Chapter II

Literature Review

2.1. Introduction

Search-based test data generation consists of exploring the input domain of a program under test for test data to satisfy a selected test data adequacy criterion. By using meta-heuristic techniques - high-level frameworks which utilize heuristics in order to find solutions - search is directed towards the most promising areas of the domain (Michael et al., 2001; McMinn, 2004).

In this Chapter we present a survey on search-based test data generation. Section 2.2 presents a general survey on search based test data generation and Section 2.3 focuses on program-based test data generation. Section 2.4 outlines the problems addressed in this thesis and Section 2.5 reviews related work.

2.2. Search Based Test Data Generation

McMinn (McMinn, 2011) notes that “work on Search-Based Software Testing dates back to 1976, with interest in the area beginning to gather pace in the 1990s.

More recently there has been an explosion of the amount of work.” He further notes that “since the cost of manual testing in practice is very high, research into automated software testing is of high concern. Search-Based Software Testing is a very generic approach in which solutions may be sought for software testing problems automatically, using optimisation algorithms.” The major problems that he outlines are a) lack of handling of the execution environment that the software under test lives within, b) the failure, in many cases, of fitness functions to give adequate guidance to the search and c) oracles and their automation. We have worked on problem (b) in this thesis and have also identified some other typical issues in test data generation which are outlined in Section 2.4.

Harman et al (Harman et al., 2009) and Harman and Mansouri (Harman and Mansouri, 2010) have published an extensive survey on Search based Software Engineering and in particular
Search Based Software Testing. They note that “The adequacy criteria must be captured by a fitness function. This has to be designed by a human, but once a fitness function has been defined for a test adequacy criterion, C, then the generation of C-adequate test inputs can be automated using Search Based Software Engineering.” They also note that most tools for test data generation follow the general structure outlined in Figure 2.1. The setups in this thesis follow a similar approach. Further, the authors list, but do not describe a number of references in Search Based Software Testing. In this chapter we briefly present some of the important contributions that are relevant to the dissertation.

Figure 2.1. A Generic Search Based Test Data Generation Scheme (Harman et al., 2009) (Edvardsson, 1999) presents a survey on program based test data generation techniques which focuses on the notion and basic concept of test data generation and explains the
functioning of a test data generator system. (McMinn, 2004) and (Harman, et al., 2005) discuss search based software test data generation techniques. They identify a number of problems for search based test data generation, but note that Metaheuristic search techniques offer much promise in regard to these problems. The problems include dealing with flag and enumeration variables; flat landscapes; unstructured control flow; and state behaviour.

Metaheuristic techniques are ideal for application to search based test data generation. Metaheuristic techniques such as simulated annealing (Tracey et al., 1998a; Tracey and Clark, 1998b; Tracey, 2000), Tabu Search (D’iaz et al., 2003), genetic algorithms (Jones and Sthamer, 1996; Michael et al., 1997; Pargas et al., 1999; Pachauri and Srivastava, 2012a; Pachauri and Srivastava, 2012b), particle swarm optimization (Windisch et al., 2007), binary particle swarm optimization (Agarwal et al., 2008), quantum particle swarm optimization (Agarwal and Srivastava, 2010), scatter search (Blanco et al., 2009), ant colony optimization (Li and Lam, 2005), memetic algorithms (Arcuri and Yao, 2007), clonal selection algorithm (Castro and Zuben, 2002) and immune genetic algorithm (Liaskos and Roper, 2008; Tan et al., 2009) have been applied to the problem of test data generation and provide evidence of their successful application.

