Property-Oriented Testing based on Simulated Annealing

Olfa Abdellatif-Kaddour, Pascale Thévenod-Fosse and Hélène Waeselynck

LAAS-CNRS

7 Avenue du Colonel Roche

31077 Toulouse Cedex 4 – France

+33/ 5 61 33 62 97

{kaddour, thevenod, waeselyn}@laas.fr

ABSTRACT

Property-oriented testing uses the specification of a property to drive the testing process. The aim is to validate a program with respect to a target property, that is, to exercise the program and observe whether the property is violated or not. We define a test strategy for safety properties in cyclic control systems. It consists of the stepwise construction of test scenarios. Each step explores possible continuation of the dangerous scenarios found at the previous step, using heuristic search techniques. The feasibility of the strategy is illustrated on a steam boiler case study, the target property being the "non explosion" of the boiler in presence of faults in the physical devices. As a first investigation, the strategy is instantiated with one heuristic search technique, simulated annealing, whose efficiency is analyzed in comparison with random sampling. The experimental results allow us to propose a revised version of the basic simulated annealing algorithm. Using the revised algorithm, the proposed strategy applied to the steam boiler is promising.

Keywords

Software testing, high-level safety property, test data generation, optimization technique, experiments.

1. INTRODUCTION

This paper investigates a strategy for property-oriented testing of cyclic real-time control systems. Property-oriented testing uses the specification of a property to drive the testing process. The aim is to validate a program with respect to a target property, that is, to exercise the program and observe whether the property is violated or not. The type of property we are interested in is any high level requirement related to the most critical failure modes of the control system, as identified in a preliminary risk analysis. Such a high-level property is required from the whole system, rather than from some delimited part of it. Formal verification would need a detailed model reproducing the interactions between the control program and its controlled environment, accounting for the physical devices, the physical laws governing the controlled process, as well as the possible occurrences of physical faults in the devices. Such an exact analysis of system behavior is generally unfeasible, and testing may be seen as a pragmatic alternative, provided that the testing environment is as close as possible to the operational environment (i.e., control program run on the target hardware and connected to a simulator of the physical environment, with facilities for fault injection). However, there is still the difficult issue of test data selection. To address this issue, we propose a test strategy consisting of the stepwise construction of test scenarios, and making use of heuristic search techniques. Two considerations led to this strategy.

The first one is related to the sequential (with memory) behavior exhibited by control systems. It is expected that property violation will occur after execution of a particular trajectory in the input domain of the system, rather than just a specific point of the input domain: the system will progressively evolve towards property violation. Hence, the test scenarios we search for are sequences of input data. There is no way to know *a priori* how long it could take to reach property violation. The proposed strategy involves several successive steps to allow a progressive exploration of longer test scenarios (until property violation, if possible).

The second consideration is the need for guiding exploration at each step of the strategy. The guiding process has to favor the choice of those test scenarios that will be the most "stressing" with respect to the target property. Given the intractability of a detailed analysis of system behavior, and the large input domain to be considered at each step, we propose to investigate the power of modern heuristic search techniques as potential solutions to this problem.

Modern heuristic search techniques [8] have proven useful to solve complex optimization problems. Their generality makes them capable of very wide application, and they are able to operate as black boxes, hence requiring little knowledge about the problem to be solved. Two representatives of these techniques, namely genetic algorithms and simulated annealing, have gained recent attention in the field of software testing. They have been used to automate the generation of test data for a number of testing problems: instruction and branch testing [4, 6, 7, 12], determination of Worst-Case Execution Time [3, 12], exception testing [11], conformance testing [10], and robustness testing [9]. Whatever the testing problem, the search process involves a series of program executions (e.g., a series of trials is necessary before finding one input data that will cause the execution of a target program branch). Each execution allows a candidate solution to be evaluated, using some problem-dependent function called *fitness function* in the framework of genetic algorithms and *cost* *function* in simulated annealing algorithms. The results of the evaluation function are used in the search algorithms to guide the generation of new candidate solutions.

Yet, the efficiency and effectiveness of heuristic techniques for automatic generation of test data is still a questionable issue. It is worth noting that the literature on testing reports examples for which heuristic search techniques did not perform very well. In [3], genetic algorithms were applied to WCET testing: it was observed that the search process was effective for testing programs with very low structural complexity (i.e., low nesting and sequencing), but became more and more unreliable as complexity increased. In the general literature on heuristic techniques (e.g., [2, 8]), it is explained that performance depends on the parameters chosen for the implementation of the generic search algorithms for a particular problem. Making good choices relies on empirical studies experimenting with alternative implementations for this particular problem. Indeed, a large number of variants of the algorithms have been proposed, including hybrids combining elements of different search techniques. In this paper, as a first investigation, the proposed test strategy is instantiated with one search technique, namely simulated annealing.

Section 2 introduces the principles of simulated annealing, as well as previous work applying it for software testing. Section 3 defines the general principles of the proposed strategy. The example used to experiment with this strategy is the boiler case study [1], described in Section 4. The target safety property is the non-explosion of the boiler. Sections 5 and 6 present the results we obtain by using two successive implementations of simulated annealing. In each case, the performance of the search process is evaluated according to two criteria: 1) effectiveness, that is, ability to find test scenarios that violate the target safety property; 2) efficiency, that is, ability to outperform a blind search. Section 7 concludes on this empirical investigation.

2. BACKGROUND AND RELATED WORK

Simulated annealing is an optimization technique based on the idea of neighborhood search. Since Kirkpatrick's original paper [5], a vast amount of research has been carried out to apply it to a variety of disciplines. As regards software testing, promising pioneering work has been recently reported (see e.g., [10]). Section 2.1 summarizes the motivation and the principle of simulated annealing. Section 2.2 concentrates on previous work related to its application to software testing.

2.1 Basic Simulated Annealing

This informal outline of simulated annealing is based on the synthesis presented in [2].

Suppose that we have a minimization problem over a set of feasible solutions S and a cost function $f : S \rightarrow \Re$ which can be calculated for all $s \in S$. Simulated annealing is a variant of the more traditional descent methods of local optimization or neighborhood search. In these methods, a subset of feasible solutions is explored by repeatedly moving from the current solution to a neighboring solution, and the only authorized moves are in a direction of improvement (only a neighbor giving rise to a decrease in the cost function is accepted as a new current solution). A main disadvantage of such descent strategies is that the solutions obtained are totally dependent on the starting solution(s) employed. As a result, these strategies often yield convergence to a local rather than a global minimum. This is the motivation of simulated annealing, which offers a way of alleviating the problem by allowing some uphill moves in a controlled manner.

