Multi-Objective Test Suite Optimization for Event-B Models

Ionut Dinca

University of Pitesti, Department of Computer Science
Str. Targu din Vale 1, 110040 Pitesti, Romania
ionut.dinca@upit.ro

Abstract. Event-B is a formalism that is used in modeling and proving the consistency of complex systems. The method has been successfully used in the development of several complex real-life applications. For complementing the theorem-proving and model-checking tools developed for Event-B models, test suite generation methods have been recently introduced as research theme. In order to optimize the large test suites produced by the existing approaches, in this paper the test suite optimization problem is introduced for Event-B models. However, there exist many optimization criteria in real-life testing problems. Given that, six specifically multi-objective test suite optimization problems are defined. Two modern Multi-Objective Evolutionary Algorithms are used for solving them: NSGA-II [6] and SPEA-2 [18]. The experiments have been conducted using five test suites generated from two industrial inspired Event-B models (five different machines).

1 Introduction

Event-B [1] is a formal modeling language for reliable systems specification and verification which was introduced about ten years ago and widely used in industrial projects. The Event-B formalism is supported by a mature tool called *Rodin*[1] which offers different capabilities such as theorem-proving, composition or model-checking of Event-B models.

Recently, there has been an increasing interest for automatically test suite generation for Event-B models [15, 7]. Most approaches generate a large number of test cases for a particular model until a test adequacy criterion is achieved. For example, the ProB tool [13] (available in the Rodin platform) can be used to explore the state space of Event-B models, verify various properties using model-checking and generate test cases along the traversal using certain coverage criteria (e.g. event coverage). This approach has been applied to models from the business application area in [15].

The cost of executing, storing, and maintaining these large test suites can be reduce through *test suite optimization* techniques. The test suite optimization produces a subset of the initial test suite that preserves the original test

[1] http://sourceforge.net/projects/rodin-b-sharp

adequacy criterion by removing the redundant test cases with respect to the considered criterion. However, in real testing problems, there exist multiple test criteria, because a single ideal criterion is simply impossible to be formulated and achieved. Harman argues in his recently paper [9] that single-objective test suite optimization is not useful in practical, because testers typically have many different objectives. For example, a frequently optimization problem is to produce a minimal test suite which achieves maximal coverage of the model entities with a minimal execution cost.

This paper introduces for the first time the multi-objective test suite optimization problem for Event-B models. Due to the complexity of this problems (it is exponentially related to the original test suite size), we chose the Multi-Objective Evolutionary Algorithms for solving them.

The primary contributions of this paper are as follows:

- The paper introduces a multi-objective formulation of test suite optimization problem for Event-B models. Six specifically test suite optimization problems were proposed: minimize the size of test suite, minimize the number of executed events, minimize the longest execution path, minimize the execution time, maximize the distribution quality and balance the lengths of the paths while the longest path is minimized. For all this problems, the test adequacy criterion is the event coverage. The mathematical formulations of this problems facilitate the using of multi-objective evolutionary algorithms for solving them.
- In order to increase the confidence, the paper uses two modern Multi-Objective Evolutionary Algorithms for solving the above optimization problems: Non Dominating Sorting Genetic Algorithm (NSGA-II) [6] and Strength Pareto Evolutionary Algorithm 2 (SPEA-2) [18].
- The paper also chose a total of 5 test suites generated from two industrial inspired Event-B models (five different Event-B machines under test) as subjects for the test suite optimization problems.

In the remainder of the paper we describe the Event-B framework (Section 2), introduce the test suite optimization problem for Event-B models (Section 3), mathematically define the six different test suite optimization problems (Section 3.1), present the multi-objective evolutionary algorithms (Section 3.2), describe our experiment set up and results (Section 4), and draw the conclusions (Section 5).

2 Event-B and Test Suite Generation

Event-B is a formal method [1] for modeling the states and behavior of a system in order to prove its consistency. The states are modeled by global variables while the behavior is modeled by events. Events transform the system from a state to another state by updating the values of variables. Event-B uses mathematical proof based on set theory and logic to ensure the consistency of modeled system.

The components of an Event-B model are grouped in two categories: *Contexts* and *Machines*. Contexts contain types and constants (static parts of system) while Machines contain variables and events (the dynamic parts).

