Using Dominance for Aiding the Search-Based Testing to Overcome the Limitations of the Control-Dependencies

Ahmed S. Ghiduk
Faculty of Science, Beni-Suef University, Beni-Suef, Egypt
Email: asaghiduk@{yahoo.com OR gmail.com}

Abstract—Search-based optimization techniques have been utilized for a number of software engineering activities. The representation of the problem and the definition of the fitness function (FF) are 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 and efficiency of the search-based testing (SBT). Several search based test-data generation techniques have been developed. A wide range of these techniques utilized the control dependencies (CD) in the control-flow graph of the program under test for guiding the search in the direction of finding test data. To direct the SBT to generate test data, Ghiduk et al. have presented a search-based technique that utilizes the dominances (Dom) between the nodes of the control-flow graph (CFG) of the program under test. In this paper, we investigate the efficiency and effectiveness of dominances in a control-flow graph against the control dependencies in guiding the SBT for generating test data. The paper presents two schemes for overcoming these problems. The first scheme improves the definitions of the fitness functions of the previous work to overcome the control-dependencies problems. The second scheme presents a general form for a fitness function in terms of dominances and postdominances nodes. This function enhances significantly the efficiency of the SBT; consequently the SBT overcomes the control-dependencies problems. In addition, the paper compares between the efficiency of dominances and control dependencies in guiding SBT with proper examples from the literatures.

Index Terms—search-based testing; genetic algorithms, test-data generation, dominance, control dependencies

I. INTRODUCTION

Search-based optimization techniques (e.g., simulated annealing, genetic algorithms, ant colony and particle swarm) have been applied to a wide variety of software engineering activities including cost estimation, next release problem, and test-data generation [1]. Genetic algorithms have been the most widely employed search technique in search-based testing.

However, no matter what search technique is employed, it is the fitness function that captures the crucial information; it differentiates a good solution from a poor one, thereby guiding the search. Thus, a well-designed fitness function is essential to the effectiveness and efficiency of search-based testing. Several search based test-data generation techniques have been developed. A lot of these techniques guide the search to find the test data using the control dependencies in the control-flow graph of the program under test.

Search based test-data generation work had focused on finding test data to satisfy a number of control-flow and data-flow testing criteria (e.g., paths, branches, statements, and def-use [2, 3]). McMinn [4] surveyed the previous work undertaken in this area.

Pargas et al. [2] used the control-dependence graph of the program under test to define the fitness function. 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 structure (e.g., statements and branches).

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

\[ FFT = \left( \frac{\text{executed}}{\text{dependent}} \right) \times \text{dist} \quad \text{(1)} \]

where dependent is the number of the control-dependence nodes for the target structure, executed is the number of successfully executed control-dependence 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 \(m_{\text{dist}}\)). The fitness function is zero if the target structure is executed, otherwise, the fitness value is:

\[ FFW = m_{\text{dist}} + \text{approximation level} \quad \text{(2)} \]

where \(\text{approximation level} = (\text{dependent} - \text{executed} - 1)\).

In the work of Wang et al. [7], a flattened control-flow graph and a flattened control-dependence graph for the switch-case are presented and a fitness calculation approach is proposed for the switch-case structure. The formula:

\[ FFWa = \text{normalize(dist)} + \text{approximation level} \quad \text{(3)} \]

where \(\text{normalize(dist)} = 1 - 1.001^{-\text{dist}}\) is used to find the fitness value. When the execution diverges away at a switch branching node, the branch distance \(\text{dist}\) is \(\text{|expr-C|+1}\), 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, the branch distance 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 testing criteria. The technique applies the concepts of dominance relations between the nodes of the control-flow graph to define a multi-objective fitness function to evaluate the generated test data.

McMinn [4] has discussed the problems of the fitness functions which are based on the control dependencies.

In this paper, we investigate the efficiency of the dominance in the control-flow graph of the program under test to overcome the problems which due to using the control dependencies for guiding the SBT to find test data. The paper gives many key problems of the SBT which is guided by the control dependencies. The paper introduces two schemes for overcoming these problems. The first scheme redefines the fitness functions of the previous work to overcome the control-dependencies problems. The second scheme presents a general form for a new fitness function in terms of the dominances and postdominances in the control-flow graph of the program under test. This function significant improves the efficiency of the SBT; consequently the search overcomes the problems of the control dependencies. In addition, we use many proper examples from the literatures to compare between the efficiency of dominances and control dependencies in guiding SBT.

The rest of the paper is organized as follow. Section II gives some basic concepts and definitions. Section III introduces a number of the problems of the control dependencies based fitness functions. Section IV presents two schemes and the key ingredients to overcome these problems. Section V provides the related work. Section VI gives the conclusions and future work.