The simulated annealing algorithm is similar to the random descent method in that a neighborhood structure on the solution space is defined, and the neighborhood of the current solution is sampled at random. It differs in that a neighbor giving rise to an increase of the cost function may be accepted and this acceptance depends on a control parameter – called *temperature*, and on the magnitude of the increase.

Figure 1 shows the principle of the basic simulated annealing algorithm. The search is driven by a cost function f with the aim of minimizing it. The algorithm works by selecting randomly a new solution s, which is in the neighborhood of the current solution s_0. Better solutions with respect to the objective ($\delta \leq 0$) are always accepted. Moves to inferior solutions ($\delta > 0$) are allowed but their frequency is governed by a probability function, which depends on the temperature t and the magnitude of the increase δ. Starting from t_0, the temperature is gradually reduced during the search (function α), to progressively decrease the acceptance rate of inferior solutions. For a given value t, a number *nrep* of iterations is performed. The manner and rate at which t is reduced, called the *cooling schedule*, is governed by *nrep* and α.

```
Select an initial solution $s_0$;
Select an initial temperature $t_0 > 0$;
Select a temperature reduction function $\alpha$;
REPEAT
  REPEAT
      Randomly select $s \in N(s_0)$;
      $\delta = f(s) - f(s_0)$;
      IF $\delta \leq 0$
      THEN $s_0 = s$
      ELSE
          Generate random x uniformly in range (0, 1);
          IF $x < exp(-\delta/t)$ THEN $s_0 = s$;
  UNTIL iteration_count = nrep;
  Set $t = \alpha(t)$;
UNTIL stopping condition = true.
```

Figure 1. Simulated annealing for a minimization problem with solution space *S*, cost function *f* and neighborhood structure *N*

A number of decisions must be made to implement the general simulated annealing algorithm for the solution of a particular problem. They can be divided into two categories:

- *Generic decisions*, which involve the parameters of the annealing algorithm itself, that is: initial temperature t_0, cooling schedule (*nrep* and α) and stopping condition.
- *Problem-specific decisions*, which are concerned with the solution space S, the neighborhood structure N and the cost function f.

Both types of decisions will affect the speed of the algorithm (efficiency) and the quality of the obtained solutions (effectiveness). Unfortunately, it is not possible to set down a series of rules that will always define the best choices for a given problem. However, based on a review of some theoretical results related to the convergence properties of the simulated annealing, the following 'qualitative' guidelines on the decision making process are argued in [2].

As regards the *generic decisions*, if the final solution is to be independent of the starting solution, t_0 must be 'hot' enough to allow an almost free exchange of neighboring solutions. Some time should be spent at high temperatures when the rate of acceptance is high. Then a (comparatively) long time should be spent at lower temperatures to ensure that a local minimum has been fully explored. An appropriate rate at which t is reduced may be achieved by using either a large number of iterations *nrep* at few temperatures or a small value of *nrep* at many temperatures. Finally, empirical and theoretical results suggest that the manner of cooling (function α) is not as important as the rate. Hence, for a new application, it is probably better to start off with one of the two most widely used functions: either a geometric function $\alpha(t) = at$ (where $0.8 < a < 0.99$), or a function $\alpha(t) = t/(1 + \beta t)$ where β is a small value.

As regards the *problem-specific decisions*, some desirable properties are outlined. The first consideration in determining the neighborhood structure is to ensure that every solution should be reachable from every other (this reachability condition is sufficient to prove convergence). Several results also demonstrate that the number of required iterations depends on the size of the solution space, which suggests that this should be kept as small as possible. To do this in cases of problems with large solution spaces, a practical way to proceed is to divide the solution space into several subsets of smaller sizes, and associate a specific sub-problem with each solution subspace (see Section 4.2). In addition to this, it is also useful to aim for reasonably small neighborhoods. This enables a neighborhood to be searched adequately in fewer iterations, but conversely means that there is less opportunity for dramatic improvements to occur in a single move. Thus, there must be a compromise here but, in general, small simple neighborhoods are preferable to large complex ones.

In conclusion, when faced with a new problem, Dowsland's recommendation is to start with an implementation of the basic algorithm following the previous guidelines, with the most obvious neighborhood structure, a geometric cooling rate of 0.95, and a starting temperature determined by few experiments. If the basic algorithm fails to produce the required results after a reasonable amount of experiments, then – based on the analysis of the results of the first experimentation – improvements of the basic algorithm may be proposed to adapt it to the type of problem under study. Indeed, a number of modifications have already been proposed for a number of different problems [2]. We will return to these modifications in Section 5.3, after having analyzed first experimental results related to the effectiveness and the efficiency of the basic algorithm applied to property-oriented testing.

2.2 Application to Software Testing

Pioneering work based on the use of simulated annealing for test data generation is reported by Tracey et al. [10, 11], who defined a generalized test data generation framework. This framework can incorporate a number of testing criteria, including conformance testing with respect to a specification. This is a form of property-oriented testing: the target properties involve pre- and post-conditions, to be locally satisfied by a procedure under test.

To introduce the approach, [10] gives the example of the simple wrap-round increment routine shown in Figure 2. This routine should count from 0 to 10 modulo 11. $N^{\sim}$ refers to the input value and N to the output value.

```
Procedure Wrap_Inc (N : in out Integer);
--# pre N~ ≥ 0 and N~ ≤ 10;
   ...
--# post ( N~ < 10 → N = N~ + 1 ) and ( N~ = 10 → N = 0 );
end Wrap-Inc
```

Figure 2. Specification of wrap-round increment routine

The objective is to violate this property, that is, to find a test input data (corresponding to a point in the input domain), which satisfies the pre-condition before execution and whose output violates the post-condition after execution. The objective is to obtain the pre-condition and the negated post-condition. The first step of Tracey's approach is to convert the objective to DNF, which leads to two constraints as follows:

$$N^{\sim} \geq 0 \wedge N^{\sim} \leq 10 \wedge (N^{\sim} < 10 \wedge N \neq N^{\sim} + 1) \quad (1)$$

$$N^{\sim} \geq 0 \wedge N^{\sim} \leq 10 \wedge (N^{\sim} = 10 \wedge N \neq 0) \quad (2)$$

The search process then attempts to find a solution to each constraint in turn, using simulated annealing.