An event has *guards*, *actions* and optionally *parameters*. The guards represent the enabling conditions of the event while actions determine how specific variables change as a result of the event execution. Parameters are local variables whose values can be used for updating the global variables. The general form of an event is

$$\text{Event} \mathrel{\widehat{=}} \textbf{any } p \textbf{ where } G(p,v) \textbf{ then } S(p,v) \textbf{ end},$$

where p is the set of local parameters, v is a set of global variables appearing in the event, G is a predicate over p and v, called the guard and $S(p,v)$ represents a substitution. If the guard of an event is false, the event cannot occur and is called disabled. The substitution S describes how the global variables in the set v are modified. The values of the global variables are constraint by *invariants* which are properties of the system that should be preserved during system execution. The execution of a model starts with a special event which initializes the global variables. At each execution step the set of enabled events (for which the guards are satisfied) is computed and one enabled event is non-deterministically chosen to be executed (all its actions are simultaneously executed).

The Event-B development process is based on *refinement*: a system is modeled as a series of successive refinements, starting with an abstract representation of the system (the details are ignored). Details are added gradually to the abstract model. A refinement step introduces new functionality (new events) or add details of current functionality (a detailed version of an existing event). From a given machine, M_1, a new machine, $M2$, can be built as a refinement of M_1. Therefore this model of development produces refinement chains of Event-B machines.

Our approach for test suite generation. Given an Event-B machine M with $E = \{e_1, e_2, ..., e_m\}$ the set of its events, a test case can be defined as a sequence of events in E that can be executed in the machine M (an execution path). Each test case begins with a special event called *INITIALISATION* which serves to initialize the global variables of the machine before starting the execution of a test case. A test suite is by definition a collection of test cases.

An approach using the explicit model checker ProB was proposed in [15] for test suite generation. It suffers from the classical state space explosion when applied to models with large variable domains. Given that, we chose to implement the algorithm from [12]. It constructs a successive set of finite approximation models for the set of Event-B executable paths up to a length ℓ. The iterative nature of the algorithm fits well with the notion of refinement from the Event-B method. A detailed presentation of this algorithm is out of the scope of this paper. We just say that the algorithm was implemented as a plug-in [2] for Rodin platform and was used to generate the subjects for our experiments.

[2] `http://wiki.event-b.org/index.php/MBT_plugin`

3 Multi-Objective Test Suite Optimization for Event-B Models

In this section we introduce the multi-objective test suite minimization problem. We adopt here the definitions from [16]. Generally, a multi-objective optimization problem can be defined as to find a vector of decision variables x, which optimizes a vector of M objective functions $f_i(x), 1 \leq i \leq M$. The objective functions are the mathematical formulations of the optimization criteria. Usually, these functions are conflicting, which means that improvements with respect to one function can only be achieved when impairing the solution quality with respect to another objective function. Solutions that can not be improved with respect to any functions without impairing another one are called *Pareto-optimal solutions*.

Formally, let us assume that, without loss of generality, the goal is to minimize the functions $f_i(x), 1 \leq i \leq M$. A decision vector x is said to *dominate* a decision vector y (we write $x \succ y$) if and only if the following property is satisfied by their objective vectors:

$$f_i(x) \leq f_i(y), \forall i \in \{1, 2, ..., M\} \text{ and } \exists i_0 \in \{1, 2, ..., M\}, f_{i_0}(x) < f_{i_0}(y).$$

The dominance relations states that a solution x is preferable to another solution y if x is at least as good as y in all objectives *and* better with respect to at least one objective. The *Pareto-optimal set* is the set of all decision vectors that are not dominated by any other decision vectors. The corresponding objective vectors are said to from *Pareto frontier*. Therefore, the multi-objective optimization problem can be defined in the following manner:

Given: a vector of decision variables, x, and a set of objective functions, $f_i(x), 1 \leq i \leq M$,

Problem: minimize$\{f_1(x), f_2(x), ..., f_M(x)\}$ by finding the Pareto-optimal set over the feasible set of solutions.

With respect to multi-criteria test suite optimization, the objective functions f_i are the mathematical descriptions of the testing criteria that must be satisfied to provide desired adequate testing of the model. In real industrial testing problems, there exist multiple test criteria, because a single ideal criterion is simply impossible to be achieved. For example, a frequently optimization problem is to produce a minimal test suite which achieves maximal coverage of the model entities with a minimal execution cost. Therefore, this is a *bi-objective minimization* test suite problem.

Formally, multi-objective test suite optimization problem can be defined in the following manner [17]:

Multi-Objective Test Suite Optimization.