II. BASIC CONCEPTS

A. The Control-Flow Graph

A program’s structure is represented by a graphical representation called control-flow graph. A control-flow graph $G = (V, E)$ with two distinguished nodes $n_0$ (the unique entry) and $n_t$ (the unique exit), 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. A path $P$ in a control-flow graph is a finite sequence of nodes connected by edges. Figure 1(a) and Figure 1(b) give an example program Program1 and its control-flow graph, respectively. A node $n$ postdominates by node $n$ in a control-flow graph iff $m \neq n$ and every path from $n$ to the exit contains $m$.

Using postdominance, one can obtain the postdominator tree $PD(T)(G)$ (whose nodes represent the control-flow graph nodes) rooted at $n_0$. A tree $PD(T)(G) = (V, E)$ is a control-flow graph 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 $dom(n)$ (order sequence of nodes) from the root $n_0$ to each node $n$ [11]. Figure 1(c) gives the dominator tree of Program1. The dominance path of node $9$ is $dom(9) = entry, 1, 2, 5, 6, 7, 9$.

A node $m$ postdominates by node $n$ in a control-flow graph iff $m \neq n$ and every path from $n$ to the exit contains $m$.

Using postdominance, one can obtain the postdominator tree $PD(T)(G)$ (whose nodes represent the control-flow graph nodes) rooted at $n_0$. A tree $PD(T)(G) = (V, E)$ is a control-flow graph 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) postdominance path $pd(n)$ (order sequence of nodes) from the root $n_0$ to each node $n$ [11]. Figure 2(a) gives the postdominator tree of Program1. The postdominance path of node $9$ is $pd(9) = 9, 10, 6, 11, exit$.

The Control-Flow Graph 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 $dom(n)$ (order sequence of nodes) from the root $n_0$ to each node $n$ [11]. Figure 2(a) gives the postdominator tree of Program1. The postdominance path of node $9$ is $pd(9) = 9, 10, 6, 11, exit$.

B. Dominances

Let $G = (V, E)$ be a control-flow graph with two distinguished nodes $n_0$ and $n_t$, 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 dominance nodes in a control-flow graph can be obtained using the algorithm of Lengauer and Trajan [8].

Using dominance, one can obtain the dominator tree $DT(G)$ (whose nodes represent the control-flow graph nodes) rooted at $n_0$. A tree $DT(G) = (V, E)$ is a control-

C. Control Dependencies

For nodes $n$ and $m$ in a control-flow graph, $m$ 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 [2]. Figure 2(b) gives the control-dependence graph of Program1. The control-dependence graph can be constructed using the method of Ferrante et al. [9].

Figure 1. Program1 (a), its Control-Flow Graph (b), and its Dominator Tree (c).

Figure 2. Postdominator tree of Program1 (a), Control-dependence graph of Program1 (b).
D. Genetic Algorithms

The basic concepts of the genetic algorithm (GA) were developed by Holland [10]. GA creates a population of individuals represented by chromosomes, which are typically encoded solutions to a problem. The chromosomes then undergo a process of evolution according to rules of breeding. Each individual receives a measure of its fitness in the population. Breeding selects individuals with high fitness values in the population, and through crossing a new population is derived. The basic algorithm of GA is given below.

Simple Genetic Algorithm

\[
\begin{aligned}
\text{initialize population;} \\
\text{while termination criterion not reached} \\
\quad \text{select solutions for next population;} \\
\quad \text{perform crossover and mutation;} \\
\quad \text{evaluate population;}
\end{aligned}
\]

The previous algorithm will iterate until the population has evolved to form a solution to the problem, or until a maximum number of iterations have occurred.

III. THE CONTROL-DEPENDENCIES PROBLEMS IN GUIDING SBT

In this section, we introduce some key problems of using the control dependencies in the control-flow graph to guide the SBT. We utilize many proper examples from the literatures to show the problems of control dependencies in guiding SBT.

1. The first problem is determining the control-dependence path for the statements following unstructured transfers of control, such as goto, continue, and break [2, 4] and repeat-until structure (do-while structure) [12].

For example, consider the control-flow graph (CFG) in Figure 3(a). This control-flow graph was taken from the work of Ball and Horwitz [12]. The dominator tree of the control-flow graph CFG is given in Figure 3(b). Figure 4(a) gives the postdominator tree of the control-flow graph CFG. The control-dependence graph of the control-flow graph CFG is given in Figure 4(b).

