An Empirical Investigation of Simulated Annealing Applied to Property-Oriented Testing

Olfa Abdellatif-Kaddour, Pascale Thévenod-Fosse, and Hélène Waeselynck
LAAS-CNRS, 7 Avenue Colonel Roche,
31077 Toulouse CEDEX 4 – France
Tel: +33 / 5 61 33 62 97
Fax: +33 / 5 61 33 64 11
{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. The paper defines a test strategy for safety properties in cyclic control systems. It consists of the stepwise construction of test scenarios. Each step explores possible continuations of the dangerous scenarios found at the previous step, using black-box sampling techniques. Here, emphasis is put on using a heuristic search technique – namely, simulated annealing – as an automatic way to sample over the input space. Empirical investigation is conducted on a steam boiler case study. The target property is the "non explosion" of the boiler in presence of faults in the physical devices. The experimental results lead us to propose a revised version of the basic simulated annealing algorithm, whose efficiency is promising.

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

1. INTRODUCTION
The automated test-data generation remains an open issue for many software testing problems. This paper focuses on a particular class of problems, i.e., property-oriented testing of cyclic real-time control systems. Property-oriented testing uses the specification of a property to drive the testing process [6]. 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., the control program is run on the target hardware and connected to a simulator of the physical environment, with facilities for fault injection). However, there is the issue of test data selection. To address this issue, we have proposed a strategy consisting of the stepwise construction of test scenarios [1]: each step explores possible continuations of the "dangerous" scenarios identified at the previous step, using black-box sampling techniques. In this paper, emphasis is put on using a heuristic search technique called *simulated annealing* [12], as an automatic way to sample over the input space. Empirical investigation is conducted on a steam boiler case study [2], the target property being the "non explosion" of the boiler in presence of faults in the physical devices.
Section 2 recalls the proposed strategy. Section 3 mainly concentrates on simulated annealing. The steam boiler case study is described in Section 4. First experiments reported in Section 5 lead us to observe a poor efficiency of the basic simulated algorithm applied to the case study. Then, a revised version of the algorithm is defined in Section 6, which exhibits a promising efficiency. Conclusions and future directions are given in Section 7.

2. OVERVIEW OF THE TEST STRATEGY
The ultimate objective is to find reachable system states where the target property is violated (if any). Due to the sequential (with memory) behavior exhibited by control systems, it is expected that property violation will occur after execution of particular trajectories in the input domain of the system, rather than just at specific points of the input domain: the system will progressively evolve towards property violation. During this progressive evolution, the system reaches intermediate states that correspond to *dangerous situations*, that is, states from which property violation may eventually occur if the system is not robust enough to recover. For example, a general notion of dangerousness for control systems may be related to the fact that the state of the controlled environment, as perceived by the control program, differs from the actual state. In practice, the dangerous situations specific to a given system domain can be elaborated based on the safety analyses that accompany system development. Their pertinence is a prerequisite to the effectiveness of the test strategy we propose. Hence, the strategy should be conducted in close connection with domain experts.
There is no way to know *a priori* how long it could take to reach property violation. Using the dangerous situations as intermediate test objective gives us the opportu-

nity to progressively explore solution spaces of larger test sequences. This is the rationale for the stepwise construction of scenarios we propose: each step explores possible continuations of the dangerous situations reached at the previous step, trying to get close to a violation.

The stepwise construction of test scenarios proceeds as follows. The *first step* searches for test sequences s of predefined maximum length L_1. The test objective includes both property violation during the execution of s, and the reaching of a dangerous situation at the end of the execution of s (without property violation). Then, all sequences found are analyzed to determine whether or not some of them require further tests. From each dangerous scenario s retained at the first step, the *second step* explores possible continuations, in the form of sequences s' of predefined maximum length L_2. During the experiments, each s' is prefixed by a sequence s that forces the system into the dangerous situation under investigation, yielding a complete sequence $s.s'$. If a test sequence $s.s'$ is retained, the *third step* is 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 — either to a property violation, or to a safe situation.

At step i, the process of test sequence selection should try to favor the choice of those scenarios that will be the most "stressing" with respect to the test objective. The definition of such a guiding process requires consideration of two main issues: (1) the large input space to be sampled, even for a bounded length L_i, and (2) the intractability of a detailed analysis of the system behavior. The first problem may be alleviated (to a certain extent) by decomposing the input space into smaller subspaces to be separately explored during testing. The second one compels us to use a black-box approach to select the test sequences.