Given: a test suite TS, a vector of M objective functions $f_i, 1 \leq i \leq M$

Problem: to produce a subset $T \subset TS$, such that T is a Pareto-optimal set with respect to the set of the above objective functions.

In the following, we instantiate this general multi-objective test suite optimization problem with respect to our Event-B models.

Let be an Event-B machine M for which we have generated a test suite TS. Of course, TS satisfies a set of test requirements which are expressed as a level of coverage of the model. In this paper we only consider that the test suite TS achieves the following simple coverage criterion:

Event Coverage Criterion: A test suite $TS = \{t_1, ..., t_m\}$ of m test cases for an Event-B model M is said to achieve *event coverage criterion* if and only if for each event e of the model M there exists a test case $t_i \in TS$ which covers e.

Having the above criterion in mind, we can formulate the following optimization problem:

Test Suite Minimization Problem.
Given: A test suite TS generated for a machine M with $E = \{e_1, e_2, ..., e_n\}$ the set of events, and subsets of TS, T_is, one associated with each of the e_is such that any one of the test cases t_j belonging to T_i can be used to cover e_i.
Problem: Find minimal test suite T from TS which covers all e_i.

This problem is NP-complete because it can be reduced to the minimum set-cover problem [5] in the following manner.

We recall that for us a test case $tc \in TS$ is an execution path which consists in a sequence of events from E. Let be $cov(tc) = \{e \in E | tc \text{ covers } e\}$ the set of events covered by test case tc. By definition, $cov(tc)$ is a subset of E. Therefore the solution T of the above test suite minimization problem is exactly a minimum set cover for E, because

$$\bigcup_{t \in T} cov(t) = E$$

and T is the minimal subset of TS which covers E.

Many solutions have been proposed to solve this test suite minimization problem [4, 10, 2, 14, 3]. Due to its exponential complexity, in this paper we use Multi-Objective Evolutionary Algorithms for solving it. For that, we mathematically reformulate it as a constraint bi-objective test suite optimization problem (TSO1 problem from the next section).

3.1 Optimization Criteria

Based on practical experience for industrial projects at SAP [15], we propose here different test suite optimization criteria. In order to solve the optimization problems described in this section using Multi-Objective Evolutionary Algorithms, this criteria are mathematically formulated as six different constraint multi-objective optimization problems.

TSO1-Minimizing the size of the test suite. Due to the restrictions of time, obtaining a minimal test suite which achieves maximal level of coverage is of particular interest among testers. Therefore the goal of this problem is to produce a test suite that contains the smallest possible number of test cases that achieve the same coverage (in our case, the event coverage) as the complete test suite. We formulate this problem as a constraint bi-objective optimization problem: maximize event coverage (the first objective) by a minimum number

of test cases (the second objective) under the constraint that at least a test case has been selected. The problem can be mathematically described in the following manner.

Let be $TS = \{t_1, t_2, ..., t_m\}$ the initial set of m test cases and $E = \{e_1, e_2, ..., e_n\}$ the set of the events to be covered. We recall that $cov(tc)$ is the set of events covered by the test case tc. Given an order between the elements of a set, a subset $T \subset TS$ can be mathematically represented by a binary vector $x = (x_1, x_2, ..., x_m) \in \{0,1\}^m$ with

$$x_i = \begin{cases} 1, t_i \in T \\ 0, t_i \notin T \end{cases}, 1 \leq i \leq m.$$

Therefore the constraint bi-objective test suite optimization problem to be solved is the following:

$$\text{Minimize } (f_1(x), f_2(x))$$

Subject to:

$$\sum_{i=1}^{m} x_i \geq 1 \ (T \neq \varnothing)$$

Where:

$$f_1(x) = 1 - \sum_{i=1}^{m}(x_i \cdot \frac{|cov(t_i)|}{n}) \text{ (maximize the coverage)}$$

$$f_2(x) = \frac{\sum_{i=1}^{m} x_i}{m} \text{ (minimize the size of test suite).}$$

A Pareto-optimal solution of the above problem corresponds to a minimal subset of the test suite TS which achieves a maximal level of coverage. More, we can see that $f_1 : \{0,1\}^m \to [0,1)$ and $f_2 : \{0,1\}^m \to (0,1]$. Therefore we avoid to select the empty set as a solution.