In this example, suppose that the testing criterion is the all-statements criterion (all-nodes criterion) and the target structure is node 5 the shaded node in Figures 3(a), 3(b), 4(a), and 4(b). In addition, suppose that \(TD_1\) and \(TD_2\) are two individuals in a population (i.e., two groups of test data). The tests \(TD_1\) and \(TD_2\) execute the path \(EP_1 = (entry, T), (1, F), (7, exit)\) and the path \(EP_2 = (entry, T), (1, T), (2, T), (3, F), (6, T), 7, exit\) in the control-flow graph CFG, respectively.

In this case, there are two control-dependence paths from the \(entry\) node to the target node (node 5). These two control-dependence paths are \(CDP_1\) and \(CDP_2\). The path \(CDP_1\) is \((entry, T), (1, T), (2, F)\) and the path \(CDP_2\) is \((entry, T), (1, T), (2, T), (3, T)\).

Let us compute the fitness values for \(TD_1\) and \(TD_2\) by using the fitness functions of Pargas, Tracey, and Wegener.

- According to the fitness functions of Pargas et al. [2], the fitness value of \(TD_1\) is 1 (the number of predicates on the executed path \(EP_1\) that is common with the predicates on \(CDP_1\) and \(CDP_2\) the two control-dependence paths of 5).

The fitness value of \(TD_2\) is 2 (the number of predicates on the executed path \(EP_2\) that is common with the predicates on \(CDP_1\)) or 3 (the number of predicates on the executed path \(EP_2\) that is common with the predicates on \(CDP_2\)).

Consequently, \(TD_1\) has two identical fitness values 1 but \(TD_2\) has two different fitness values 2 and 3.

- According to the fitness functions of Tracey [5], the fitness value of \(TD_1\) is \(\frac{1}{3} \times dist\) or \(\frac{1}{4} \times dist\) (by substituting in equation (1)). Where, \(executed = 1\) the number of successfully executed control dependent nodes on path \(EP_1\) and \(dependent = 3\) or 4 the number of the control dependence nodes for the target node (node 5) on the two control-dependence paths \(CDP_1\) and \(CDP_2\), respectively.

The fitness value of \(TD_2\) is \(\frac{2}{3} \times dist\) or \(\frac{3}{4} \times dist\) (by substituting in equation (1)). Where, \(executed = 2\) or 3 the number of successfully executed control dependent nodes on path \(EP_2\) and \(dependent = 3\) or 4 the number of the control dependence nodes for node 5 on the two control-dependence paths \(CDP_1\) and \(CDP_2\), respectively.
Consequently, both $TD_1$ and $TD_2$ have two different fitness values ($\frac{1}{3} \times dist$ and $\frac{1}{4} \times dist$ for $TD_1$ and $\frac{2}{3} \times dist$ and $\frac{3}{4} \times dist$ for $TD_2$).

- According to the work of Wegener et al. [6] the fitness value of $TD_1$ is equal to $m \_dist + 1$ or $m \_dist + 2$ (by substituting in equation (2)). Where, $executed = 1$ the number of successfully executed control dependent nodes on path $EP_1$ and $dependent = 3$ or $4$ the number of the control dependence nodes for node 5 on the two control-dependence paths $CDP_1$ and $CDP_2$, respectively.

The fitness value of $TD_2$ is equal to $m \_dist$ (by substituting in equation (2)). Where, $executed = 2$ or $3$ the number of successfully executed control dependent nodes on path $EP_2$ and $dependent = 3$ or $4$ the number of the control-dependence nodes for node 5 on the two control-dependence paths $CDP_1$ and $CDP_2$, respectively.

Consequently, $TD_1$ has two different fitness values ($m \_dist + 1$ or $m \_dist + 2$) but $TD_2$ has two identical fitness values ($m \_dist$).

From the previous discussion of the example, we can conclude that the fitness functions which were used by the search-based test-data generation techniques of Pargas et al. [2], Tracey [5], and Wegener et al. [6] cannot evaluate the individuals ($TD_1$ and $TD_2$) (i.e., these functions cannot find the fitness values of $TD_1$ and $TD_2$) because all of these fitness functions use the $dependent$ (the number of the control-dependence nodes for the target structure) as a vital term of their definitions. Therefore, all of these fitness functions will fail in determining a unique set of $dependent$ nodes for the target (node 5) subsequently cannot find the fitness value.

2. The second problem is the poor search performance in the case of selection structure nested within repetition structure (e.g., if structure within if structure) [4].

Figure 5 which was presented by McMinn [4] demonstrates the problem of structures nested within loops. Figure 5 gives an example program (loop_example) for if structure nested in if structure to the left and its control-flow graph to the right.

Figure 6 shows the control-dependencies graph of the loop_example to the left and its dominator tree to the right.

Node 3 (the shaded node in Figure 5(b)) is the structure to be covered (i.e., the target of the search).