(Harman et al., 2009) note that research on Search Based Test Data Generation for program-based criteria is now relatively mature, and work is now focussing on theoretical analysis of the problem characteristics and specially tailored search algorithms that are adapted specifically to the program based structural test data generation problem. In this context, Harman et al. (Harman et al., 2007b) show how traditional Software Engineering dependence analysis techniques can be used to analyze dependence for predicates under test in structural testing. The dependence analysis can be used to reduce the size of the search space for test data generation, thereby improving the efficiency of test data generation. (Bouchachia, 2007) has proposed a modification to genetic algorithm by incorporating immune operators to it. This approach applies to the condition coverage testing criteria. (Alba and Bernabé, 2008) describe the effect of parameters and operators of Genetic Algorithm such as crossover operators, mutation probability, crossover probability and parent selection strategy on Genetic Algorithm. They generate test data for condition coverage using two Genetic Algorithms and two approaches based on Evolutionary Strategies. (Shen et al., 2009) have proposed an intermediate hybrid search technique using genetic algorithm and Tabu search, named as GATS algorithm. Tabu search is a local search technique and is used as a mutation operator. (Li et al. 2010) propose a new type of algorithm, GPSMA (genetic particle swarm mixed
algorithm) to improve the accuracy and convergence speed. (Harman et al., 2010) suggest a memory based approach that uses the greedy approach and genetic algorithm to reduce the oracle cost of software under test by reducing the number of test data generated. This thesis also focusses on ways of augmenting search for test data with genetic algorithm using different schemes. Taking a step forward Wilkerson (Wilkerson, 2012) has worked on evolutionary driven search based testing and correction. For a given program, testing, locating, and correcting faults is critical. The work addresses these challenging problems through the use of two complimentary evolutionary computing based systems. The first one is the Fitness Guided Fault Localization (FGFL) system, which uses a specification based fitness function to perform fault localization. The second is the Coevolutionary Automated Software Correction (CASC) system, which employs a variety of evolutionary computing techniques to perform testing, correction, and verification of software.

2.3. Program Based Structural Test Data Generation

Harman et al. (Harman et al., 2009) note that “The most widely studied area of testing research that has been addressed using SBSE is structural testing…The idea is to instrument the program to measure coverage of some structural criterion. The most commonly considered criterion is branch coverage, though other structural criteria have been attacked.” Several papers have addressed the issue of test data generation with program-based structural criteria, referred to as program-based criteria here, (Michael et al., 2001; Girgis., 2005; Andreou et al., 2007; Chen and Zhong, 2008; Ahmed and Hermadi, 2008; Ghani and Clark, 2009) and in particular the branch coverage criterion (Jones et al., 1998; Wegener et al., 2001; Blanco et al., 2007; Agarwal et al., 2008; Wang et al., 2008; Harman, 2008; Chen et al., 2009; Gross et al., 2009; Agarwal and Srivastava, 2010; Pachauri and Srivastava, 2012a).

In program based test data generation using meta-heuristic techniques, the basic approach is that of dynamic test data generation described in Section 1.2.1. The source code of the program under test is instrumented to collect information about the program as it executes. The resulting information, collected during each execution of the program, is used to heuristically determine how close the test case is to satisfying a specified test requirement as specified by the selected test criterion. This allows the test generator to modify the program’s inputs gradually, moving them ever closer to values that actually do satisfy the requirement. In other words, the problem of generating test data reduces to the well understood problem of function optimization. Furthermore, such test generation methods can handle arrays and
pointer references because the values of array indices and pointers are known throughout the generation process. Wegener et al. (Wegener et al., 2002) point out that because of the non-linearity of software (conditional statements, loops, flags, switch-case, break) the conversion of test problems into optimization tasks usually results in complex, discontinuous and non-linear search spaces for which search methods such as hill climbing are not suitable, but metaheuristic search methods can be employed.

Amongst the foundation papers for program-based test data generation are the papers by Korel (Korel, 1990a; Korel, 1990b). (Korel, 1990a) describes a test data generation approach based on program execution, dynamic data flow analysis, and the function minimization methods. The test data generation problem is reduced to a sequence of sub-goals. Function minimization methods are used to solve these sub-goals. Dynamic data flow analysis is applied to speed up the search process by identifying those input variables that influence undesirable program behaviour. (Korel, 1990b) describes two approaches for test data generation: random and path-wise. He introduces the concept of dynamic test data generation for node coverage criterion, critical branch and sub-goals.