TSO2-Minimizing the number of the executed events. In order to reduce the effort of the testing process, the number of executed events from the whole test suite should be minimized. Therefore we want to obtain test suites which achieve the event coverage criterion with a minimum number of executed events. The first objective function f_1 and the constraint from the problem TSO1 remain valid. Let be $len(tc)$ the length of the test case $tc \in TS$. The second objective function f_2 which can be used to minimize the number of executed events by the subset $T \subset TS$ is

$$f_2(x) = \frac{1}{\sum_{k=1}^{m} len(t_k)} \sum_{i=1}^{m}(x_i \cdot len(t_i)).$$

TSO3-Minimizing the length of the longest execution path. The longer execution paths are harder to maintain. In this problem we control the lengths of the execution paths by minimizing the length of the longest test case. The mathematical formulation is the following:

$$\text{Minimize } (f_1(x), f_2(x))$$

Where $f_1(x)$ is the same as for TSO1 problem and

$$f_2(x) = \max\{len(t_i)|x_i = 1 \text{ and } 1 \leq i \leq m\}.$$

The second objective function f_2 is used for minimizing the length of the longest test case.

TSO4-Minimizing the execution time. We measure the execution time for each test case tc from the initial test suite TS. Let us denote by $time(tc)$ the execution time of tc. Then the execution time of a test suite $T \subset TS$ is $\sum_{tc \in T} time(tc)$. In this problem the goal is to minimize the execution time of the test suites. The first objective and the constraint are the same as for TSO1 problem. The second objective function f_2 to be minimized is

$$f_2(x) = \sum_{i=1}^{m}(x_i \cdot time(t_i)) \text{ (minimize the execution time).}$$

TSO5-Maximizing the distribution quality. In order to understand the problem proposed here, let us consider a simple example. Let be $T_1 = \{e_1e_3e_4, e_1e_2, e_3e_2e_5\}$ and $T_2 = \{e_2e_2e_4, e_1e_2, e_3e_5\}$ two test suites which cover the set of events $E = \{e_1, e_2, ..., e_5\}$. The events e_1 and e_2 are executed an equal number of times in T_1, while they are not in T_2. We say that T_1 has a better *distribution quality*. Therefore the goal is to obtain test suites with a good distribution of the events. This property is a practical requirement of users.

In the following, we propose an objective function which measures the distribution quality of a given test suite $T \subset TS$. Let be $TS = \{t_1, t_2, ..., t_m\}$ the initial test suite and $E = \{e_1, e_2, ..., e_n\}$ the set of the events. Let be a matrix A which captures the events covered by each test case tc in TS; the number of rows of A equals the number of events to be covered, n, and the number of columns equals the number of test cases in the initial test suite, m. Therefore the entries $(a_{ij})_{1 \leq i \leq n, 1 \leq j \leq m}$ of A are

$$a_{ij} = \begin{cases} k, \ t_j \text{ covers } e_i \text{ by } k \text{ times} \\ 0, \ e_i \text{ is not covered by } t_j \end{cases}, 1 \leq i \leq n, 1 \leq j \leq m.$$

Let be $x = (x_1, x_2, ..., x_m) \in \{0, 1\}^m$ the mathematical representation of the test suite $T \subset TS$. We define the matrix $D(x)$ to be

$$D(x) = A \times \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_m \end{pmatrix}$$

More exactly, $D(x)$ is a vector of n components $d_i(x), 1 \leq i \leq n$. From the definition, the entry $d_i(x) = \sum_{k=1}^{m}(a_{ik} \cdot x_k)$ of D denotes the number of times the event e_i was covered by the test suite T.

Now the mean amount of executions per event in T is exactly

$$m_T(x) = \frac{1}{n}\sum_{i=1}^{n} d_i(x).$$

If the test suite T has a good distribution of the events, we would expect $d_i(x), 1 \leq i \leq n$ values to stay near the mean value $m_T(x)$. Therefore in order to obtain a good distribution of the events we define the objective function to be minimized in the following manner:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n}(d_i(x) - m_T(x))^2.$$

Let us illustrate this definition on our simple example. We consider that $TS = T1 \cup T_2 = \{e_1e_3e_4, e_1e_2, e_3e_2e_5, e_2e_2e_4, e_1e_2, e_3e_5\}$. Then, $x_1 = (1,1,1,0,0,0)$ and $x_2 = (0,0,0,1,1,1)$ are the mathematical descriptions of T_1 and T_2 respectively. Given that, the matrix A will be

$$A = \begin{pmatrix} 1&1&0&0&1&0 \\ 0&1&1&2&1&0 \\ 1&0&1&0&0&1 \\ 1&0&0&1&0&0 \\ 0&0&1&0&0&1 \end{pmatrix}$$

and

$$D(x_1) = A \times \begin{pmatrix} 1\\1\\1\\0\\0\\0 \end{pmatrix} = \begin{pmatrix} 2\\2\\2\\1\\1 \end{pmatrix},\ D(x_2) = A \times \begin{pmatrix} 0\\0\\0\\1\\1\\1 \end{pmatrix} = \begin{pmatrix} 1\\3\\1\\1\\1 \end{pmatrix}$$

Further calculation shows that $f(x_1) = 0.24$ and $f(x_2) = 0.64$. Therefore the test suite $T1$ has a better distribution of the events.