In fact, paths taking the false branch from node 2 can still execute node 3 in subsequent iterations of loop. Therefore, node 3 is not control dependent on node 2. Consequently, the search does not receive guidance regarding the fact that the true branch from node 2 must be covered to reach the target statement. This problem results in poor search performance.

For example, suppose that $TD_1$ and $TD_2$ are two test data which execute the two paths $P_1$ and $P_2$, respectively. Where, $P_1 = entry, 1, 2, 1, exit$ and $P_2 = entry, 1, exit$.

Similar to the example in the first problem in this section, we can compute the fitness values of $TD_1$ and $TD_2$ using the fitness functions of Pargas et al. [2], Tracey [5], and Wegener et al. [6]:

$entry \ void \ loop\_example (int \ i) \$

{ int n;
  for (n = 0 ; n <= 10; n++)
    { if ( n == 10 & & i == 0 )
    { // target statement
      exit 
    }
  }
}

According to the fitness functions of Pargas, both $TD_1$ and $TD_2$ have the same fitness value (the value is 1 because there is one predicate on the executed paths $P_1$ and $P_2$ common with the control-dependencies path of node 3). Similarly, both the fitness functions of Tracey and Wegener will assign for each of $TD_1$ and $TD_2$ the same approximation level.

From the previous discussion of the example, we can conclude that the control-dependencies based fitness functions do not give any guidance for the search and cause a poor in the search performance.

3. The third problem is assignment of the approximation level for some classes of program with unstructured control flow [13].

Figure 7 which is taken from the work of Baresel et al. [13] demonstrates the problem of the assignment of approximation level for program with unstructured control flow such as switch-case structure. Figure 7(a) shows the code of the switch-case structure and Figure 7(b) gives its control-flow graph. Figure 7(c) gives the control-dependencies graph of this structure and Figure 8 gives its dominator tree.

Suppose that the target of the search is the execution of node 6 (the shaded node in figures 7(b) and 7(c)). However, there are three different control-dependent paths from the entry node through node 6; the first path is $entry, 1, 5, 6$, the second path is $entry, 1, 2, 6$, and the third path is $entry, 1, 2, 3, 5, 6$ (see Figure 7(c)) and two possibilities for node 2; the first path is $2, 3, 4, 5, 6$ and the second path is $2, 3, 6$.

According to the computation of the fitness values in the first and the second problems, one can compute the fitness values and the approximation levels for the third problem by using the functions of Pargas et al.
[2], Tracey [5], and Wegener et al. [6]. Consequently, there are two fitness values and/or two approximation level values possibilities (since two of the paths are of the same length). This problem results in misleading of the search.

```
switch (x) {
    1.  case 1:  
        if (condition_1)
            return;
    2.  if (condition_2)
        return;
    3.  case 2:  
        if (condition_3)
            return;
    4. }
// target statement
```

Figure 7. Unstructured control flow example (a), its control-flow graph (b), and its control-dependence graph (c).

Figure 8. Dominator of the unstructured control flow example.

IV. OVERCOMING THE PROBLEMS OF CONTROL DEPENDENCIES

In this section, we introduce two schemes for overcoming the problems of using the control-dependencies in the control-flow graph for guiding the search-based testing techniques to generate the test data. Our two schemes depend on using the dominances in the control-flow graph instead of the control-dependencies to define the fitness function. We use the same examples in section III to show the efficiency of dominances in guiding SBT against control dependencies.

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

1. According to the definition of the dominance in the control-flow graph, there is a unique path between any two dominated nodes. Therefore, using dominance will overcome the problem of determining the control-dependencies path for some structures. Where some structures have multi-control-dependencies paths for the target structure such as the cases of unstructured transfers and do-while structure.

2. From the definition of postdominance, there is a unique path between any two postdominated nodes. Thus, using postdominance can overcome the problems of assignment the approximation level for some structures.

From the above two key ingredients, our proposal to overcome the problems which are related to the control-dependencies has two schemes. The first scheme is redefining the previous fitness functions using the dominances instead of the control dependencies. The second scheme is constructing a general form for the fitness function using the dominances and postdominances in the control-flow graph.

A. The first scheme: redefining the previous fitness functions

The main idea of the first scheme of our proposal is replacement the control-dependencies based terms in the definitions of the previous related fitness functions (i.e., Pargas, Tracey, and Wegener fitness functions) with dominances-based terms.

According to our first scheme, the redefined fitness functions of Pargas, Tracey, and Wegener will be as follows.