As regards the neighborhood structure, they consider the data type of each input variable. For example, the neighbor of an integer value is this value ± some proportion of allowed range. As regards the cooling schedule, the framework implements a simple geometric variation of the temperature.

The overall cost associated with each constraint is equal to the sum of costs associated with the terms of the constraint. For example, the cost associated with term $N^{\sim} \leq 10$ is defined as follows:

$$Cost = \begin{cases} 0 & \text{if } N^{\sim} \leq 10 \\ (N^{\sim}-10)+K & \text{if } N^{\sim} > 10 \end{cases}$$

k being a positive constant which penalizes the solutions that do not fulfill the objective.

The stopping condition is to reach either a zero cost or a predefined maximum number of iterations. Reaching a zero cost means succeeding in finding a test input data that violates the target property.

The first results seem to be promising. Our aim is to extend the use of this optimization technique to sequential problems, specifically cyclic real-time control systems, and high-level properties.

3. PROPOSED TEST STRATEGY

We first explain how the testing problem is formulated in terms of a search problem (Section 3.1). Compared with Tracey's work, this requires (1) changing the granularity of the solution space, i.e., from input data for combinatorial systems to sequences of input data for sequential systems; and (2) adapting the cost function to make it appropriate to high-level properties. Then Section 3.2 presents the stepwise construction of test scenarios based on the search results.

Given a particular control system and a target property, the proposed strategy is expected to be conducted in close connection with domain experts, aiding both to formulate the problem and to analyze the obtained scenarios.

3.1 Formulation of the Search Problem

A preliminary analysis determines the objective of the search. In addition to the violation of the target property, a set of *dangerous situations* is identified. The dangerous situations can be elaborated based on the safety analyses that accompany system development. They characterize intermediate states that are of interest when exploring progressive evolution towards property violation. So, the heuristic algorithm will not only aim to reach a violation of the target property, but also to reach a dangerous situation, i.e., to get close to a violation.

The *solution space S*, i.e., the set of input sequences to be sampled during the search, has also to be determined. It is modeled in terms of both the functional activity and some fault hypotheses. The combination of faults with the functional activity is likely to yield a large solution space. Then, according to the guidelines presented in Section 2.1, it may be decomposed into smaller subspaces. They correspond to classes of test sequences to be independently searched. As an example, the decomposition can be done according to the system operating modes. Finer analysis of system behavior may be considered, as long as it remains tractable.

Based on the previous analyses, the testing problem can be formulated as a collection of optimization problems, one for each class of test sequences. In each case, the objective is defined as a set of sub-objectives $\{1,...,n\}$ corresponding to either the property violation during the execution of a test sequence s, or the achievement of a dangerous situation at the end of the execution of s. The cost function f_j ($f_j \geq 0$) associated with sub-objective j measures the "distance" between the results of the execution of s and sub-objective j, with $f_j(s) = 0$ if and only if sub-objective j is reached. As regards violation of the high-level property (say, sub-objective n), a typical function f_n just indicates whether it is reached or not (e.g., observation of boiler explosion, or not). It is of type "all or nothing", that is, $f_n : S \rightarrow \{0, K\}$ where 0 means that the property is violated and K is a constant cost associated with property satisfaction. Obviously, such a cost function does not help much in guiding the optimization process. Using the concept of dangerous situation, a gradual cost function may be defined based on a "measure" of the severity of the dangerous level reached. The overall cost function $f(s)$ is equal to the minimum value of the different sub-objective cost functions: $f(s) = min\{f_1(s), ..., f_n(s)\}$. Every test sequence s with $f(s) = 0$ meets at least one of the sub-objectives, and thus is retained as a solution to the optimization problem.

3.2 Stepwise Construction of Scenarios

There is no way to know a priori how long it could take to reach property violation. Retaining dangerous situations as solutions, gives the opportunity to progressively explore solution spaces of larger test sequences. It is the rationale for the stepwise construction of scenarios: each step explores possible continuation of the dangerous scenarios found at the previous step, using simulated annealing.

The *first step* searches for test sequences s of predefined maximum length L_1. It involves an instantiation of the simulated annealing algorithm for each class of test sequences previously identified. This first step may allow us to identify several test scenarios: some of them actually violate the target property, while the others only lead to dangerous situations. The latter sequences are analyzed. Some of them may be eliminated based on their very low probability of occurrence, or based on strong evidence that they cannot lead to property violation. The remaining ones require further tests to verify whether the property may be violated, or whether the system is robust enough to recover from the dangerous situation.

From each dangerous scenario s retained at the first step, the *second step* explores possible continuation, under the form of sequences s' of predefined maximum length L_2. Once again, the simulated annealing algorithm is used to guide the search for sequences s' with the aim of violating the target property. During the experiments, each s' is prefixed by a test sequence s that forces the system into the dangerous situation under investigation. The cost function used to evaluate the results of the execution of a complete test sequence $s.s'$ is identical to the one used in the first step.

If test sequences $s.s'$ that lead to dangerous situations (yet, without property violation) are still identified, a *third step* will be performed, using $s.s'$ as the new prefix of subsequent sequences s'' of maximum length L_3. And so on, until every dangerous situation retained at step i, leads - after further tests at step i+1 - to either property violation, or safe situations.

How to formulate the search problem, and how to perform the stepwise construction of scenarios, is exemplified in the next three sections, using first the basic version of simulated annealing algorithm (Section 2.1) as recommended in [2].

4. THE STEAM BOILER CASE STUDY

We have carried out a steam boiler case study [1] for which high-level requirements, control program code, and a software simulator of physical devices, are publicly available [13].

4.1 The Steam Boiler Description

The physical environment comprises the boiler, four pumps to provide the boiler with water, four pump controllers to sense the state of the pumps (open/closed), a device to measure the quantity of water in the boiler, and a device to measure the quantity of steam, which comes out of the boiler. The function of the control program is to maintain the water level in the steam boiler between two predefined thresholds (denoted N_1 and N_2) by controlling pumps. Besides this main function, the control program must also maintain safety by shutting the steam boiler down if its water level is either too low ($<M_1<N_1$) or too high ($>M_2>N_2$) for more than five seconds, otherwise, the steam boiler can be seriously damaged (explosion). Finally, the control program must be able to withstand some physical failures by continuing to operate while part of the equipment is malfunctioning, until it is repaired. The corresponding degraded operational modes are defined in the requirements. If the control program cannot assure its function, it must shut the system down. Our objective is to study the capacity of the control program to maintain safety even in presence of faults affecting some physical devices. So, the target safety property is the "non explosion" of the steam boiler, i.e., "The water level must not be $< M_1$ or $> M_2$ during more than 5 s", where M_1 and M_2 are the safety limits of water level.

