Guiding the Search-Based Testing via Dominances vs. Control Dependencies

Ahmed S. Ghiduk
Senior Member of IACSIT,
Faculty of Science, Beni-Suef University, Egypt &
College of Computers & Information Systems, Taif University, Saudi Arabia
asaghiduk@yahoo.com

Abstract

The representation of the problem and the definition of the fitness function are the two key ingredients for the application of search-based optimization to software engineering problems. Therefore, a well-defined fitness function is essential to the effectiveness of search-based testing (SBT). Several search based test-data generation techniques utilized the control dependencies (CD) for guiding the search to find tests. Ghiduk et al. presented a search-based technique that utilizes the dominances to direct the search to generate test data. In this paper, we illustrate the efficiency of dominances in the control-flow graph (CFG) in guiding the SBT. The paper gives some problems for SBT which is guided by the CD. The paper introduces a general form for a fitness function in terms of dominances nodes and postdominances. This function will improve the efficiency of the search consequently; the SBT overcomes the CD problems.

1. Introduction

Search-based optimization techniques (e.g., simulated annealing, genetic algorithms, ant colony and particle swarm) have been applied to a number of software engineering activities such as test-data generation [1]. Genetic algorithms have been the most widely employed search technique in SBT.

However, no matter what search technique is employed, it is the fitness function that differentiates a good solution from a poor one, thereby guiding the search. Thus, a well-designed fitness function is essential to the efficiency of SBT. A lot of search based test-data generation techniques guide the search using the CD to find the test data for satisfying a number of control-flow and data-flow testing criteria. McMinn [4] surveyed the previous work undertaken in this area.

For guiding the SBT, Pargas et al. [2] used the control dependence graph of the tested object. The fitness function is the number of predicates on the executed path that is common with the predicates on a control-dependence path of the target.

To direct the search, Tracey [5] used the formula:

$$\left(\frac{executed}{dependent}\right) \times dist$$

where *dependent* is the number of the control dependence nodes for the target, *executed* is the number of successfully executed control dependent nodes, and *dist* is the branch distance calculation preformed at the branching node.

Wegener et al. [6] modified the Tracey's function by mapping *dist* into the range [0, 1] (called *m_dist*). The fitness function is zero if the target structure is executed, otherwise, the fitness value is:

$$approximation\ level + m\_dist,$$

where *approximation level = (dependent-executed - 1)*.

Wang et al. [7] presented a flattened CFG, a flattened control-dependence graph and a fitness calculation approach for the *switch-case* structure. The formula:

Fitness = approximation level + normalize(dist);

Where $normalize(dist) = 1 - 1.001^{-dist}$ is used to find the fitness value and $dist = |expr\text{-}C|+1$, where *expr* is the value of the expression after the *switch* keyword, and *C* is the constant for the desired *case* branch. When the execution diverges away at other branching node, *dist* is calculated by Tracey's method [5].

Ghiduk et al. [3] presented genetic algorithms based technique, which generates test data to satisfy a wide range of data-flow criteria. The technique applies the concepts of dominance relations between nodes only to define a multi-objective fitness function.

McMinn [4] has discussed the problems of the control-dependencies based fitness functions.

This paper gives some problems of the SBT which is guided by the CD. The paper introduces a general form for a fitness function defined by dominances and postdominances. This function will improve the efficiency of the SBT consequently; SBT overcomes the CD problems.

The rest of the paper is organized as follow: section 2 gives some basic concepts; section 3 gives two of the problems of using CD in SBT and the key ingredients to overcome these problems. Section 4 gives the conclusions and future work.

2. Basic Concepts

This section gives some concepts and definitions.

2.1. The Control-Flow Graph (CFG)

A graph $G = (V, E)$ with two distinguished nodes n_0 (*entry*) and n_k (*exit*), which consists of a set V of nodes, where each node represents a statement, and a set E of directed edges, where a directed edge $e = (n, m)$ is an ordered pair of two adjacent nodes, called *tail* and *head* of *e*, respectively is called control-flow graph (CFG).