2.3.1. Test Data Generation for Path Coverage

Many papers have focussed on search based test data generation for path coverage, where paths are identified in the control flow graph of the program under test. (Bueno and Jino, 2001; Bueno and Jino, 2002) present a new technique and a tool for test data generation for path testing. They are based on the dynamic test data generation technique using Genetic Algorithm, which evolves a population of input data towards reaching and solving the predicates along the program paths. They improve the performance of test data generation by using past input data to compose the initial population for the search. (Mansour and Salame, 2004) present simulated annealing algorithm (SA), and a genetic algorithm (GA) for generating test data that execute specified paths in a program. These are based on an optimization formulation of the path testing problem which includes both integer and real-valued test cases. They empirically compare the SA and GA algorithms with each other and with a hill-climbing algorithm, Korel’s algorithm (KA), for integer valued input subject programs and compare SA and GA with each other on real-value subject programs. They claim that SA performs better than GA but both SA and GA are superior to KA in the number of executed paths. However, KA, when it succeeds in finding the solution, is the fastest. (Ghiduk et al., 2007) proposed a test data generation approach using du (definition-use) paths
coverage testing criteria. They focus on generating the dominance tree from the control flow graph of the program. (Ahmed and Hermadi, 2008) describe test data generation for multiple specific paths using genetic algorithm. They have designed a GA-based test data generator which is able to synthesize multiple test data to cover multiple target paths in one run. (Srivastava and Kim, 2009) have proposed a test data generation technique for path coverage criteria using genetic algorithm based on the criticality of the path. Weights are assigned to the edges of the control flow graph and fitness is calculated by taking the sum of the weights of all the edges of a particular path. The path with maximum value of fitness is the most critical. (Alakeel, 2010) suggests a new type of heuristic approach, which uses data-dependency analysis among assertions, for array variables, to improve the accuracy and maximize code coverage using path testing.

Lin and Yeh (Lin and Yeh, 2001) describe a genetic algorithm that can be used to generate test cases for a selected path. They introduce a metric named Normalized Extended Hamming Distance, based on which a function named SIMILARITY is defined to determine the fitness of test data. Since this proposal is used in this thesis, we elaborate on this proposal. Let \( Q \) be the set of all program paths, referred to as complete paths in the paper, in program \( P \)'s control-flow graph. Let \( S^1_i = \{ g | g \text{ is an edge of the } i^{th} \text{ complete path, } \text{path}_i \} \), \( S^2_i = \{ g | g \text{ is an ordered pair of cascaded edges of } \text{path}_i \} \), \ldots, \( S^n_i = \{ g | g \text{ is an ordered n-tuple of cascaded edges of } \text{path}_i \} \), then the normalized \( n^{th} \) order distance, \( \text{NEHD} \), between \( \text{path}_i \) and \( \text{path}_j \) is defined as

\[
\text{NEHD}_{i-j} = \frac{|S^1_i \oplus S^n_j|}{|S^1_i|} \]

where \( \oplus \) represents symmetric difference. The \( n^{th} \) order similarity between \( \text{path}_i \) and \( \text{path}_j \) is defined as \( \text{SIMILARITY}_{i-j} = 1 - \text{NEHD}_{i-j} \). Fitness function \( \text{SIMILARITY} \) between \( \text{path}_i \) and \( \text{path}_j \) is now defined as \( \text{SIMILARITY}_{i-j} = W_1 \times M^1_{i-j} + W_2 \times M^2_{i-j} + \cdots + W_n \times M^n_{i-j} \) where \( W_i < W_2 < \cdots < W_n \). These factors are assigned by experience. The function can help even if there are loops in the target path.