A number of formal approaches have been used to model and verify this steam boiler problem [1]. Few of them have considered the whole system, since most have worked on the control program in isolation and proved the safety property (non-explosion of the boiler) on a model of the control program. Our aim is to verify the non-explosion property on the whole system, including the control program and the physical environment. To do this, we use a simulator that mimics the behavior of the physical devices including some physical faults. As shown in Figure 3, our input domain includes the physical faults that can be injected via the simulator, and the cycle numbers during which the physical faults are injected. Recall that our goal is to reach either a boiler explosion or a dangerous situation. Since no safety analysis is available for this case study, we adopt a general notion of dangerousness for control systems: a dangerous situation is reached when the state of the environment perceived by the control program differs from the actual state. Such a situation is identified from the messages exchanged between the boiler simulator and the control program. Explosions are directly observed from the simulator.

In the following sections, we define the parameters of the basic simulated annealing algorithm.

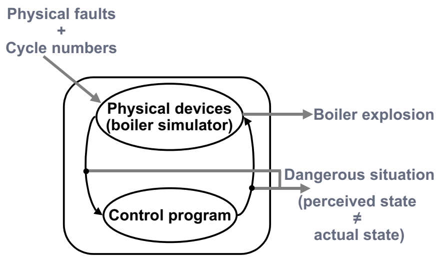

Figure 3. Overview of the system under test

4.2 Test Sequences

A test sequence is defined as a sequence of faults affecting physical devices (e.g., faults affecting pumps or pump controllers) with the cycle numbers during which the faults are to be injected. In this case study, we can inject up to 10 faults affecting the 10 physical devices: 4 pumps (actuators), 4 pump controllers (sensors), the water level sensor and the steam sensor. When a pump fails, its state remains unchanged ("do nothing") until it is repaired. This means that if the pump is open, it remains open until reparation, and if it is closed, it remains closed until reparation. A sensor failure is simulated by a stuck-at-previous-value before failure. The sensor keeps on sending the value of the sensor when the failure arises. Each of the ten possible faults is denoted by a unique identifier, yielding *Fault A .. Fault J.*

According to the principles presented in Section 3.1, we divide the problem into smaller sub-problems. This means that we divide the search space of test sequences into subspaces to be independently explored. For this, we account for, (1) the mode of the boiler which can be *transient* when the boiler heats or *permanent* when it produces steam, and (2) the number of injection cycles which characterizes the temporal dispersion of faults. In the absence of faults, manual calculation indicates that the transition from transient to permanent mode should occur at cycle 3. This result is confirmed by preliminary test experiments coupling the boiler simulator with the control program. The boiler mode thus determines the first potential injection cycle (0 and 3 for transient and permanent modes, respectively). Then, the number of injection cycles can be small, medium or large compared to the boiler dynamics (respectively 3, 5 or 10 cycles). By combining the mode and number of injection cycles, the six identified subspaces are:

Class 1:	Sequences injecting during cycles 0..2 (transient, small);
Class 2:	Sequences injecting during cycles 0..4 (transient, medium);
Class 3:	Sequences injecting during cycles 0..9 (transient, large);
Class 4:	Sequences injecting during cycles 3..5 (permanent, small);
Class 5:	Sequences injecting during cycles 3..7 (permanent, medium);
Class 6:	Sequences injecting during cycles 3..12 (permanent, large).

Whatever the class, we add a fixed number of cycles (8 cycles) after the injection cycles, during which the system evolves. After the system evolution time, the cost of the test sequence is evaluated from the final state reached by the system under test (see Section 4.5). In case the boiler did not explode, the final state is deduced from the observation of all the messages exchanged during the previous cycles.

Figure 4 shows an example of test sequence of Class 2, where faults may be injected during cycles 0..4. In this test sequence, *Fault C* is injected during cycle 0, *Fault H* during cycle 1 and *Faults E* and *A* during cycle 4. After these fault injections, we let the system evolve for 8 cycles. After cycle 12, the cost of the test sequence is evaluated by accounting for the observations that have been collected during the 13 cycles.

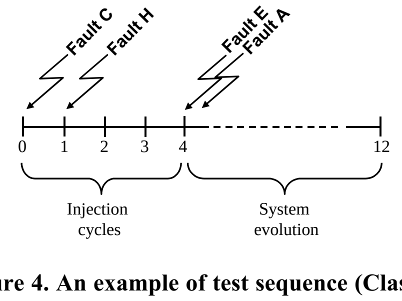

Figure 4. An example of test sequence (Class 2)

4.3 Initialization

Figure 5 shows the algorithm used to generate the initial test sequence (i.e., s_0 in Figure 1) and calculate its cost (using the cost function that will be defined in Section 4.5). It is worth noting that the initial test sequence is generated in accordance with the considered class of sequences: the allowable injection cycles depend on the target class.

```
Choose randomly the total number of faults Nb_Inject_Fault
between 1 and 10
FOR i = 1 to Nb_Inject_Fault
    Choose randomly a fault (without replacement)
    Choose randomly its injection cycle (according to the
    considered class)
ENDFOR
Apply the sequence to simulator + control program
Calculate Current_Cost
```

Figure 5. Initialization of current solution (test sequence) and current cost

4.4 Neighborhood Search

The neighborhood of a test sequence T is defined as the set of test sequences that are obtained from T by:

- either changing the injection cycle of one fault of T (according to the considered class of sequences);
- or adding one fault to T at an allowed injection cycle (according to the considered class of sequences);
- or removing one fault from T.

For instance, a possible neighboring test sequence of the one presented in Figure 4 is the one obtained by changing the injection cycle of *Fault H* from 1 to 0.

The algorithm randomly chooses one of the three alternatives. It then randomly selects one of the possible test sequences. This defines a reasonably small neighborhood structure. Each test sequence is theoretically reachable from every other sequence in the same subspace, possibly at the expense of a large number of iterations.

