Theoretical Computer Science 576 (2015) 45–60

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Complexity of node coverage games ☆

Farn Wang [a,b,c,*], Sven Schewe [d], Jung-Hsuan Wu [a]

[a] Department of Electrical Engineering, National Taiwan University, Taiwan, ROC
[b] Graduate Institute of Electronic Engineering, National Taiwan University, Taiwan, ROC
[c] Research Center for Information Technology Innovation (CITI), Academia Sinica, Taiwan, ROC
[d] Department of Computer Science, University of Liverpool, UK

ARTICLE INFO

Article history:
Received 27 May 2014
Received in revised form 24 January 2015
Accepted 3 February 2015
Available online 7 February 2015
Communicated by V.Th. Paschos

Keywords:
Testing
Nondeterminism
Coverage
Game
Strategy
Complexity

ABSTRACT

Modern software systems may exhibit a nondeterministic behavior due to many unpredictable factors. In this work, we propose the *node coverage game*, a two-player turn-based game played on a finite game graph, as a formalization of the problem to test such systems. Each node in the graph represents a *functional equivalence class* of the *software under test* (*SUT*). One player, the *tester*, wants to maximize the node coverage, measured by the number of nodes visited when exploring the game graphs, while his opponent, the SUT, wants to minimize it. An optimal test would maximize the cover, and it is an interesting problem to find the maximal number of nodes that the tester can guarantee to visit, irrespective of the responses of the SUT. We show that the decision problem of whether the guarantee is less than a given number is NP-complete. We also discuss two extensions of our result and present a testing procedure based on our result when the SUT is not so hostile.

1. Introduction

Coverage-based techniques [29,30] have been widely used in the management of testing projects of large and complex software systems. The idea is to model the *software under test* (*SUT*) as a finite number of *functional equivalence classes* (*FEC*). The number of FECs that a test plan can cover is then used as an indication of completeness of a verification task and quality of the SUT. For white-box testing, typical test criteria include line (statement) coverage, branch coverage, path coverage, dataflow coverage, class coverage, function/method coverage, and state coverage [29,30]. For black-box testing, popular criteria include input domain coverage, GUI event coverage, etc.

Coverage technique is a common part of quality management techniques that are applied in industry for quality control. Full coverage does not normally imply the correctness of the SUT, but empirical studies [17] have shown that coverage-based techniques are effective in detecting software faults. High coverage shows that the test engineers have systematically and methodically sampled behaviors of the SUT to observe. However, most programs are so complicated that, even if all specified items of a program have been tested, the management can obtain high confidence on the quality of the program, but not a correctness proof for the program. For example, even if we have achieved full branch coverage in white-box testing (that is, even if all conditional branches have been covered in the source code execution), usually the majority of all combinations

☆ The work is partially supported by Grant MOST 103-2221-E-002 -150 -MY3 and Project "Performance Testing Techniques That Reflect User Experiences" of Research Center of Information Technology Innovation, Academia Sinica, Taiwan, ROC.

* Corresponding author at: Dept. of Electrical Engineering, National Taiwan University, Nr. 1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, ROC.
E-mail address: farn@ntu.edu.tw (F. Wang).

http://dx.doi.org/10.1016/j.tcs.2015.02.002
0304-3975/

of truth values are still left unobserved. The correctness of a program, e.g., formalized as the problem whether or not the program violates a safety predicate, is undecidable, such that the holy grail of proven correctness is unobtainable.

However, if a project or quality manager sees low coverage, then the manager knows for sure that the behaviors of some parts of the SUT have not been observed. Thus, software project managers use coverage techniques to evaluate the progress of the testing tasks for an SUT and to manage the quality control process, and ultimately to evaluate their confidence in the correctness/quality of the SUT. It is thus important for testers and managers to create test plans that yield high coverage, such that they can gain a high confidence on the quality of the SUT and the testing tasks.