We formulate this problem as a constraint single-objective optimization problem and search for solutions which minimize $f(x)$ subject to

$$d_i(x) \geq 1,\ 1 \leq i \leq n \text{ (each event is covered at least one time).}$$

TSO6-Balancing the lengths while minimizing the longest path. Finally, we propose here to balance the lengths of the execution paths while we keep valid the two objectives of **TSO3** problem (achieve event coverage while minimize the length of the longest path). Therefore this problem is a 3-objective test suite optimization problem. We search here for test suites which achieve event coverage by short and balanced execution paths. The third objective function can be mathematically formulated as below.

We remember that $len(tc)$ denotes the length of the test case tc. Let be $T \subset TS$ a test suite and x its mathematical description. First, we define the mean of the lengths as

$$m_T^{len}(x) = \frac{1}{|T|}\sum_{i=1}^{m}(x_i \cdot len(x_i)).$$

Table 1. Summarize the six test suite optimization problems.

Problem	Type	Constraint	Description
TSO1	bi-objective	yes	Minimizing the size of the test suite
TSO2	bi-objective	yes	Minimizing the number of the executed events
TSO3	bi-objective	no	Minimizing the longest execution path
TSO4	bi-objective	yes	Minimizing the execution time
TSO5	single-obj.	yes	Maximizing the distribution quality
TSO6	3-objective	no	Balancing the lengths + TSO3 problem

If the test suite T contains balanced execution paths, the $len(tc), tc \in T$ values will stay near the mean value $m_T^{len}(x)$. Given that, the third objective function to be minimized can be defined as

$$f_3(x) = \frac{1}{|T|} \sum_{i=1}^{m} (x_i \cdot (len(t_i) - m_T^{len}(x))^2)$$

We solve all these six test suite optimization problems using multi-objective evolutionary algorithms. In Table 1 we summarize the properties of our problems.

3.2 Multi-Objective Evolutionary Algorithms

We chose two modern and widely used Pareto efficient genetic algorithms, NSGA-II and SPEA-2[18].

NSGA-II is a multi-objective genetic algorithm developed by Deb et al [6]. The output of NSGA-II is a set of solutions which are Pareto-optimal solutions. NSGA-II differs from normal genetic algorithms in two main aspects. First, Pareto-optimality is used in the process of selection of individuals for the next generation. It performs the non-dominated sorting in each generation in order to preserve the individuals on the current Pareto-frontier into the next generation. For example, solutions on the current Pareto-frontier get assigned dominance level 0. Then, after taking these solutions out, fast-non-dominated sorting calculates the Pareto-frontier of the remaining population; solutions on this second frontier get assigned dominance level of 1, and so on. The dominance level becomes the basis of selection of individual solutions for the next generation.

The second difference concerns the problem of selecting one individual out of a non-dominated pair. In order to achieve a wider Pareto frontier, NSGA-II uses *crowding distance* for make this decision. Crowding distance measures the density of individuals near a particular individual. NSGA-II selects individuals that are far from the others.

A high level outline of the main loop of NSGA-II is presented in the **Algorithm 1**. First, in the line (1) a combined population $R_t = P_t \cup Q_t$ is formed. Then, algorithm assigns (line (2)) dominance level to individuals. Inside the loop

Algorithm1. NSGAIIMainLoop

Input: The parent population, P_t
The children population, Q_t
The population size, N
Output: The next population, (P_{t+1}, Q_{t+1})
(1) $R_t \leftarrow P_t \cup Q_t$
(2) $\mathcal{F} \leftarrow FastNondominatedSort(R_t)$
(3) $P_{t+1} \leftarrow \varnothing$ and $i \leftarrow 1$
(4) **repeat**
(5) $CrowdingDistanceAssignment(\mathcal{F}_i)$
(6) $P_{t+1} \leftarrow P_{t+1} \cup \mathcal{F}_i$
(7) $i \leftarrow i + 1$
(8) **until** $|P_{t+1}| + |\mathcal{F}_i| \leq N$
(9) $Sort(\mathcal{F}_i, \prec_n)$
(10) $P_{t+1} \leftarrow P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$
(11) $Q_{t+1} \leftarrow MakeChildrenPopulation(P_{t+1})$
(12) $t \leftarrow t + 1$