4.5 Cost Function

A dangerous situation is achieved when, at the end of a test sequence, the state of the boiler simulator perceived by the control program departs from the actual state. This difference may be expressed in three sub-objectives: fewer failure detections, more failure detections, and bad estimation of the water level. The first sub-objective is fulfilled when the control program does not identify the presence of at least one of the injected faults. The second one is fulfilled when the control program wrongly identifies the presence of a fault, which indeed was not injected. The bad estimation of the water level is reached when the water level in the simulator is not included within the plausibility limits (*qc1*, *qc2*)[*] that are calculated by the control program. Figure 6 gives an example of cost function associated with these dangerous situations, where *Nb_Inject_Fault* denotes the number of faults that are injected during the current test sequence, and K_i are positive constants (to be calibrated, see Section 4.6). Let us note that the three sub-objectives are not exclusive: a test sequence can fulfill several sub-objectives at the same time (e.g., both one injected fault not identified and bad estimation of the water level).

[*] *qc1* (respectively *qc2*) denotes the minimal (respectively maximal) water level limit that is calculated by the control program. The observation of *qc1* and *qc2* requires a slight instrumentation of the control program.

Contrary to the gradual cost associated with the three sub-objectives, the cost function associated with the boiler explosion is an "all or nothing" cost function. If the objective is fulfilled, it is equal to zero; otherwise, it is equal to a constant K_4 (see Figure 6).

Our aim is to find test sequences that lead to either an explosion or a dangerous situation. In other words, we aim at finding test sequences that fulfill at least one of the sub-objectives presented in Figure 6. Then, the overall cost of a test sequence equals the minimum value of the four costs associated with the sub-objectives.

Objective	Cost Function
Fewer failure detections	*If* TRUE *then* 0 *else* $\frac{K_1}{Nb\_Inject\_Fault+1}$
More failure detections	*If* TRUE *then* 0 *else* $\frac{K_2}{10-Nb\_Inject\_Fault+1}$
Bad estimation of water level	*If* TRUE *then* 0 *else* $\frac{K3}{qc1-qc2}$
Explosion	*If* TRUE *then* 0 *else* K_4

Figure 6. Proposed cost function

4.6 Calibration

Concerning the cooling schedule (*nrep*, α, see Figure 1), the adopted strategy is to have a small value of *nrep* (equals 1) at many temperatures. As regards α, a classical geometric variation of the temperature, as advised in [2,7], is used: $\alpha(t) = at$, where a is a constant close to 1 (here, $a = 0.95$).

The other constants, like K_i, initial temperature t_0, etc., were calibrated in order to have an acceptance rate between 40% and 60% as reported in [2,7]. After calibration, we obtain the following values: $t_0 = 100$, $K_1 = 250$, $K_2 = 250$, $K_3 = 4000$, $K_4 = 25$, for a maximal number of iterations of 100 (to converge to a zero temperature at the last iterations).

4.7 Basic Simulated Annealing Algorithm for Test Sequence Generation

Figure 7 shows the basic simulated annealing algorithm used in the framework of our experiments. The cost of a test sequence is calculated as shown in Figure 6. The stopping condition is reaching either a zero cost, or a predefined maximal number of iterations (100).

```
Initialization of Current_Test_Sequence and Current_Cost
    (see Figure 5);
Initial Temperature = 100;
Iteration_count = 0;
WHILE (Current_Cost ≠ 0) AND (Iteration_count < 100)
    Select Randomly "New_Test_Sequence" in the
        neighborhood of the Current_Test_Sequence;
    Calculate the "New_Cost";
    δ = New_Cost – Current_Cost
    IF (δ ≤ 0) THEN
        Current_Test_Sequence = New_Test_Sequence;
        Current_Cost = New_Cost;
    ELSE
        IF Random (0,1) < Exp(-δ / Temperature) THEN
            Current_Test_Sequence = New_Test_Sequence;
            Current_Cost = New_Cost;
        ELSE
            -- Do not accept --
Temperature = 0,95 × Temperature;
Increment Iteration_count;
END WHILE
```

Figure 7. Basic simulated annealing for test sequence generation

5. FIRST STEP OF THE TEST STRATEGY

The *first step* searches for test sequences of predefined maximum length (according to the considered class, e.g., 13 cycles for Class 2 and 21 cycles for Class 6).

First experiments are aimed at evaluating both the effectiveness and the efficiency of basic simulated annealing. As regards the effectiveness issue, we study the ability of this approach to find test sequences that lead to either a boiler explosion or a dangerous situation. From previous work [1], we know that such scenarios do exist. It must be verified whether or not the known scenarios, or new ones, are found. As regards the efficiency issue (speed of the search), we compare this approach with another approach that does not use any optimization technique. This latter approach will be called random sampling.

Random sampling uses the same algorithm to generate test sequences as the one initiating the simulated annealing algorithm (see Figure 8). However, it takes into account neither the neighborhood structure nor the cost function in the generation process. The test sequences are generated independently of each other. This generation is performed for each class of sequences. The maximum number of the test sequences generated is 100 (as in the simulated annealing algorithm).

```
Iteration_count = 0;
REPEAT
    Initialization of Current_Test_Sequence and Current_Cost
    (see Figure 5);
    Increment Iteration_count;
UNTIL (Current_Cost = 0) OR (Iteration_count = 100)
```

Figure 8. Random sampling algorithm

For both approaches, 35 experiments were performed for each class of sequences. Each experiment may or may not reach the predefined maximal number of iterations (100).

5.1 Effectiveness

Both the basic simulated annealing and random sampling approaches allowed us to identify three test scenarios leading to steam boiler explosion and two test scenarios leading to a dangerous situation. The identified cases for explosion are not caused by the non-conformance of the control program to its requirements. The problem originates from the very definition of the degraded modes in the requirements (specification fault).

The three scenarios that lead to a boiler explosion are:

1. Faults affecting several pumps or pump controllers followed by the fault affecting the water level sensor, under the condition that the control program detects at least two failures of pump or pump controller before the sensor fails;
2. Fault affecting the steam sensor during cycles 0 or 1;
3. Faults affecting the water level sensor and the steam sensor, during cycles 0 or 1.