As a first investigation, we used random sampling as a simple example of black-box technique. The feasibility of the strategy instantiated with this technique was illustrated on a steam boiler case study [1]. Experimental results are promising since four scenarios that violate the property were found. However, the effectiveness of random sampling remains questionable when very few scenarios from the subspace may reach the test objective, which hopefully should be the case for real safety-critical systems. In this paper, we investigate the power of heuristic search techniques as potential solutions to the problem of selecting "stressing" scenarios.

3. RELATED WORK

3.1. Heuristic Search Technique for Testing Problems

Modern heuristic search techniques [12] 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. 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 [9, 10, 11], conformance testing [14], exception testing [15], robustness testing [13], and determination of Worst-Case Execution Time [7].

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.

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 [7], 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., [5, 12]), 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, we focus on one search technique, namely simulated annealing.

3.2. Basic Simulated Annealing

This informal outline of Simulated Annealing (SA) is based on the synthesis presented in [5].

Suppose that we have a minimization problem over a set of feasible solutions S and a cost function $f : S \rightarrow \Re$ that can be calculated for all $s \in S$. SA 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 solutions employed. As a result, these strategies often yield convergence to a local, not a global, minimum. This is the motivation for SA, which offers a way of alleviating the problem by allowing some uphill moves in a controlled manner.