2.2 Dominances (Dom)

Let $G = (V, E)$ be a CFG with two distinguished nodes n_0 and n_k, the unique *entry* and *exit* nodes respectively. A node *n* dominates a node *m* if every path *P* from the *entry* node n_0 to *m* contains *n* [8].

The dominator tree $DT(G) = (V, E)$ is a CFG in which one distinguished node n_0, called the root, is the head of no edges; every node *n* except the root n_0 is a head of just one edge and there exists a (unique) dominance path from the root n_0 to each node *n*.

A node *m* postdominates by node *n* iff $m \neq n$ and every path from *n* to the *exit* contains *m*.

2.3 Control Dependencies (CD)

For nodes *n* and *m* in a CFG, *m* is control dependent on *n* iff (1) there exists a path *P* from *n* to *m* with all node *x* in *P* (excluding *n* and *m*) postdominated by m (2) *n* is not postdominated by *m*. Where, nodes represent statements, and edges represent the control dependencies between statements.

3. Dominance versus Control Dependencies

This section gives some problems of using the CD to guide the SBT. In addition, we present a methodology to overcome these problems.

3.1 The problems of the CD

- Determining the CD path for statements following the unstructured transfers of control, such as *goto, continue,* and *break* [2, 4] and *do-while* structure [9]. In these cases there is more than one path.
- Finding the approximation level. For example, in the case of selection structure nested within repetition structure (e.g., *if* structure within *for* structure) [4] and in the *switch-case* structure [4, 7].

3.2 Overcoming the CD problems

The key ingredients for using the dominances in the control-flow graph to define the fitness function are:

- From the definition of the dominance, there is a unique path between any two dominated nodes. Therefore, using dominance will solve the problem with unstructured transfers and *do-while* structure.
- From the definition of postdominance, there is a unique path between any two postdominated nodes. Thus, using postdominance will overcome the problem of finding the approximation level.

From the above key ingredients, the general form of the fitness function is:

Fitness = fit_value + approximation_value;

where *fit_value* is a function in dominance nodes, and *approximation_value* is a function in postdominance nodes. Currently, we work to define this function.

4. Conclusions and future work

This paper gave some problems of using the control dependencies to guide the SBT. In addition, the paper presented a general form for a function for guiding the SBT. Currently, we work to define this function and investigate its ability to overcome the problems of CD.

5. References

[1] M. Harman, "The Current State and Future of Search Based Software Engineering," Proc. of the International Conference on Future of Software Engineering (FOSE 07), May 2007, pp. 342-357.

[2] R. P. Pargas, M. J. Harrold, and R. R. Peck, “Test Data Generation Using Genetic Algorithms” Journal of Software Testing,Verifications and Reliability,vol.9,pp.263-282, 1999.

[3] A. S. Ghiduk, M. J. Harrold, M. R. Girgis, “Using Genetic Algorithms to Aid Test-Data Generation for Data Flow Coverage,” Proc. of 14th Asia-Pacific Software Engineering Conference (APSEC 07), Dec. 2007, pp. 41-48.

[4] P. McMinn, "Search-Based Software Test Data Generation: A Survey," Journal of Software Testing Verification and Reliability, vol. 14, no. 2, June 2004, pp. 105-156.

[5] N. Tracey, "A search-based automated test data generation framework for safety critical software," Ph. D. thesis, University of York, 2000.

[6] J. Wegener, K. Buhr, and H. Pohlheim, "Automatic test data generation for structural testing of embedded software systems by evolutionary testing" In Proc. of the 2002 Genetic and Evolutionary Computation Conference (GECCO ’02), 2002, pp 1233–1240.

[7] Y. Wang, Z. Bai, M. Zhang, W. Du, Y. Qin, and X. Liu, "Fitness calculation approach for the switch-case construct in evolutionary testing." In Proc. of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO ’08), 2008, pp 1767– 1774.

[8] T. Lengauer and R. E. Trajan, "A fast algorithm for finding dominators in a flowgraph." ACM Transactions on programming Languages and Systems, vol. 1, 1979, pp.121-141.

[9] T. Ball, and S. Horwitz, "Constructing control flow from control dependence," TR-92-1091, University of Wisconsin-Madison, 1992. http://www.cs.wisc.edu/wpis/papers/tr92-1091.ps.