Concrete examples of such test scenarios are given in Table 1. For instance, let us consider the first test sequence. It is composed of the fault affecting pump controller 2 ("stuck-at-previous-value", noted *Pump_Ctr2-Fault*) injected during cycle 4 (put in brackets), and the fault affecting pump 3 ("do nothing") injected during cycle 6, and the fault affecting the water level sensor ("stuck-at-previous-value") injected

Table 1. Examples of test sequences leading to boiler explosion or a dangerous situation

TYPE OF SCENARIO	CONCRETE EXAMPLES Fault (injection cycle)	CONTROL PROGRAM BEHAVIOR	
1	Pump_Ctr2-Fault (4), Pump3-Fault (6), Water-Level-Fault (10)	Detection of the pump and pump controller failures, and non-detection of the water level failure	Explosion
2	Steam-Fault (1)	Wrong detection of the water level failure, and non-detection of the steam failure	
3	Water-Level-Fault (0), Steam-Fault (0)	No failure detection	
a	Pump1-Fault (2)	Detection of the pump failure, and wrong detection of its pump controller failure	Dangerous situation
b	Steam-Fault (3)	Non-detection of the steam sensor failure	

during cycle 10. When we apply this test sequence to the simulator + control program, the control program detects the pump controller and pump failures before cycle 10, and consequently enlarges the plausibility interval [*qc1*, *qc2*] of water level for safety reasons. After injection of the fault affecting the water level sensor, the sensor's value remains within [*qc1*, *qc2*]. As a result, the control program will not be able to detect this failure. It continues to trust the sensor, i.e., it believes that the water level is stable until the boiler explodes.

It is worth noting that none of the classes of sequences can find all these scenarios. For example, Classes 4, 5 and 6 do not produce the second and third scenarios since they start fault injection from cycle 3. So, using several classes of sequences allows a better exploration of the input domain. The first scenario was already known in the literature. To the best of our knowledge, the others two were not.

The two scenarios that lead to a dangerous situation are:

a. Fault affecting a pump, while the associated pump controller is working correctly;
b. Fault affecting the steam sensor, just when the mode turns to permanent (during cycle 3).

Table 1 shows two concrete examples. Scenario *a* was already reported in [1] and cannot lead to an explosion. Let us consider the second scenario, which is composed of a fault affecting a steam sensor injected during cycle 3. When we apply this test sequence to the simulator + control program, the control program is not able to detect the steam sensor failure. This is due to the erroneous value being close to the constant value normally delivered during permanent mode. Thus, the steam sensor's value will remain within the interval of steam value calculated by the control program. This simple fault cannot cause a boiler explosion; however, it introduces a small error in the calculation of the expected steam and water level limits. Starting from this dangerous situation, it must be verified whether or not the property is violated when additional faults are injected. Hence, the second scenario requires further testing: this will be the focus of the second step of our strategy (Section 6).

5.2 Efficiency

As regards efficiency, random sampling finds test sequences that fulfill our main objective more quickly than the approach using basic simulated annealing. Table 2 shows examples of results obtained for test sequences of Class 2. In this table, the random seed corresponds to the starting point of the random number generation. It is worth noting that for a given random seed, the first iteration corresponds to the same initial test sequence whatever the generation technique, since the same initialization algorithm is used.

For each approach, Table 2 gives the number of sequences generated until we achieve one of the sub-objectives, and indicates which sub-objective was fulfilled. Whatever the experiment, the predefined maximal number of iterations (100) is never reached: the stopping condition is always a zero cost. Let us consider the first random seed. The approach using basic simulated annealing performs two iterations to reach a dangerous situation (fewer detections), whereas random sampling performs three iterations. But simulated annealing is rarely the quickest technique. For example, for the third random seed, random sampling is much more efficient.

Looking at the average numbers of sequences calculated from a sample of 35 seeds[*], we observe that random sampling is much quicker than the other approach. Indeed, the number of iterations performed by this approach is on average 3.8 while it is 12.7 for basic simulated annealing (Table 2). It is worth noting that the probability of selecting a dangerous test sequence for the steam boiler is high. This explains the small number of iterations that are processed before finding a scenario. Yet, in view of the mean numbers of iterations, we are tempted to say that basic simulated annealing algorithm tends to slow down the search of a solution. Another observation concerns the standard deviation of the iteration number performed by the basic simulated annealing, which is large in value (18.3 as opposed to 2.9 in the case of random sampling)[**]. It means that the efficiency of the approach using the basic simulated annealing is highly dependent on the random seed: if we start in an area that is far away from a solution, because of neighborhood search, we may stay a long time before finding a solution that fulfills the objective.

Given the previous comments on both the mean and standard deviation, the comparaison is in favor of random sampling. Similar results are obtained for the other classes of sequences.

Table 2. First experimental results for test sequences of Class 2

Random Seed	Basic Simulated Annealing		Random Sampling	
	Sequences Number	Final State	Sequences Number	Final State
1st	2	Fewer Detections	3	Fewer Detections
2nd	3	Explosion	2	Explosion
3rd	17	Fewer Detections	4	Fewer Detections
4th	1	Fewer Detections	1	Fewer Detections
5th	3	Explosion	5	Explosion
6th	9	Explosion	4	Fewer Detections
7th	9	Explosion	2	Explosion
8th	1	Fewer Detections	1	Fewer Detections
9th	12	Explosion	6	Explosion
10th	14	Explosion	5	Explosion
…	…	…	…	…
Mean (35 seeds)	12.7		3.8	
Standard Deviation	18.3		2.9	

[*] The sample of 35 seeds ensures that the actual mean for random sampling lies in the interval 3.8±1 with a 95% confidence level.

[**] As a result, the actual mean for basic simulated annealing cannot be accurately evaluated from the sample of 35 seeds. However, the probability of this mean being lower than the random sampling one is negligible.

5.3 Improvements of the Basic Simulated Annealing

The poor performance of basic simulated annealing leads us to investigate modifications in order to improve its efficiency. Indeed, as said by Dowsland [2], "Many of the reported successes involve modifications to the basic annealing algorithm". We assume that the optimization problem has been adequately formulated, i.e., the search has to be improved keeping the same search objective and associated cost function.

The first experiments presented above provide a significant insight into the main weakness of the basic algorithm applied to our search problem: its efficiency remains overly dependent on the initial solution. This is due to the fact that when no cost improvement is observed, the search does not allow us to perform moves larger than those authorized by the neighborhood function. Hence, a beneficial improvement could be obtained by searching in the neighborhood of the current solution while allowing, in case of no cost improvement, to search elsewhere.