The SA 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 SA 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 of 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 s0;
Select an initial temperature t0 > 0;
Select a temperature reduction function α;
REPEAT
  REPEAT
      Randomly select s ∈ N(s0);
      δ = f(s) – f(s0);
      IF δ ≤ 0
      THEN s0 = s
      ELSE
          Generate random x uniformly in range (0, 1);
          IF x < exp(-δ/t) THEN s0 = s;
  UNTIL iteration_count = nrep;
  Set t = α(t);
UNTIL stopping condition = true.
```

Figure 1. SA 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 algorithm for the solution of a particular problem. They can be divided into two categories:

- *Generic decisions*, which involve three parameters: 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 SA, some 'qualitative' guidelines are argued in [5].

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. 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 number of iterations at many temperatures. The two most widely used cooling rate functions are a geometric function $\alpha(t) = at$ (where $0.8 < a < 0.99$), and a function $\alpha(t) = t/(1 + \beta t)$ where β is a small value.

As regards the *problem-specific decisions*, 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. Small simple neighborhoods are generally 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 number of experiments, then – based on the analysis of the results of these first experiments – improvements of the basic algorithm may be proposed to adapt it to the type of problem under study.

4. THE STEAM BOILER CASE STUDY

We have carried out a case study based on a steam boiler problem for which high-level requirements, control program code, and a software simulator of physical devices, are publicly available [4]. A number of formal approaches have been used to model and verify this steam boiler problem [2]. Few of them have considered the whole problem, since most have focused on the control program. They built formal models based on its informal specification, paying little attention to the (implicit) assumptions this specification puts on the controlled environment. In contrast to previous work, our aim is to verify the absence of critical failure modes when the control program is connected to the physical environment. Most of the complexity of the control program specification comes from the definition of fault tolerance mechanisms. Hence, we put emphasis on a specific category of environmental inputs: the faults to be tolerated. The stepwise strategy is applied to the search for scenarios combining faults with the functional activity, and yielding a violation of safety requirements.

4.1 Steam Boiler Description

The physical environment comprises the boiler, 4 pumps to provide the boiler with water, 4 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 aim is to study the capacity of the control program to maintain safety in the presence of faults affecting the 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 $> M_2$ for more than 5 s", where M_1 and M_2 are the water level safety limits. This is a high-level property depending on proper interaction between the control program and the physical environment. To test this property, we use a simulator that mimics the behavior of the physical devices, including some physical faults.

4.2. Test Input Domain and Dangerous Situations

As shown in Figure 2, 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. Hence, 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 fault occurs, its state remains unchanged ("do nothing") until it is repaired: if the pump is open (respectively closed), it remains open (respectively closed) until it is repaired. When a sensor fault occurs, the sensor keeps sending the value sensed immediately prior to the fault occurring.

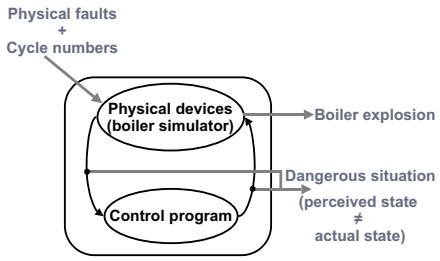

Figure 2. Overview of the system under test

According to Section 2, the test objective 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, as perceived by the control program, differs from the actual state. This difference may be expressed in three sub-objectives:

- *Fewer failure detections*, which is fulfilled when the control program does not identify the presence of at least one of the injected faults.
- *More failure detections*, which is fulfilled when the control program wrongly identifies the presence of at least one fault, which indeed was not injected.
- *Bad estimation of the water level*, which is reached when the water level in the simulator is not included within the plausibility limits (qc_1, qc_2)* that are calculated by the control program.

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 one not-injected fault wrongly identified).
A dangerous situation is identified from the messages exchanged between the simulator and the control program. An explosion is directly observed from the simulator (see Figure 2).

4.3. Previous Experimental Results

The test strategy has already been applied to the steam boiler, using random sampling to select test sequences [1]. The random sampling algorithm was the following: choose uniformly the number of injected faults; then, choose uniformly each fault (without replacement) and its injection cycle.
This first instantiation of the strategy involved two steps. According to Section 2, for each step, we decomposed the large input space into subspaces separately explored during testing. For example, at the first step of the strategy, six subspaces were defined, taking into account: (1) the operating mode of the boiler, and (2) the number of injection cycles, which characterizes the temporal dispersion of faults.
Experimental results allowed us to identify four test scenarios leading to steam boiler explosion. Three of them were found at the first step and have a high probability to be selected by random sampling. The fourth one required two steps. At the first step, a dangerous scenario was identified, in which the control program is not able to detect a steam sensor failure at cycle 3 (transition from initialization to permanent mode). This introduces a small error in the calculation of the water level interval. The continuation of this dangerous scenario at the second step led us to identify the fourth explosive scenario.

5. EXPERIMENTATION WITH BASIC SA

Since random sampling was quite efficient during the first step of the strategy, more sophisticated search techniques are not needed in that particular case. As a result, the empirical framework we used to explore SA on the steam boiler case study is restricted to the second step of the

---

* qc_1 (respectively qc_2) denotes the minimal (respectively maximal) water level limit that is calculated by the control program. The observation of qc_1 and qc_2 requires a slight instrumentation of the control program.

strategy. Our aim is to assess its efficiency in producing the fourth explosive scenario – called thereafter the explosive scenario, for short. As regards the search space, we retain only part of the input domain that was considered at the second step of the strategy: for the reason just explained, we focus on a subspace where the random sampling technique turned out not to be very efficient.

5.1. Search Space

The search space involves test sequences prefixed by the injection of *Steam-Fault* at cycle 3, so as to trigger the dangerous situation identified at the first step of the strategy. The continuation after the dangerous situation calls for injection of additional faults during cycles 4..8. After these allowable injection cycles, we let the system evolve for 8 cycles. In the example sequence (Figure 3), after injection of *Steam-Fault* at cycle 3 (dangerous situation), *Pump2-Fault* is injected during cycle 5 and *Water-Level-Fault* during cycle 8. After the system evolution time (cycles 9 to 16), the test outcome is determined by accounting for the observations that have been collected during the 17 cycles.
In [1], the random sampling algorithm over this search space worked as follows: choose uniformly the number of faults injected in addition to *Steam-Fault*; then, choose uniformly each fault (without replacement) and its injection cycle in [4..8].

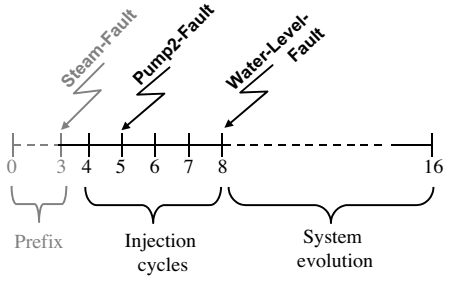

Figure 3. Example test sequence

We now describe our implementation of basic SA as an alternative sampling technique.

5.2. Cost Function

Figure 4 gives the example cost function designed for guiding the SA search, where K_i are positive constants (to be calibrated, see Section 5.4). The test objective is to reach either a boiler explosion or a dangerous situation. Hence, a zero cost is assigned to test sequences that fulfill this test objective.
The 'else' part of the function defines the gradual cost to be calculated when the test objective is not fulfilled. M_1 and M_2 denote the safety limits of the water level in the boiler, $qc_1 < qc_2$ are the plausibility limits of water level that are calculated by the control program, and q_r is the "real" water level in the boiler. This cost function rewards test sequences that increase the incertitude of the water level (qc_2-qc_1) or make the real water level get close to the safety limits (M_1 and M_2).

IF [(*Explosion*) OR (*Fewer failure detections*) OR (*More failure detections*) OR (*Bad estimation of water level*)]

THEN

Cost = 0

ELSE

$$\text{Cost} = \frac{K_1}{qc_2 - qc_1} + K_2 \,.\, Min\,(\,|\,q_r - M_1\,|\,,\,|\,q_r - M_2\,|\,)$$

Figure 4. Proposed cost function

5.3. 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;
- or adding one fault to T at an allowed injection cycle;
- or removing one fault from T.

For instance, starting from the test sequence in Figure 3, we may change the injection cycle of *Pump2-Fault* from 5 to 4, and obtain one instance of the explosive scenario.
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. Each test sequence is theoretically reachable from every other sequence in the search space, possibly at the expense of a large number of iterations.

5.4. Calibration

Concerning the cooling schedule (*nrep*, α, see Figure 1), the adopted strategy is to have a small value of *nrep* (equal to 1) at many temperatures. As regards α, we use a classical geometric variation of the temperature, as advised in [5, 12]: $\alpha(t) = at$, 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 initial acceptance rate between 40% and 60%, as recommended in [5, 12]. After calibration, we obtain the following values: $t_0 = 100$, $K_1 = 10000$, $K_2 = 0.2$, for a maximal number of iterations of 100 (to converge to a zero temperature at the last iterations).

5.5. Overview of the Algorithm

Figure 5 shows the basic SA algorithm used in the framework of our experiments. The initialization of the search is performed by calling once the random sampling algorithm: this gives the first "*Current_Test_Sequence*", whose cost is calculated after execution on the boiler system. The stopping condition is true when either the test objective is reached (zero cost) or 100 test sequences have been unsuccessful. A complete execution of the algorithm will be called a run.