- The redefined fitness function of Pargas (RFFP) is the number of nodes on the executed path that is common with the nodes on the dominance path of the target structure.
- The redefined fitness function of Tracey (RFFT) is the following formula:

\[
RFFT = \left( \frac{\text{executed}}{\text{dominated and ated}} \right) \times \text{dist} \quad \quad \quad \quad (4)
\]

where dominated is the number of the dominance nodes for the target structure, executed is the number of successfully executed dominance nodes, and dist is the branch distance calculation performed at the branching node.

- The redefined fitness function of Wegener (RFFW) is zero if the target structure is executed, otherwise, the fitness value is:

\[
RFFW = m_{\text{dist}} + \text{approximation level} \quad \quad \quad (5)
\]

where approximation level = (dominated-executed).

To explain the efficiency of the new versions (RFFP, RFFT, and RFFW) of the fitness functions in overcoming the problems of the control-dependencies based functions, consider the control-flow graph (CFG) and its dominator tree in Figures 3(a) and 3(b), respectively.

Suppose also that the target structure is node 5 the shaded node in Figures 3(a) and 3(b). In addition, suppose that TD1 and TD2 are two test cases. TD1 and TD2 execute the path \( EP_1 = \text{entry}, 1, 7, \text{exit} \) and the path \( EP_2 = \text{entry}, 1, 2, 3, 6, 7, \text{exit} \) in the control-flow graph CFG, respectively, see Figure 3(a). We have illustrated that the old versions of these fitness functions cannot evaluate the tests TD1 and TD2 because they assign more than one fitness value for each of them.

The computation of the fitness values for TD1 and TD2 using the new functions (i.e., RFFP, RFFT, and RFFW) is depicted in the following paragraphs.

- According to RFFP the new version of Pargas, the fitness value of TD1 is computed as follows: Where the path \( EP_1 = \text{entry}, 1, 7, \text{exit} \) is executed by TD1 and the dominance path dom(5) of node 5 is dom(5) = entry, 1, 2, 5. Therefore, the fitness value of TD1 is RFFP(TD1) = 2 because entry and 1
are the only common nodes between $EP_1$ and $dom(5)$. In fact, $RFFP(TD_1)$ is unique value. Similarly, $RFFP(TD_2)$ is unique value.

- According to $RFFP$ the fitness value of $TD_1$ is $\frac{2 \times dist}{4}$ and the fitness value of $TD_2$ is $\frac{3 \times dist}{4}$. It is clear that, the fitness values of $TD_1$ and $TD_2$ are unique values.

- According to $RFFW$ the new version of Wegener the fitness value of $TD_1$ is $m\cdot dist +1$ and the fitness value of $TD_2$ is $m\cdot dist +1$. It is clear that, the fitness values of $TD_1$ and $TD_2$ are also unique values.

From the above example, we can conclude that the new versions of the fitness functions of Pargas, Tracey, and Wegener succeeded to overcome the first problem of the control-dependencies based fitness functions by generating unique value for each test case.

Concerning the second problem, consider the example with the second problem in Figure 5. In this example the target node is node 3 and the test cases are $TD_1$ and $TD_2$ which execute the two paths $P_1 = \text{entry}, \ 1, \ 2, \ 1, \ \text{exit}$, and $P_2 = \text{entry}, \ 1, \ \text{exit}$, respectively. From Figure 6(b), the dominance path of node 3 is $\text{dom}(3) = \text{entry}, \ 1, \ 2, \ 3$.

- According to $RFFP$ the new version of Pargas the fitness value of $TD_1$ is 3 and the fitness value of $TD_2$ is 2.

- According to $RFFT$ the new version of Tracey the fitness value of $TD_1$ is $\frac{3 \times dist}{4}$ and the fitness value of $TD_2$ is $\frac{2 \times dist}{4}$.

- According to $RFFW$ the new version of Wegener the fitness value of $TD_1$ is $m\cdot dist + 1$ and the fitness value of $TD_2$ is $m\cdot dist + 2$.

From the above example, we can conclude that the new versions of the fitness functions of Pargas, Tracey, and Wegener succeeded to overcome the second problem of the control-dependencies based fitness functions by improving the search in the convergence direction.

Concerning the third problem, consider the example with the third problem in Figure 7. In this example the target node is node 6. In addition, there are three control-dependencies paths from entry to node 6 but the path $\text{dom}(6) = \text{entry}, \ 1, \ 2, \ 3, \ 6$ is the only dominance path from the entry to node 6. Therefore, each function of the new versions of the fitness functions of Pargas, Tracey, and Wegener will find a single fitness value for the test cases; consequently all of them will overcome the third problem.

The computation of the fitness values for $TD_1$ and $TD_2$ using the new version of the fitness functions is as follows.

- According to $RFFP$ the fitness value of $TD_1$ is 2 and the fitness value of $TD_2$ is 5.