Several modifications have already been confirmed useful in adapting the simulated algorithm to a number of different problems [2]. Some of them were aimed at increasing the probability of uphill moves. Here is a non exhaustive list of proposed adaptations:

- As regards the acceptance rate of inferior solutions, the standard exponential distribution, which is a function of the magnitude of the cost increase δ, gives virtually no chance of acceptance of large increases, while small increases may be accepted regularly. Hence, some authors suggest the use of a linear distribution, or even of probabilities independent of δ.
- As regards the cooling schedule, some authors observed that most of the useful work is done in the middle of the schedule. Thus, it may be advantageous to search only in the middle part of the temperature range. To do this, an extreme solution is to use a constant temperature instead of gradually reducing it during the search (function α).
- If we accept that there is no reason to cool the temperature, we may authorize the use of heating. The idea is to move downhill, but if no progress is apparent in searching the current valley a concerted uphill effort would be made in order to widen the scope of the search.
- Another type of adaptation consists in adjusting the neighborhood structure. For example, the search can start with a large neighborhood at high temperatures, and this neighborhood is restricted as the temperature drops. Or, independently of the temperature, it may be decided to enlarge the neighborhood structure after a given number of trials with no cost improvement observed.
- Finally, the use of several short runs (obtained by restarting the algorithm) rather than a single run (with a slow cooling schedule), has also been suggested.

Having analyzed the previous adaptations, and in order to allow significant moves in case of no cost improvement, the revised version of the simulated annealing we propose is based on the following principles. When the new solution is inferior to the current one:

(i) The acceptance probability is independent of δ and a constant temperature is used. The simplest way to achieve this is to set the acceptance probability to a predefined constant value.

(ii) Allowing the use of heating is expected not to be sufficient since it increases the probability of accepting inferior solutions, but these solutions remain in the neighborhood structure. A more effective process should be the use of several short runs, by allowing the random choice of a new initial solution (restart of the algorithm). The principle we retain is to set a probability of restarting the algorithm in case of no cost improvement. This principle is much simpler to implement than the one consisting in varying the neighborhood structure.

Figure 9 shows the revised algorithm (modifications are indicated in bold characters). In the *WHILE* loop, better solutions are always accepted. When an inferior solution is selected, it is either accepted with a probability of 30%, or rejected with a probability of 30%, or the algorithm restarts with a probability of 40%. These values mean that 57% of the rejected moves lead to a reset, which gives a significant chance to escape from the small neighborhood space of the current solution. Obviously, the probabilities have been empirically calibrated. The efficiency of the algorithm is assessed in the next section.

```
Iteration_count = 0;
REPEAT
  Initialization of Current_Test_Sequence and Current_Cost
     (see Figure 5);
  Reset-Flag = FALSE;
  WHILE (Current_Cost $\neq$ 0) AND (Reset_Flag = FALSE)
          AND (Iteration_count < 100)
     Select Randomly "New_Test_Sequence" in the
          neighborhood of the Current_Test_Sequence;
     Calculate the "New_Cost";
     Increment Iteration_count;
     $\delta$ = New_Cost – Current_Cost
     IF ($\delta \leq 0$) THEN
        Current_Test_Sequence = New_Test_Sequence;
        Current_Cost = New_Cost;
     ELSE
        30 %: -- Accept New_Test_Sequence -- OR
        30 %: -- Do not accept -- OR
        40 %: Reset_Flag = TRUE;
  END WHILE;
UNTIL (Current_Cost = 0) OR (Iteration_count = 100)
```

Figure 9. Revised version of the simulated annealing algorithm

6. SECOND STEP OF THE TEST STRATEGY

Because of the high probability of selecting a test sequence that leads to an explosion or a dangerous situation, we did not apply the revised algorithm to the first step of our test strategy. Indeed, random sampling alone is very likely to be more efficient. Hence, we only applied it to the second step.

For each dangerous scenario identified in the first step, the *second step* performs further tests. It explores possible continuation of the scenarios with the aim of violating the safety property. In these experiments, only dangerous scenarios of type *b* are considered (see Table 1), since according to previous work [1], those of type *a* cannot lead to boiler explosion. For the purpose of comparison, three techniques are implemented: basic simulated annealing, the revised version, and random sampling.

Whatever the technique, the second step of the test strategy requires adapting the generation of test sequences. We have to account for the fact that faults must be injected during the cycles that follow the dangerous situation. Concretely, since the dangerous situation is the fault affecting the steam sensor at cycle 3, the continuation of the scenario may inject faults only from cycle 4. We keep the six classes of sequences of Section 4.2, but the beginning of the injection cycles is delayed by four cycles, i.e., after the dangerous situation. This means that the classes that started fault injection at cycle 0 now start injection at cycle 4, and those that started injection at cycle 3 now start at cycle 7. It is worth noting that the classification of test sequences is no longer related to the boiler mode being transient or permanent. Rather, it characterizes the time spent in the dangerous situation before additional faults may occur.

First experimentation revealed a problem requiring another adaptation of test sequence generation. Boiler explosion was repeatedly observed, but this was due to the reproduction of Scenario 1 already identified at the first step of the test strategy. Indeed, we were able to confirm that the dangerous situation played no role in the observed explosions: the same results were observed when changing the prefix of the corresponding test sequences (i.e., removing the steam sensor fault at cycle 3). In order to overcome this problem, we implemented a functionality to automatically eliminate the cases for boiler explosion that are unrelated to the dangerous situation. The algorithms of both versions of simulated annealing, and of random sampling, are modified in this way:

1. Each time a new test sequence is generated, it is applied to the system without "stress", i.e., without putting the system in the dangerous situation; then
2. If this test sequence leads to an explosion, the algorithms generate another test sequence. Otherwise, the test sequence is retained as a candidate solution, to be applied to the system put in the dangerous situation.

Note that, in the general case, this functionality is not expected to be required since safety property violation should be seldom observed.

Table 3 shows the results obtained for Class 2. In this table, the number of sequences corresponds to retained test sequences, i.e., the ones that do not lead to boiler explosion when applied without "stress". Given a random seed, test sequence generation proceeds until we achieve either the objective or the maximal number of iterations (100 retained test sequences). Unsuccessful search is observed several times: property violation due to the continuation of scenario *b* is not trivially triggered.

Each of the three techniques, basic simulated annealing, the revised version or random sampling, allows us to identify a new explosion scenario related to the dangerous situation. A concrete example of the scenario is:

Steam_Fault (3), Pump2-Fault (4), Water-Level-Fault (8)

The scenario involves one pump or pump controller failure followed by water level failure. It corresponds to the triggering of an extremal/special case. When the above test sequence is applied without "stress" (no *Steam-Fault* at cycle 3), the boiler safely shuts down. This is so because the value delivered by the faulty sensor turns out to fall outside the plausibility interval [*qc1, qc2*] calculated by the control program.