```
Initialize Current_Test_Sequence and calculate Current_Cost;
Initial Temperature = 100;
Iteration_count = 0;
WHILE (Current_Cost ≠ 0) AND (Iteration_count < 100)
    Select Randomly "New_Test_Sequence" in the neighbor-
        hood 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 5. Basic SA for test sequence generation

5.6. Experimental Results

Experiments with basic SA involved 35 runs. Some of them allowed us to produce the explosive scenario. For the purpose of comparison, we also performed 35 runs with random sampling, taking the same random seed as for SA. That is, a random sampling run starts from the same test sequence as the SA run. The performance of the basic SA was disappointing compared to random sampling: only 7 successful runs versus 14 in the case of random sampling (see Table1). We are tempted to say that the basic SA algorithm slows down the search for a solution. Having a closer look at the runs, we noticed that the efficiency of the approach using basic SA is highly dependent on the random seed: if we start in an area that is far away from a solution, according to neighborhood search, we may take a long time to find a solution that fulfills the objective. The poor performance of basic SA has led us to investigate modifications in order to improve its efficiency.

6. ADAPTATION OF SA

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.

6.1. Variants of the Basic SA

The main weakness of the basic algorithm applied to our search problem is its overly dependency 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, an improvement could be obtained by searching in the neighborhood of the current solution while allowing search elsewhere when there is no cost improvement.

Several modifications have already been confirmed useful for adapting the SA algorithm to a number of different problems [5]. 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 search scope.
- Another type of adaptation consists of 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 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, we now propose a revised version of SA.