Fig. 1. Outline of the main loop for NSGA-II

(lines (4) to (8)), all the non-dominated frontiers are added to the next generation. The remaining members of the new generation (the population P_{t+1}) are chosen from subsequent non-dominated front in according to the descending order of crowding distance (lines (9,10)). The new population P_{t+1} of size N is used for selection, crossover and mutation to create a new children population Q_{t+1} ($MakeChildrenPopulation$ from line (11)). The algorithm uses a binary tournament selection operator, but the selection criterion is based on the crowded-comparison operator $\prec_n$. This operator states that between two solutions from different dominance levels the solution with better level is preferred. Otherwise, if both solutions belong to the same dominance level, then the solution that is located in a lesser crowded region is preferred.

SPEA-2 uses a regular population and an archive (an external set). The main loop is presented in **Algorithm 2.** We do not provide here a detailed description of this second multi-objective evolutionary algorithm. For a more detailed description the interested reader is referred to [18].

Solution Encodings. When using evolutionary algorithms for solving a multi-objective test suite optimization problem, we must properly encode the possible solutions of the problem. Let be $T \subset TS$ a subset of the initial test suite $TS = \{t_1, t_2, ..., t_m\}$. We use the mathematical representation $x \in \{0, 1\}^m$ of T (see Section 3.1) to encode the possible solutions. Therefore binary encoding is considered to be a natural representation for the possible solutions. The inclusion and exclusion of a test case within a subset of the initial test suite are represented by 1 and 0 respectively in a binary string (*chromosome* string).

Algorithm2. SPEA2MainLoop

Input: The population size, N
The archive size, $\overline{N}$
The maximum number of generations, T
Output: The nondominated set, A
(Step 1) **Initialization:** Generate an initial population P_0 and create the empty archive $\overline{P_0} = \varnothing$. Set $t = 0$.
(Step 2) **Fitness assignment:** Calculate fitness values of individuals in P_t and $\overline{P_t}$.
(Step 3) **Environmental selection:** Copy all nondominated individuals in P_t and $\overline{P_t}$ to $\overline{P_{t+1}}$. If size of $\overline{P_{t+1}}$ exceeds $\overline{N}$ then reduce $\overline{P_{t+1}}$ by means of the truncation operator; otherwise if size of $\overline{P_{t+1}}$ is less $\overline{N}$ then fill with dominated individuals in P_t and P_t.
(Step 4) **Termination:** If $t \geq T$ or another stopping condition is satisfied then set A to the set of decision vectors represented by the nondominated individuals in $\overline{P_{t+1}}$. Stop.
(Step 5) **Mating selection:** Perform binary tournament selection with replacement on $\overline{P_{t+1}}$ in order to fill the mating pool.
(Step 6) **Mating selection:** Apply recombination and mutation operators to the mating pool and set $\overline{P_{t+1}}$ to the resulting population. Increment generation counter ($t = t + 1$) and go to (Step 2).

Fig. 2. Outline of the main loop for SPEA-2 [18].

4 Experiments

Subjects. We conducted the experiments with a total of 5 test suite subjects of varying sizes and complexity levels. The test suites were generated from two industrial inspired Event-B models: the BepiColombo and SSFPilot models which are publicly available DEPLOY model repository[3]. The first 4 machines are different levels of refinements of BepiColombo project and the last machine is the high level of abstraction of SSFPilot model. The sizes of the machines are listed in Table 2.

The two models are summarized below:

– *BepiColombo*: This is an abstract model[4] of two communication modules in the embedded software on a space craft. The Event-B model was proposed for formal validation of software parts of BepiColombo mission to Mars[5]. The model has different levels of refinements. In the abstraction, M_0, the

[3] http://deploy-eprints.ecs.soton.ac.uk
[4] http://eprints.ecs.soton.ac.uk/22048/5/Rodin_Space_Craft.zip
[5] See http://deploy-eprints.ecs.soton.ac.uk/72/1/BepiColombo_-_Modelling_Approach.pdf and http://en.wikipedia.org/wiki/BepiColombo

Table 2. Sizes of five test suite subjects generated from two industrial inspired models (number of events, size of test suites and maximum length of test cases).