Table 3. Comparison of three techniques for the second class of sequences

Random Seed	Basic Simulated Annealing		Revised Version of Simulated Annealing		Random Sampling	
	Sequences Number	Final state	Sequences Number	Final state	Sequences Number	Final state
1st	100	---	11	Explosion	68	Explosion
2nd	62	Explosion	28	Explosion	100	---
3rd	100	---	87	Explosion	100	---
4th	100	---	100	---	65	Explosion
5th	100	---	83	Explosion	82	Explosion
6th	100	---	100	---	100	---
7th	76	Explosion	100	---	55	Explosion
8th	100	---	87	Explosion	70	Explosion
9th	100	---	100	---	93	Explosion
10th	39	Explosion	31	Explosion	100	---
Total Number (35 seeds)	3146		2296		2668	
Successful Search	6		20		17	

As already mentioned, the injection of *Steam-Fault* at cycle 3 introduces a small error in the calculation of the water level limits, yielding a slight overestimation $qc2+ \varepsilon$ of the upper plausibility bound. Now, the scenario is such that when the fault affecting the water level sensor occurs, the erroneous sensor value is greater than *qc2*, but still lower than $qc2+ \varepsilon$. The water level sensor failure is not detected, and the system inexorably evolves toward boiler explosion.

As to the comparison of the three techniques, we have seen that all were effective. As regards efficiency, we observe a significant improvement of the revised version over the basic version of simulated annealing. The former technique finds the new scenario for 20 different seeds, whereas the latter finds it only 6 times. By comparing the revised technique to random sampling, we observe a slight improvement both in terms of total number of iterations and successful search.

It is worth noting that no new dangerous scenario is identified by these experiments. Hence, the test strategy applied to the steam boiler case study involves only two steps.

7. CONCLUSION AND FUTURE WORK

The test strategy we propose aims at validating cyclic real-time control systems with respect to high-level safety properties. Given a control program and its controlled environment (or a simulator of the physical devices), it consists in exercising the program in closed loop with its (simulated) environment, and in observing whether a target property is violated or not. The strategy is based on the stepwise construction of test scenarios. Each step explores possible continuation of the scenarios retained at the previous step, using heuristic search techniques. As a first investigation, the strategy is instantiated with simulated annealing.

The results of the steam boiler case study tends to confirm the feasibility and usefulness of the stepwise construction of test scenarios. This principle allowed a progressive exploration of longer trajectories in the input domain until property violation, as exemplified by the last uncovered scenario.

As regards the use of simulated annealing, the performance of the basic algorithm was really disappointing compared to blind random sampling. The reason for its poor efficiency seems to be its overly dependence on the initial solution. We propose a variant of the basic algorithm, in which 57% of the rejected moves lead to a reset of the algorithm. This gives a significant chance to escape from the neighborhood space of the current solution, when no cost improvement is observed. As indicated by the general literature on heuristic techniques, some effort is always required to tune the search process to the problem to be solved. In our case, the effort was fruitful since the revised algorithm exhibited much better performance. Yet, compared to random sampling, the cost-effectiveness of the revised algorithm remains questionable. Other case studies will be conducted to assess the power of our algorithm and investigate further improvement. In particular, it will be interesting to experiment with examples for which random sampling becomes quite ineffective. Our future work will also consider instantiations of the test strategy with another candidate technique, genetic algorithms.

As a general comment, we experienced that the effort required to apply heuristic search techniques is far from negligible. Using simulated annealing, many trials were necessary for investigating alternative design choices, and calibrating the corresponding parameters. This is an intrinsic limitation of any one of the modern heuristic search techniques. But, when the complexity of the problem is such that more traditional approaches are not sufficient, heuristic techniques may offer an automated solution to guide the exploration of large input spaces. What we need is feedback from empirical studies, reporting both success and difficulties, in order to gain further insights into the use of the heuristics for typical instances of testing problems. It is hoped that the detailed experimental material provided in this paper contributes to this goal.

8. ACKNOWLEDGEMENTS

This work is partially supported by the European Community (Project IST-1999-11585: DSoS – Dependable Systems of Systems).

9. REFERENCES

[1] J-R. Abrial, E. Börger and H. Langmaack, “Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control”, Springer-Verlag, 1996.

[2] K.A. Dowsland, “Simulated Annealing”, in Reeves, “Modern Heuristic Techniques for Combinatorial Problems”, chapter 2, pp. 20-69, McGraw-Hill, 1995.

[3] H.G. Gross, B. Jones and D. Eyes, “Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems”, *in IEE Proceedings software*, 147(2), pp. 25-30, April 2000.

[4] B. Jones, H. Sthomer and D. Eyes, “Automatic Structural Testing Using Genetic Algorithms”, *Software Engineering Journal*, 11(5), pp. 299-306, 1996.

[5] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, “Optimization by Simulated Annealing”, *Science*, 220(4598), pp. 671-680, 1983.

[6] C.C. Michael, G. McGraw and M.A. Schatz, “Generating Software Test Data by Evolution”, *IEEE Transactions on Software Engineering*, 27(12), pp. 1085-110, 2001.

[7] R.P. Pargas, M.J. Harrold and R.R. Peck, “Test-Data Generation Using Genetic Algorithms”, *Journal of Software Testing, Verification and Reliability*, 9(4), pp. 263-282, 1999.

[8] V.J. Rayward-Smith, I.H. Osman, C.R. Reeves and G.D. Smith, “Modern Heuristic Search Methods”, Wiley, 1996.

[9] A.C. Shultz, J.J. Grefenstette and K.A. De Jong, “Learning to Break Things: Adaptive Testing of Intelligent Controllers”, Naval Research Laboratory, Oxford University Press, 1995.

[10] N. Tracey, J. Clark and K. Mander, “Automated Program Flaw Finding using Simulated Annealing”, in Proc. Int. Symp. on Software Testing and Analysis (ISSTA’98), Clearwater Beach, Florida, USA, ACM Press, pp. 73-81, 1998.

[11] N. Tracey, J. Clark, K. Mander and J McDermid, “Automated test-data generation for exception conditions”, *Software–Practice and Experience*, 30(1), pp 61-79, 2000.

[12] N. Tracey, “A Search-Based Automated Test-Data generation Framework for Safety-Critical Software”, Doctoral Dissertation, University of York, 2000.

[13] http://www.atelierb.societe.com/BOILER/UK/main.h