### 2.3.2 Test Data Generation for Branch Coverage

(Harman et al., 2009) observe that the most common criterion considered is the branch coverage criterion. They further note that “Early work considered the goal to be the coverage of as many branches as possible, so the representation was a test suite and the fitness function sought to maximise coverage. However, this was found to produce solutions that achieve reasonable coverage, but not full coverage because they tended to avoid the branches that
were hard to cover.” Considerable research has been directed at improving search performance for branch coverage.

(Ferguson and Korel, 1996) introduced a search technique known as the Chaining Approach for test data generation. The Chaining Approach identifies statements on which the target structure is data dependent, and incrementally develops chains of dependencies in an event sequence. (McMinn and Holcombe, 2006) incorporated this approach into Evolutionary Testing for branch coverage. They perform a test data search for each generated event sequence. It is then possible to direct search into potentially promising, unexplored areas of the test object’s input domain. The authors claim that the technique works even with flat fitness landscape.

(Pargas et al., 1999) present a goal-oriented technique for automatic test data generation using a genetic algorithm which is guided by the control dependencies in the program for achieving statement and branch coverage. The genetic algorithm conducts its search by constructing a new test data from previously generated test data with high fitness. The algorithm guides the search direction by using the program’s control dependence graph. To improve search performance, the implementation uses multiple processors and load balancing. (Michael et al., 1997; McGraw et al., 1998; Michael et al., 2001) discuss the use of genetic algorithms for automatic software test data generation for branch coverage. Their research extends previous work on dynamic test data generation. In their work, the function is minimized by using one of two genetic algorithms in place of the local minimization techniques used in earlier research. They have implemented a tool named GADGET to generate test data which uses a branch table to keep a track of all the branch conditions for both of their true and false parts. (Wegener et al., 2001) focus on test data generation using several structural test coverage criteria using evolutionary approaches. They observe that all the test data generation techniques focus on only single test criteria at a time. (Bottaci, 2003) uses evolutionary algorithm for covering specific branches and on defining different cost functions for test data generation. (Korel et al., 2005) use testability transformation to maximize branch coverage. They use data dependence analysis for identification of those statements which affect computation of the fitness function associated with the target statement and only those statements are used to explore different ways of fitness computation. Executions in the transformed program are used to guide the search in the original program to find an input on which the target statement is executed. (Xiao et al., 2007) obtain test cases for the condition-decision coverage using the metaheuristic techniques of Genetic Algorithm, Simulated
Annealing, Genetic Simulated Annealing and Simulated Annealing with Advanced Adaptive Neighbourhood. (D’iaz et. al, 2008) use Tabu search algorithm for automatic test data generation. A cost function is used for intensifying the search and for diversifying the search. Memory and backtracking are used to avoid getting stuck in local minima. Evaluation of the generator was performed using complex programs under test and large ranges for input variables. (Chen et al., 2009) compare the branch distance with extended hamming distance for a standard benchmark program of triangle classifier problem and show that the branch coverage criterion gives promising results on comparing with the path coverage criterion. (Tan et al. 2009) introduce Annealing Immune Genetic Algorithm (AIGA) for software testing to improve coverage and reduce number of generations consumed to achieve the desired objective. (Gross et al. 2009) describe a method of evolutionary testing, typically genetic algorithm for pointers and arrays to achieve maximum branch coverage but with minimum number of test data generated.

2.3 Open Problems and Thesis Objectives

Although the literature highlights many shortcomings, those which have been addressed in this dissertation are:

1 One of the problems faced in generating test data for branch coverage using a population based metaheuristic technique such as genetic algorithm is that when a branch is chosen as the target for coverage, it may happen that none of the individuals (individuals encode input test data) in the population encode inputs for which the execution path reaches the predicate node of the target branch, i.e., a critical branch is taken that causes the predicate node of the target branch to be missed in an execution of the program. Existing approaches deal with the problem in different indirect ways. The idea of branch ordering has not been explored.

2 The idea of exploiting and comparing different metaheuristic techniques for branch coverage using branch ordering and other enhancements has not been explored.