6.2. Revised Version and Experimental Results

The revised version of SA 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 not expected to be sufficient. It increases the probability of accepting inferior solutions, but these solutions remain in the neighborhood. 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 the case of no cost improvement. This principle is much simpler to implement than the one consisting of varying the neighborhood structure.

```
Iteration_count = 0;
REPEAT
  Initialize of Current_Test_Sequence and Current_Cost;
  Reset-Flag = FALSE;
  WHILE (Current_Cost _ 0) AND (Reset_Flag = FALSE)
        AND (Iteration_count < 100)
     Select Randomly "New_Test_Sequence" in the neigh-
        borhood of the Current_Test_Sequence;
     Calculate the "New_Cost";
     Increment Iteration_count;
     δ = New_Cost – Current_Cost
     IF (δ ≤ 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 6. Revised version of the SA algorithm

Figure 6 shows the revised algorithm (modifications are indicated in bold characters). In the *WHILE* loop, better solutions are always accepted (as before). When an inferior solution is found, 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 of escape from the small neighborhood of the current solution. These probabilities have been empirically calibrated.

New experiments have been conducted to assess the efficiency of the revised algorithm (see Table 1). We observe a significant improvement of the revised version over the basic version of SA. The former technique finds the explosive scenario for 19 different seeds, whereas the latter finds it only 7 times. By comparing the revised technique to random sampling, we observe a moderate improvement both in terms of total number of iterations and successful search.

These results are encouraging and justify the interest of further work, both theoretical and experimental, on heuristic search techniques applied to property-oriented testing.

7. FUTURE WORK

In this paper, we experimented with the use of a heuristic search technique, simulated annealing. The investigation is performed in the framework of a strategy for property-oriented testing of cyclic control systems: the heuristic technique is intended to offer an automated solution to guide the exploration of large input spaces at the various steps of the strategy.

The performance of the basic SA algorithm was disappointing compared to random sampling. The reason for its poor efficiency seems to be its overly dependence on the initial solution. We have proposed a variant of the basic algorithm, in which 57% of the rejected moves lead

Table 1. Comparison of three techniques

Random seed of the run	Random Sampling		Basic Simulated Annealing		Revised Version of Simulated Annealing	
	Sequences generated	Final state	Sequences generated	Final state	Sequences generated	Final state
1st	88	Explosion	100	---	100	---
2nd	95	Explosion	100	---	100	---
3rd	100	---	100	---	42	Explosion
4th	45	Explosion	92	Explosion	71	Explosion
5th	100	---	100	---	100	---
6th	100	---	100	---	100	---
7th	100	---	100	---	75	Explosion
8th	100	---	100	---	15	Explosion
9th	100	---	100	---	21	Explosion
10th	100	---	100	---	100	---
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Total number of sequences generated (35 runs)	2789		3253		2423	
Successful Search	14		7		19	

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 than the basic algorithm. Yet, compared to random sampling, the improvement is not so impressive.

To further improve efficiency, we are now considering variations in the search problem formulation. Let us recall that the cost function was let unchanged when investigating variants of SA. We will work on tuning this function, and experiment with alternative penalty/reward policies for test sequences that do not reach the test objective (i.e., whose cost is not zero). Intuitively, it can be understood that a key factor to efficiency is the correlation between the cost function and neighborhood structure. If the costs of neighbors are unrelated, then there is no point in exploring the neighborhood of the best solutions found so far.

Theoretical work on heuristic search techniques has settled a mathematical framework to study this issue. The cost function and neighborhood structure form what is called a *landscape*. The metaphor gives rise to the visual image of a more or less mountainous landscape with valleys, ridges and peaks (consider the cost as defining the "height"). A smooth landscape is one in which neighbors have nearly the same cost, while a rugged one is one in which the cost values are uncorrelated. Several measures of ruggedness have been proposed and used to make quantitative predictions about search on landscapes. Previous work considered either classical optimization problems (e.g. the Quadratic Assignment Problem) [3], or artificial landscapes that can be tuned from smooth to rugged [8]. We will investigate the predictive power of such measures for guiding the formulation of a testing problem in terms of a search problem.

8. ACKNOWLEDGEMENTS

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

9. REFERENCES

[1] O. Abdellatif-Kaddour, P. Thévenod-Fosse and H. Waeselynck, "Property-Oriented Testing: A Strategy for Exploring Dangerous Scenarios", *ACM Symposium on Applied Computing (SAC 2003)*, Melbourne, Florida, USA, (2003).

[2] J-R. Abrial, E. Börger and H. Langmaack (Eds), *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, Springer-Verlag, 1996.

[3] E. Angel and V. Zissimopoulos, "On the classification of NP-complete problems in terms of their correlation coefficient**",** *Discrete Applied Mathematics*, vol. 99, pp. 261-277, (2000).

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

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

[6] G. Fink and M. Bishop, "Property-Based Testing; A New Approach to Testing for Assurance", *ACM SIGSOFT on Software Engineering*, 22(4), pp. 74-80, (1997).

[7] 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, (2000).

[8] W. Hordijk, "A measure of landscapes", *Evolutionary Computation*, vol. 4, no. 4, pp. 335-360, (1996).

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

[10] 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).

[11] 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).

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

[13] A.C. Schultz, J.J. Grefenstette and K.A. De Jong, "Learning to Break Things: Adaptive Testing of Intelligent Controllers", in *Handbook on Evolutionary Computation*, Chapter G3.5, IOP Publishing Ltd. and Oxford Press, 1997.

[14] 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).

[15] 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).