- According to $RFFT$ the fitness value of $TD_1$ is $\frac{2 \times dist}{5}$ and the fitness value of $TD_2$ is $\frac{5 \times dist}{5}$.

- According to $RFFW$ the fitness value of $TD_1$ is $m\cdot dist + 3$ and the fitness value of $TD_2$ is zero.

The above discussion shows the efficiency of dominances in guiding $SBT$ against control dependencies.

We can replace all the pervious fitness functions by a unique fitness function. In the next section we introduce the definition of this fitness function.

**B. The second scheme: constructing a general form for the fitness function.**

The second scheme focuses on constructing a general form for the fitness function based on the dominances and postdominances relationships in the control-flow graph of the program under test.

From the definitions of the dominances and postdominance (implication) relations which is given by Bertolino and Marré [11], we can define the essential path for any node $n$ in a control-flow graph of a program as follows.

Definition 1: Essential Path

For any node $n$ in a control-flow graph $G = (V, E)$ of a program with two distinguished nodes $n_0$ (the unique entry) and $n_k$ (the unique exit), there is a path

$$P = n_0, n_{m_0}, n_{m_1}, ..., n_{m_i}, n_{n_k}, n_{n_{l_1}}, n_{n_{l_2}}, ..., n_{n_{l_j}}, n_k$$

where $n_0, n_{m_0}, n_{m_1}, ..., n_{m_i}$ are nodes of the dominator tree of the control-flow graph $G$ and $n_{n_k}, n_{n_{l_1}}, n_{n_{l_2}}, ..., n_{n_{l_j}}$ are nodes of the postdominator tree of the control-flow graph $G$, such that any path covers $n$ has to cover all the other nodes of $P$. Then, $P$ is called the essential path of $n$. The path $P$ is the concatenation of the dominances path of node $n$ and the postdominance path of $n$.

For example the dominance path of node 5 in Figure 3(b) is entry, 1, 2, 5 and the postdominance path of 5 in Figure 4(a) is 5, 6, 7, exit. Therefore, the essential path of node 5 is entry, 1, 2, 5, 6, 7, exit.

The key idea of our suggested fitness function is how far is the executed path from covering the essential path of the target? In other words, on the essential path of the target how many nodes in the front of the target without covering and how many nodes in behind it without covering? Consequently, our fitness function measures the coverage ratio of the essential path of the target. Therefore, the general form of the fitness function is:

$$FF = fit\_value + \text{approximation\_value}$$

where the $fit\_value$ is a function in dominance nodes, and the $\text{approximation\_value}$ is a function in postdominance.
nodes. The fit_value is computed from the following formula.

$$\text{fit_value} = \frac{1}{2} \times \frac{\text{Common_dom}}{\text{Total_dom}}$$

where $\text{Common_dom}$ is the number of nodes on the executed path that is common with the nodes on the dominance path of the target structure and $\text{Total_dom}$ is the total number of nodes on the dominance path of the target structure. In addition, the approximation_value is computed from the following formula.

$$\text{approximation_value} = \frac{1}{2} \times \frac{\text{Common_postdom}}{\text{Total_postdom}}$$

where $\text{Common_postdom}$ is the number of nodes on the executed path that is common with the nodes on the postdominance path from the target structure to the root of the postdominator tree and $\text{Total_postdom}$ is the total number of the nodes on this postdominance path.

The target is covered when the fitness function $FF$ assigns the value 1.

To illustrate the efficiency of the new fitness function in overcoming the problems of the control dependencies based fitness functions:

- Concerning the first problem, consider the example in Figures 3 and 4. In addition, suppose that $TD_1$ and $TD_2$ are two test cases which execute the path $EP_1 = (\text{entry}, T), (1, F), 7, \text{exit}$ and the path $EP_2 = (\text{entry}, T), (1, T), (2, T), (3, F), (6, T), 7, \text{exit}$ in the control-flow graph $CFG$, respectively and the target structure is node 5 the shaded node in Figures 3 and 4.

From the dominator tree and the postdominator tree which are given in Figures 3(b) and 4(a), respectively, the essential path $P$ of node 5 is $\text{entry, } 1, 2, 5, 6, 7, \text{exit}$. In addition, the dominance path of node 5 is $\text{entry, } 1, 2, 5$ and the postdominance path of 5 is $5, 6, 7, \text{exit}$.

For the first test case $TD_1$, the common nodes between the executed path $EP_1 = (\text{entry}, T), (1, F), 7, \text{exit}$ and the dominance path of node 5 dom(5)=entry, 1, 2, 5 are entry and 1. Therefore, $\text{Common_dom} = 2$ and $\text{Total_dom} = 4$. In addition, the common nodes between the executed path $EP_1$ and the postdominance path of 5 $\text{postdom}(5)=5, 6, 7, \text{exit}$ are 7 and exit. Therefore, $\text{Common_postdom} = 2$ and $\text{Total_postdom} = 3$.

