 50 predictions
   - 50 runs

For each approach, the model tries to classify the average case time complexity of the black-box function as one of these time complexities- $O(n)$, $O(log_2 n)$, $O(nlog_2 n)$, $O(n^2)$, $O(n^3)$, $O(n^4)$, and $O(n^5)$.

Figure 1, 2, and 3 shows the experiment results of all solutions for algorithm 1-5, 6-10, and 11-14 respectively.

The accuracy of Sol_1 is higher than Sol_2^a. Hence, we can see that the majority voting technique can actually improve the accuracy. The accuracy of Sol_3^a is the worst. Hence, we can see that using a small input size (from 1 to 7) cannot help to estimate the complexity. However, increasing input size will make the execution time increase exponentially which is not a feasible solution. Although the execution time of Sol_1 is the highest, the execution time of Sol_1^a is less than 200 seconds (except algorithm 5).

Sol_2^b and Sol_2^c tries to improve the accuracy by increasing the sample size for each input size to 20000 and 80000 respectively. In general, the accuracy of Sol_2^b is better than Sol_2^a, but with higher execution time. Although the accuracy of Sol_2^b is better than Sol_2^a, it is still worse than Sol_1. Then, we further increase the sample size for each input size from 20000 to 80000. Hopefully, the accuracies would be improved. Unfortunately, the accuracy gets worse. Hence, we believe that using large sample size without majority vote cannot help improving accuracy.

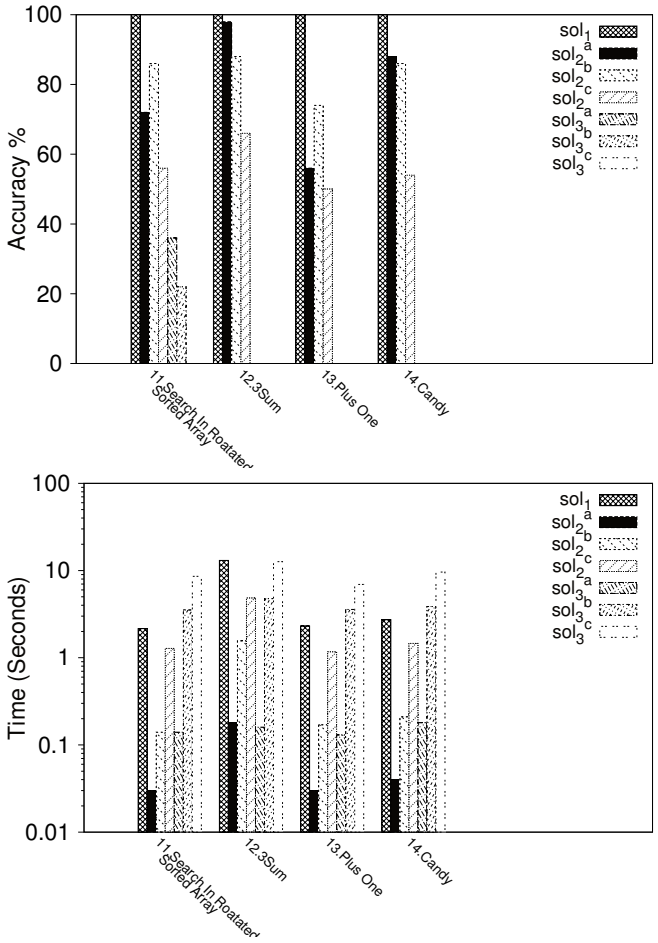

Fig. 3. Algorithm 11 - 14

Sol_3^b tries to improve the accuracy by changing the input size to [1,2,3,4,5,6,7,10] while not using majority vote. Sol_3^c tries to improve the accuracy by changing the input size to [1,2,3,4,5,6,7,10] while using majority vote. The accuracy of Sol_3^b and Sol_3^c are similar to Sol_3^a. However, there is a non-linear increase in average execution time. This shows that slightly increasing the input size (from [1,2,3,4,5,6,7] to [1,2,3,4,5,6,7,10]) cannot help improving accuracy. However, largely increasing the input size (e.g. using [1,2,5,10,50]) would result in an unacceptably high execution time.

V. Discussion

A. Execution Time Issue

The execution time of high complexity algorithms increase non-linearly (e.g. algorithm 5). Under such situation, the model may have to automatically reduce the input and sample size so as to have a quick response. Therefore, the model can estimate the execution time of the algorithm before deciding the input and sample size.

B. Input-Execution Time Relationship

In this paper, we assume that the execution time is related to the input size. However, that is not always true for an arbitrary black-box function. We need another approach for estimating the average case time complexity of those black-box functions. One of the direction to solve this issue is that the system can draw uniform input size sample instead of setting the input size to a specific sequence (e.g. n=1,2,5,10,50,100).

C. Input Type

In the experiment section, we assume that the input is an array of integer. However, the input of a black-box function can be any type. Therefore, generating uniform input for black-box functions with integer array as input is different from generating uniform input for black-box functions with other input types. Given a uniform input generator is the pre-condition of using our avergae case time complexity estimator.

VI. Conclusion

In this paper, we propose an approach to estimate the average case time complexity of a black-box function. We propose to use sampling to improve the efficiency and majority voting to improve the stability of the approach. Experimental result shows our proposed approach (Sol_1) can estimate the average case time complexity of different black-box functions accurately and efficiently.

References

[1] www.leetCode.com.
[2] S. Ben-david, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44:193–219, 1997.
[3] G. Bernat, A. Colin, and S. Petters. pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems. Technical report, 2003.
[4] G. Bernat, A. Colin, and S. M. Petters. Wcet analysis of probabilistic hard real-time system. In *RTSS*, pages 279–288. IEEE Computer Society, 2002.
[5] L. Breiman. Bagging Predictors. *Mach. Learn.*, 24(2):123–140, Aug. 1996.
[6] J.-F. Deverge and I. Puaut. Safe measurement-based WCET estimation. In R. Wilhelm, editor, *5th International Workshop on Worst-Case Execution Time Analysis (WCET'05)*, volume 1 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2007. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
[7] S. Gulwani. Speed: Symbolic complexity bound analysis. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2009.
[8] R. Kirner, P. Puschner, and I. Wenzel. Measurement-based worst-case execution time analysis using automatic test-data generation. In *IN PROC. IEEE WORKSHOP ON SOFTWARE TECH. FOR FUTURE EMBEDDED AND UBIQUITOUS SYSTS. (SEUS05*, pages 7–10, 2004.
[9] R. Wilhelm. Determining bounds on execution times.