Subject	No. of ev.	Size of TS	Max. size of tcs
BepiColombo_M0	5	40	7
BepiColombo_M1	10	170	7
BepiColombo_M2	12	256	7
BepiColombo_M3	16	240	7
SSFPilot_TCTM	13	786	8

Table 3. TSO1. Average reduced sizes for optimized test suite T.

		NSGA-II		SPEA-2	
Subject	$f_2(x_{TS})$	Avg $f_2(x_T)$	Avg%	Avg $f_2(x_T)$	Avg%
BepiColombo_M0	40	1.03	97.42	1.01	97.47
BepiColombo_M1	170	7.59	95.53	8.72	94.87
BepiColombo_M2	256	28.87	88.72	30.98	87.89
BepiColombo_M3	240	26.14	89.10	27.97	88.34
SSFPilot_TCTM	786	228.42	70.93	232.5	70.41

main goal of the system is modeled. The details of the system are added through three refinement levels, M_1, M_2 and M_3. The modeling approach starts on the first level with 5 set-type variables and 5 events and ends up with 18 variables and 16 events.

– *SSFPilot*: This is an Event-B model [6] of a pilot for a complex on-board satellite mode-rich system: Attitude and Orbit Control System (AOCS). In [11] the authors present a formal development of an AOCS in Event-B modeling language. They show that refinement in Event B provides the engineers with a scalable formal technique that enables both development of mode-rich systems and proof-based verification of their mode consistency.

Results. The test suite optimization techniques attempt to reduce the test suite cost w.r.t. a given coverage criterion (event coverage in our case). Given that, the percentage reduction will be used as a measure for comparative analysis. To increase the confidence, we compare the results produced by the two algorithms: NSGA-II and SPEA-2.

We have used the multi-objective evolutionary algorithm framework jMetal [8] for our experiments. The two algorithms were configured with population size of 100. The archive size of SPEA-2 was set to the same value, 100. The stopping criterion is to reach the maximum number of generation which was set to 100. The both algorithms use the following genetic operators: *the binary tournament selection* operator, *the single point crossover* operator with probability of 0.9 and *the single bit-flip mutation* operator with the mutation rate of $1/m$ where m is the length of the bit-string (i.e. the size of the initial test suite).

[6] `http://deploy-eprints.ecs.soton.ac.uk/58/`

Table 4. TSO2. Average reduced number of executed events for optimized test suite T.

		NSGA-II		SPEA-2	
Subject	$f_2(x_{TS})$	Avg $f_2(x_T)$	Avg%	Avg $f_2(x_T)$	Avg%
BepiColombo_M0	252	8.02	96.8	8.02	96.8
BepiColombo_M1	1300	65.09	94.99	71.93	94.46
BepiColombo_M2	1977	224.42	88.65	236.34	88.04
BepiColombo_M3	1873	204.77	89.06	221.39	88.17
SSFPilot_TCTM	6554	1897.79	71.04	1931.98	70.52

Table 5. TSO3. Average length of the longest path of optimized test suite T.

Subject	NSGA-II	SPEA-2
BepiColombo_M0	4.69	4.84
BepiColombo_M1	7	7
BepiColombo_M2	7	7
BepiColombo_M3	7	7
SSFPilot_TCTM	8	8

Table 6. TSO4. Average execution time (in seconds) of optimized test suite T.

		NSGA-II		SPEA-2	
Subject	$f_2(x_{TS})$	Avg $f_2(x_T)$	Avg%	Avg $f_2(x_T)$	Avg%
BepiColombo_M0	4.6	0.13	97.07	0.14	96.95
BepiColombo_M1	48.43	1.88	96.11	2.16	95.54
BepiColombo_M2	130.16	12.39	90.48	13.40	89.70
BepiColombo_M3	204.28	20.43	89.99	22.13	89.16
SSFPilot_TCTM	197.80	50.78	74.32	51.38	74.02

Table 7. TSO5. Average distribution quality of optimized test suite T.

		NSGA-II		SPEA-2	
Subject	$f(x_{TS})$	Avg $f(x_T)$	Avg%	Avg $f(x_T)$	Avg%
BepiColombo_M0	520.24	0.16	99.96	0.16	99.96
BepiColombo_M1	8771.4	17.03	99.80	22.45	99.74
BepiColombo_M2	19840.90	238.98	98.79	270.26	98.63
BepiColombo_M3	14432.43	169.14	98.82	191.42	98.67
SSFPilot_TCTM	166187.40	13251.76	92.02	13667.67	91.77

Table 8. TSO6. Average balancing values of the lengths of optimized test suite T.