The fitness value of $TD_1$ according to the new fitness function

$$FF = \text{fit_value} + \text{approximation_value} = \frac{1}{2} \times \frac{\text{Common_dom}}{\text{Total_dom}} + \frac{1}{2} \times \frac{\text{Common_postdom}}{\text{Total_postdom}}$$

$$= \frac{1}{2} \times \frac{2}{4} + \frac{1}{2} \times \frac{2}{3} = 0.58$$

For the second test case $TD_2$, the common nodes between the executed path $EP_2 = (\text{entry}, T), (1, T), (2, T), (3, F), (6, T), 7, \text{exit}$ and the dominance path of node 5 are entry, 1, and 2. Therefore, $\text{Common_dom} = 3$ and $\text{Total_dom} = 4$. In addition, the common nodes between the executed path $EP_2$ and the postdominance path of 5 are 6, 7, and exit. Therefore, $\text{Common_postdom} = 3$ and $\text{Total_postdom} = 3$.

The fitness value of $TD_2$ according to the new fitness function

$$FF = \text{fit_value} + \text{approximation_value} = \frac{1}{2} \times \frac{\text{Common_dom}}{\text{Total_dom}} + \frac{1}{2} \times \frac{\text{Common_postdom}}{\text{Total_postdom}}$$

$$= \frac{1}{2} \times \frac{3}{4} + \frac{1}{2} \times \frac{3}{3} = 0.88$$

From the fitness values of the two test cases $TD_1$ and $TD_2$ we can conclude that our suggested fitness function is more efficient than the versions of the previous fitness functions because it succeeded to evaluate $TD_1$ and $TD_2$ by finding a unique fitness value for each of them. In addition, it has the ability to distinguish and differentiate between the two test cases. It is clear that the fitness value of $TD_2$ is greater than the fitness value of $TD_1$ because the executed path of $TD_2$ is closer to cover the target node than the executed path of $TD_1$. Consequently, the new fitness function can easily order the generated test cases.

- Concerning the second problem, consider the example in Figure 5 and Figure 6 which demonstrate the problem of structures nested within loops. Suppose that $TD_1$ and $TD_2$ are two tests which execute the two paths $P_1 = \text{entry, } 1, 2, 1, \text{exit}$, and $P_2 = \text{entry, } 1, \text{exit}$, respectively. Node 3 in Figures 5 and 6 is the structure to be covered.

From the dominator tree and the postdominator tree which are given in Figures 6(b) and 6(c), respectively, the essential path $P$ of node 3 is $\text{entry, } 1, 2, 3, 1, \text{exit}$. In addition, the dominance path of node 3 is $\text{entry, } 1, 2, 3$ and the postdominance path of 3 is $3, 1, \text{exit}$.

For the first test case $TD_1$, the common nodes between the executed path $P_1$ and the dominance path of node 3 are entry, 1, and 2. Therefore, $\text{Common_dom} = 3$ and $\text{Total_dom} = 4$. In addition, the common nodes between the executed path $P_2$ and the postdominance path of 3 are 1 and exit. Therefore, $\text{Common_postdom} = 2$ and $\text{Total_postdom} = 3$.

The fitness value of $TD_1$ according to the new fitness function

$$FF = \text{fit_value} + \text{approximation_value} = \frac{1}{2} \times \frac{\text{Common_dom}}{\text{Total_dom}} + \frac{1}{2} \times \frac{\text{Common_postdom}}{\text{Total_postdom}}$$

$$= \frac{1}{2} \times \frac{3}{4} + \frac{1}{2} \times \frac{3}{3} = 0.71$$

For the second test case $TD_2$, the common nodes between the executed path $P_2$ and the dominance path of node 3 are entry and 1. Therefore, $\text{Common_dom} = 2$ and $\text{Total_dom} = 4$. In addition, the common nodes between the executed path $P_2$ and the postdominance path of 3 are 1 and exit. Therefore, $\text{Common_postdom} = 2$ and $\text{Total_postdom} = 3$.

The fitness value of $TD_2$ according to the new fitness function

$$FF = \text{fit_value} + \text{approximation_value} = \frac{1}{2} \times \frac{\text{Common_dom}}{\text{Total_dom}} + \frac{1}{2} \times \frac{\text{Common_postdom}}{\text{Total_postdom}}$$

$$= \frac{1}{2} \times \frac{2}{4} + \frac{1}{2} \times \frac{2}{3} = 0.58$$
Concerning the third problem, consider the example with the third problem in Figure 7. In this example the target node is node 6. In addition, there are three control-dependencies paths from entry to node 6. In contrast there is only one dominance path from the entry to node 6 and one postdominance path from node 6 to exit. Therefore, our fitness function will assign a unique value for each of the two test cases TD1 and TD2. Consequently our new fitness function will overcome the third problem.