3 Combinational approaches to fitness function design have not been explored. Possible designs could explore combining both path and branch approaches to achieve branch coverage.

4 Most approaches to fitness function design define the test data generation problem as a minimization problem. The maximization approach has not been explored.
Structured parallel approaches to test data generation exist, but the idea of using parallel islands of search together with branch selection has not been explored.

It is these ideas that we explore in this thesis. A detailed survey on the above problems is presented in subsequent sections in this chapter.

2.4 Review of Related Work

This section reviews related work on the problems identified in the previous section.

2.4.1 Branch Selection

One of the problems faced in generating test data for branch coverage using a population based metaheuristic technique such as genetic algorithm is that when a branch is chosen as the target for coverage, it may happen that none of the individuals (individuals encode input test data) in the population encode inputs that reach the predicate node of the target branch in an execution, i.e., a critical branch is taken that causes the predicate node of the target branch to be missed in an execution of the program (Ferguson and Korel, 1996; McMinn and Holcombe, 2006). In order to deal with this problem, Michael et al. (Michael et al., 2001) postpone the selection of the branch for coverage and Baresel et al. (Baresel et al., 2002) have described the design of fitness functions to guide search. The usual approach adopted is to compute the individual fitness in a way that it incorporates information about how close was the input test data in reaching the target of interest, called approach level, and combine this with branch distance data which reflects how close was the sibling branch, of the critical branch, was to be taken and which would have actually taken the traversal closer to the target branch. Accordingly, the fitness of an individual is computed as approach level + normalized branch distance (McMinn, 2004; Harman and McMinn, 2010; Arcuri, 2010, McMinn 2011). This is elaborated further in the next section. The possibility of selecting target branches in a specific sequence and augmenting the meta-heuristic process in a way that from the step (generation) a target is selected to the generation that it is covered, the current population has at least one individual that encodes inputs for a path that includes the sibling branch of the target branch, has not been explored. We hypothesize that this should result in better coverage and performance. Harman et al. (Harman et al., 2009) note that since 1995, there has been an upsurge in the works in search based test data generation based on the achievement of branch
coverage and cite a number of references, but do not mention the idea of branch ordering in their report. McMinn (McMinn 2011) also focuses on the design of fitness functions.

2.4.2 Fitness Function Design

In program-based testing, flag variables, enumeration variables, unstructured control flow and state behaviour make search difficult. For example, in the program below,

```plaintext
1  if ( d== 0 )
2       flag = 0;
3  if (flag)
4       result = 0;
5  else
6       result = n / d;
```

Statement 4 can be executed only when the value of $d$ is zero. An objective function based on the branch condition in line 3 gives no guidance for search and in such cases, metaheuristic search performs no better than random search (McMinn and Holcombe, 2006). Program transformations, substitution by predicates used in assigning values and application of an extended chaining approach have been explored (Baresel and Sthamer, 2003; McMinn and Holcombe, 2006; Wappler et al., 2007). Liu et al. (Liu et al., 2005) develop an Evolutionary Testing (ET) System for automatic testing in presence of flag conditions, adapting fitness to cater for flags.

Taking an alternate approach, (Wegener et al., 2001; Baresel et al., 2002) have explored the design of fitness functions with the object of guiding search. Although they discuss a number of strategies, the strategy of combining an approach level and branch distance has gained popularity. The fitness function to be minimized (McMinn and Holcombe, 2006) is derived from the structural target or test goal of interest and is made up of two components - the approach level and the branch distance. The approach level assesses how close an input vector is to covering the current structural target of interest on the basis of the execution path taken through the program’s control structure. A critical branch is a branch which leads to the
structural target of interest being missed in a path through the program. Figure 2.2 from (McMinn and Holcombe, 2006) illustrates critical branch and approach level computation. At the point where control flow takes a critical branch, the branch distance is calculated. The branch distance reflects how close the alternative branch was to being taken and is computed using the values of the variables or constants involved in the predicates used in the conditions of the branching statement. This is also illustrated in Figure 2.2. (Bottaci, 2003) uses predicate expression cost functions to guide evolutionary search for test data. He shows that the set of commonly used cost functions for the satisfaction of logical predicates (the min-max functions) perform poorly in certain cases. An alternative set of cost functions (the ratio-sum functions) is proposed which overcome these specific problem cases.