However, the test coverage of an SUT has to be achieved with various assumptions and nondeterministic responses of the SUT [29,31]. For example, when we observe how a request message is served by a server SUT, we really have no control whether the server SUT will finish serving the request, deny the request, or be unaware of the request, e.g., due to loss of connection. The best that a verification engineer can do is to use various strategies to try to reach as many FECs of the server SUT as possible. We propose to model this problem as a two player finite-state game [31], which we call a *node coverage game* (*NC-game* for short). The first player is the tester (*maximizer*; he for short) and the second is the SUT (*minimizer*; she for short). The two players play on a finite *game graph* with nodes for the FECs, where the nodes are partitioned into the nodes that are owned by the tester and the nodes that are owned by the SUT. The tester and the SUT together move a pebble from node to node according to the transition relation of the game graph. The tester chooses the next node when the pebble is on one of his nodes, while the SUT chooses the successor when the pebble is on one of her nodes. The objective of the tester is to maximize the number of visited (covered) nodes, while the SUT wants to minimize the number of covered nodes.

An interesting question about NC-games is how much coverage the tester can guarantee, no matter how the SUT reacts to the test input. We call this guarantee the *maximal coverage guarantee* (*MCG* for short). The MCG is calculated under the conservative assumption that the SUT is malicious and tries to minimize the coverage. (For example, a server SUT may decline all interaction requests.) This is common in nondeterministic systems, and a test coverage below the MCG certainly implies deficiency in test execution.

We show that the MCG decision problem (the problem whether the SUT can resolve the nondeterminism to prevent a cover over a given threshold c) is NP-complete. This complexity is lower than those established for related coverage problems [5,22]. (They are PSPACE-complete, cf. Section 2.) Our proof technique is based on new observations on games and may be itself interesting. Specifically, we observed that although the optimal tester strategies may need be memoryful in achieving MCG, the SUT strategies only need be prefix-consistent which means once the SUT makes an optimal decision at a node in a play, she can stick to that decision in the play. All in all, our result not only may be theoretically interesting, but also may be used as a solid theoretical foundation for testing research.

In the remainder of the paper, we first review related work in Section 2 and basic concepts of game theory in Section 3. In Section 4, we define NC-games. We then establish the lower-bound and upper-bound of the MCG decision problem complexity, which we prove to be NP-complete, respectively in Section 5 and Section 6. In Section 7, we also discuss the complexity results of resettable NC-game decision problems and edge coverage game decision problems. Finally, in Section 8, we discuss how to apply our result to software testing in practice when the SUT is not so hostile.

2. Related work

2.1. Classic problems

One related classic problem is the 2-player reachability game [9,21,25,26] on directed graphs. The game can be turn-based or concurrent. The two players together move a pebble in the game graph. Player 1 wins if and only if eventually a node in a target set is reached. The computation of pure strategies of reachability games is in PTIME.

Another related classic problem is the Canadian traveler problem [22] of Papadimitriou and Yannakakis. A problem instance is a two player game. The players are given a partially observable game graph and do not know which edges are connected. Player 1 may try an edge when he is at the source of the edge. Player 2 then decides whether this edge is connected or not. The connectivity of an edge, once decided, cannot be changed by anyone. The answer to a problem instance is whether the ratio of the traveling distance over the optimal static traveling distance is lower than a given number. The problem is also PSPACE-complete. In contrast, we show that the MCG decision problem is 'only' NP-complete.

Yet another classic problem that could be related to this work is the pursuit-evasion game, sometimes called graph-searching game [15,19,23] on a directed graph with a group of searchers and a group of evaders. A typical version of the problem is played as follows. Initially, each searcher (also each evader) stays in a node. In each round, a searcher (also an evader) can move to another node depending on various restrictions on the velocity. An evader is caught if she and a searcher are in the same node. The decision problem asks whether a given number of searchers is enough to catch all evaders. Important (and particularly simple) variants of these games are cops-and-rober games, which are used to determine the complexity of directed graphs. The most relevant games of this type are the games to determine the DAG width [2,20] and entanglement [3] of a directed graph.

For general pursuit-evasion games, we believe that the mobility and velocity of the evaders does not seem natural for software bugs. Their role would, however, be quite restricted, as even for the simple cops-and-robber games mentioned