One can compute the two fitness values of TD1 and TD2 by the same method of the first and the second problems.

The above discussion has showed that using dominances and postdominances for defining the fitness function enhances significantly the efficiency of the SBT; consequently SBT overcomes the control-dependencies problems.

Although RFFP, RFFT, RFFW and the general fitness function depend on dominances and postdominances, each function has a different definition. Therefore, there is no any correlation between the fitness values calculated using RFFP, RFFT, and RFFW and the fitness values calculated using the general formula of the fitness function for the test cases TD1 and TD2.

V. THE RELATED WORK

In this section, we discuss how the priorous works deal with the control dependencies problems.

The work of Parags et al. [2] does not address any of the above problems of the control dependencies.

Baresel et al. [13] treat branches in the second problem that miss the target in iterations of the loop as critical branches. Therefore, node 3 is treated as if it were control dependent on node 2. In order to circumvent this problem, Tracey [5] examines the branch distance during each iteration of the loop and uses the minimum branch distance obtained for computing the final fitness value.

For the third problem, McMinn [4] claims that there is two plausible solutions to this problem include optimistic and pessimistic approximation level allocation strategies. In an optimistic strategy, a control dependent branching node is allocated its approximation level on the basis of the shortest control dependent path from itself to the target node. In this way the approximation level of node 2 is 1 on the basis of the direct path (2, 3, 6) through node 6. In the pessimistic strategy, a branching node is allocated its approximation level on the basis of the longest control dependent path to the target node. In this scheme node 2 would be assigned an approximation level of 3 on the basis of the paths 2, 3, 4, 5, 6.

Baresel et al. [13] used both optimistic and pessimistic strategies in the initial experiments. They show that the different strategies have different effects on the progress of the search. They cannot conclude which strategy works best in general.

For switch-case problem, Wang et al. [7] converted the control-flow graph and the control-dependencies graph for the switch-case into a flattened control-flow graph and a flattened control-dependence graph. Consequently, a fitness calculation approach is proposed for the switch-case structure which is based on alternative critical branches for the switch-case.

VI. CONCLUSIONS AND FUTURE WORK

Search-based optimization techniques have been applied to a wide variety of software engineering activities. Genetic algorithms have been the most widely employed search technique in search-based testing. The representation of the problem and the definition of the fitness function are two key ingredients for the application of search-based optimization to software engineering problems. Therefore, a well-defined fitness function is essential to the efficiency of SBT.

This paper introduced many key problems of using the control-dependencies in the control-flow graph to guide the search-based testing for generating test data. In addition, the paper presented two schemes for overcoming these problems. These schemes employed the dominances in the control-flow graph instead of the control-dependencies for guiding the search-based testing to find the test data. The first scheme redefined the fitness functions of the previous related work by replacement the control-dependencies based terms in the definitions of these fitness functions with dominances-based terms. The second scheme provided a general form for the fitness function with dominances- and postdominance-based terms for guiding the SBT.

The paper presented a theoretical comparison between the efficiency of dominances and control dependencies in guiding SBT by using proper examples from the literatures. The presented comparison showed the ability of the proposed schemes to overcome the problems of using the control-dependencies in the control-flow graph to guide the SBT to generate the test data.

Our future work will focus on conducting an empirical study to show how much the efficiency of the generated test data improved, after applying the control dependency-based and dominance-based approach to some real projects, respectively. In addition, we will use dominance relationship in the control flow graph to solve other fitness value calculation problems. One of the open problems is how we can apply the dominance to generate feasible path cover for a given test coverage criterion.

ACKNOWLEDGMENT

The author would like to thank the reviewers of the first International Symposium on Search-Based Software
REFERENCES


Ahmed S. Ghiduk is an assistant professor at Beni-Suef University, Egypt. He received the BSc degree from Cairo University, Egypt, in 1994, the MSc degree from Minia University, Egypt, in 2001, and a Ph.D. from Beni-Suef University, Egypt in joint with College of Computing, Georgia Institute of Technology, USA, in 2007. His research interests include software engineering especially search-based software testing, genetic algorithms, and ant colony and web application testing. Currently, Ahmed S. Ghiduk is an assistant professor at College of Computers and Information Systems, Taif University, Saudi Arabia. One can connect Ahmed S. Ghiduk on asaghiduk@yahoo.com or gamil.com.