		NSGA-II		SPEA-2	
Subject	$f_3(x_{TS})$	Avg$f_3(x_T)$	Avg%	Avg$f_3(x_T)$	Avg%
BepiColombo_M0	2.16	0.00	100	0.00	100
BepiColombo_M1	1.81	0.21	88.27	0.22	87.52
BepiColombo_M2	1.57	0.33	78.41	0.34	77.87
BepiColombo_M3	1.62	0.36	77.76	0.37	77.04
SSFPilot_TCTM	2.21	1.15	47.96	1.17	47.05

For each test suite subject, each optimization problem and each algorithm, 100 independent runs were performed. The results are presented in Tables 3-8. To compare the results, we computed for each problem the specific objective function values for the initial test suite. For example, the column $f_3(x_{TS})$ from the Table 8 indicates the values of the third objective function of the problem TSO6 when computed for the initial test suite TS. Otherwise, in each table, the average values of specific objective functions of the solutions are indicated. As shown in the tables, the results of the two algorithms are comparable. We obtained high values for the percentage reduction of test suite because of the simplicity of the event coverage criterion.

5 Conclusions

In this paper the multi-objective test suite optimization problem for Event-B testing was introduced. Different optimization criteria were proposed and the resulted problems were solved using two modern multi-objective evolutionary algorithms. For all optimization problems the considered test adequacy criterion was the event coverage. All our optimization problems can be easily formulated in a more general framework: a test suite T must meet a set of n requirements $\{r_1, r_2, ..., r_n\}$ to provide the desired 'adequate' testing of the model. We will consider in the future more complex coverage criteria.

Acknowledgment This work was supported by the European project DEPLOY (EC-grant no. 214158).

References

1. Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering.* Cambridge University Press, 2010.
2. H. Agrawal. Efficient coverage testing using global dominator graphs. In *Proceedings of the 1999 Workshop on Program analysis for software tools and engineering*, pages 11–20, 1999.
3. J. Black, E. Melachrinoudis, and D. Kaeli. Bi-criteria models for all-uses test suite reduction. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pages 106–115, Edinburgh, Scotland, United Kingdom, 2004.
4. V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3), 1979.
5. T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 2001.
6. K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer LNCS No. 1917., 2000.

7. Ionut Dinca, Alin Stefanescu, Florentin Ipate, Raluca Lefticaru, and Cristina Tudose. Test data generation for Event-B models using genetic algorithms. In *Proc. of 2nd International Conference on Software Engineering and Computer Systems*, CCIS. Springer, Berlin, 2011.
8. J.J. Durillo, A.J. Nebro, and E. Alba. The jMetal framework for multi-objective optimization: Design and architecture. In *CEC 2010*, pages 4138–4325, Barcelona, Spain, July 2010.
9. M. Harman. Making the case for MORTO: Multi Objective Regression Test Optimization. In *Proceedings of the 1st International Workshop on Regression Testing (Regression 2011)*, Berlin, Germany, 2011.
10. M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, 1993.
11. A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Developing mode-rich satellite software by refinement in Event B. In *15th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2010)*, CCIS, Antwerp, Belgium, 2010.
12. Florentin Ipate. Learning finite cover automata from queries. *Journal of Computer and System Sciences*, 2011. In Press. Online at: http://doi:10.1016/j.jcss.2011.04.002.
13. Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *Int. J. Softw. Tools Technol. Transf.*, 10(2):185–203, 2008.
14. M. Marre and A. Bertolino. Using spanning set for coverage testing. *IEEE Transactions on Software Engineering*, 29(11):974–984, 2003.
15. Sebastian Wieczorek, Vitaly Kozyura, Andreas Roth, Michael Leuschel, Jens Bendisposto, Daniel Plagge, and Ina Schieferdecker. Applying model checking to generate model-based integration tests from choreography models. In *Proc. TESTCOM'09*, volume 5826 of *LNCS*, pages 179–194. Springer, 2009.
16. S. Yoo and M. Harman. Pareto efficient multi-objective test-case selection. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2007)*, pages 140–150. ACM Press, 2007.
17. S. Yoo and M. Harman. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software*, 83(4):689–701, 2010.
18. E. Zitzler, M. Laumanss, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. *Tech. Rep.*, 103, 2001.