Figure 2.2. The Minimization Approach to Fitness Computation (McMinn and Holcombe, 2006)

(Harman et al., 2007a) introduce a multiobjective branch coverage strategy. They consider the twin objectives of branch coverage and dynamic memory consumption for both real and synthetic programs. Results of application of several multiobjective algorithms show that Pareto optimal search can yield insights into the trade-offs between the two simultaneous objectives. However, an approach combining different criteria such as branch and path coverage has not been explored.

2.4.3 Test Data Generation with Structured Evolutionary Algorithms

26
Standard genetic methods are based on a single population, and computation is usually initiated on a new generation only after the old one has been replaced. The single population or panmictic, genetic algorithm has been successfully applied to different problems in many application domains (Luque and Alba, 2011). However, with the increasing demands placed on genetic algorithms, such as searching large search spaces, costly evaluation of fitness functions and using large population sizes and with problems such as getting trapped in sub-optimal regions, structured genetic algorithm models that are amenable to parallelism have been explored for different problems and test data generation in particular (Paz, 2000; Nowostawski and Poli, 1999; Alba and Chicano, 2008). Structured GA involving parallelism with multiple populations permits speciation, a process by which different populations evolve in different directions, which lead to search different from the panmictic GA (Nowostawski and Poli, 1999).

The problem of test data generation with parallel genetic algorithm has been addressed in many works. (Pargas et al., 1999) present a technique for automatic test data generation with genetic algorithm. In their technique parallel processing is used to distribute the task of executing the instrumented program on a single test case among as many processors available. It can also be employed to speed up search. Although parallelism is available, the use is not essential in their technique. (Misailovic et al., 2007) present new algorithms for parallel search and test generation in Korat. Their algorithms partition the predicate's input space such that each worker explores only its assigned partition. Further, they present algorithms for parallel test execution, in particular when the execution immediately follows Korat's test generation, with no storing of the generated test inputs on the disk. (Alba and Chicano, 2008) analyze the application of parallel and sequential evolutionary algorithms to the automatic test data generation problem. The parallel implementation uses five islands connected with an asynchronous unidirectional ring topology. Individuals are migrated periodically and inserted into the target subpopulation if it is better than the worst individual in that subpopulation. At the beginning of the search, the test data generator seeds the population with one individual (program input) reaching the condition associated with the current partial objective. The test data generator always selects a partial objective with an associated condition reached by a previous input. (Baars et al., 2011) present an overview of search-based testing and discuss some challenges remaining to make search-based techniques applicable to the Future Internet. In this context they mention that more research is required to utilize GPGPU cards for parallel search-based testing.
Multi-population approaches have also been developed for test data generation with genetic algorithm. (Alshraideh et al., 2010) present a multiple-population genetic algorithm for branch coverage test data generation. Their approach utilizes acyclic predicate paths of the program’s control flow graph containing the target branch as goals of separate search processes using distinct island populations in order to increase the search effectiveness. They show with experiments on programs with moderate number of branches that the proposed multiple-population algorithm outperforms the single-population algorithm significantly in terms of the number of executions, execution time, time improvement, and search effectiveness. (Chen and Zhong, 2008) describe a MATLAB implementation of a multi-population genetic algorithm which selects individuals for free migration based on their fitness values. Using a triangle classifier as program under test, they show that their approach can generate path-oriented test data more effectively and efficiently than a panmictic GA. (Deepak and Samuel, 2009) design a multi-population genetic algorithm using uniform crossover. Random migration is used and individuals are added to the existing subpopulation.