Simulated Annealing Applied to Test Generation: Landscape Characterization and Stopping Criteria

Hélène WAESELYNCK, Pascale THEVENOD-FOSSE, Olfa ABDELLATIF-KADDOUR
LAAS-CNRS
7, Av du Colonel Roche
31077 Toulouse Cedex 4 – France
{Helene.Waeselynck, Pascale.Thevenod}@laas.fr

Abstract – This paper investigates a measurement approach to support the implementation of Simulated Annealing (SA) applied to test generation. SA, like other metaheuristics, is a generic technique that must be tuned to the testing problem under consideration. Finding an adequate setting of SA parameters, that will offer good performance for the target problem, is known to be difficult. Our measurement approach is intended to guide the implementation choices to be made. It builds upon advanced research on how to characterize search problems and the dynamics of metaheuristic techniques applied to them. Central to this research is the concept of landscape. Existing measures of landscape have mainly been applied to combinatorial problems considered in complexity theory. We show that some of these measures can be useful for testing problems as well. The diameter and autocorrelation are retained to study the adequacy of alternative settings of SA parameters. A new measure, the Generation Rate of Better Solutions (GRBS), is introduced to monitor convergence of the search process and implement stopping criteria. The measurement approach is experimented on various case studies, and allows us to successfully revisit a problem issued from our previous work on testing control systems.

Keywords – software testing, metaheuristic search, simulated annealing, measurement.

1 Introduction

Metaheuristic search techniques (Rayward-Smith et al., 1996) have proven useful to solve complex optimization problems. Their generality makes them capable of very wide application. For instance, they have gained attention in the field of software engineering (Clarke et al., 2003) and more specifically of software testing (McMinn, 2004).

Genetic algorithms (Holland, 1975) and simulated annealing (Kirkpatrick et al., 1983) are two examples of metaheuristic techniques used to automate the generation of test data. The underlying testing problems are as diverse as: structural testing (Jones et al., 1996) (Tracey et al. 1998a) (Pargas et al. 1999) (Michael et al., 2001) (Wegener et al., 2002), mutation testing (Adamopoulos et al., 2004), pre- and post-condition testing (Tracey et al., 1998b), exception testing (Tracey et al., 2000a), determination of worst case execution time (Wegener et al. 1997) (Gross et al., 2000), testing high-level requirements of control systems (Schultz et al., 1995) (Abdellatif-Kaddour et al., 2003b) (Wegener and Buehler, 2004). Whatever the testing problem, it is first reformulated as an optimization problem with respect to some objective function called *fitness function* (usually for maximization problems) or *cost function* (for minimization problems). Then, a search run involves a series of test executions. Each execution allows a candidate solution (i.e. a generated test case, or test sequence) to be evaluated with respect to the objective function. Evaluation results are used to guide the generation of new candidate solutions in the run.

Metaheuristic search techniques are not ready-to-use algorithms. Rather, they define generic strategies that must be instantiated for the specific problem under consideration. A number of implementation choices have to be done, and these choices affect performance. It is generally recommended to experiment with alternative implementations of a strategy, or even with a variety of strategies. It is not unfair to say that the tuning of the search requires a great deal of effort, can indeed be quite expensive, and is not guaranteed to yield adequate choices. Section 2 of the paper provides a concrete example of this bad situation, which we faced while applying simulated annealing (SA) search to a testing problem. It reflects the difficulty of understanding the dynamics of metaheuristics.

Of course, the difficulty is not specific to the field of software testing. In the general literature of metaheuristics, there is active research to investigate relations between search problem characteristics and the performance of search techniques. The long-term aim is to gain both fundamental and practical insights into the matter. Central to this research is the concept of *landscape*. If we consider the fitness (or cost) as defining the "height" of a solution, the landscape metaphor gives rise to the visual image of peaks, valleys, and plateaus. Intuitively, the "relief" of the landscape should have a strong impact on the dynamics of exploration strategies. A number of measures have been proposed to characterize landscapes, and predict – or explain – the behavior of search techniques applied to them. They will be presented in Section 3. To date, the most advanced results have concerned combinatorial problems considered in complexity theory, like the Traveling Salesman Problem (TSP) or the Boolean satisfiability problem (SAT).

For testing problems, research is far less mature. Still, we are aware of recent work referring to landscape as an explanatory concept. The survey of (McMinn, 2004) revisits previous testing work by discussing the associated landscapes (e.g., it is shown that some fitness functions induce flat plateaus along which the search is provided no guidance). In the same spirit, (Wegener and Buehler, 2004) compare the relief of two candidate landscapes to choose the most adequate fitness function. Further developments in landscape analysis are expected to emerge in the near future, as advanced topics in metaheuristic search get transferred to the field of software testing (or more generally to the field of software engineering: the manifesto by Harman and Jones, 2001, opens perspectives in that direction).

This paper is in the lineage of emerging work we have just mentioned. We investigate a measurement approach to the characterization of testing landscapes for Simulated Annealing. The aim is to support the implementation choices that have to be made to tune the search process. A failure story reported in Section 2 illustrates the significance of this issue. Section 3 presents a number of measures defined in the framework of combinatorial problems, and discusses their applicability to the testing problem under consideration. Two measures of landscape, the diameter and autocorrelation, are retained. They are experimented in Section 4, using both small examples and the case study of Section 2. The latter provides a challenging case for SA parameterization, since it defeated our previous attempts to find an adequate tuning. Besides landscape analysis, Section 5 considers the characterization of process runs, that is, of the trajectories followed when exploring the landscape. Ideally, we would like to detect runs that are unlikely to succeed before test resources are exhausted, e.g. detect runs trapped in flat regions. In this way, useless runs are stopped and the remaining resources are spent on exploring other trajectories. A new measure, the Generation Rate of Better Solutions (GRBS), is introduced to monitor runs and implement stopping criteria. Section 6 concludes with future directions.

2 Problem Statement

This section explains the motivation of our work. It first introduces the testing problem that yielded us to investigate the use of metaheuristic search. Then, we provide feedback from an experimentation of Simulated Annealing (SA) on a case study. Our experience exemplifies the difficulty of finding an adequate implementation of SA search.

2.1 The Testing Problem: Exploring Dangerous Scenarios at System Level

We have defined a strategy for testing high-level requirements of cyclic real-time control systems (Abdellatif-Kaddour et al., 2003a). The aim is to validate a system with respect to a safety-critical property, that is, to test the system and observe whether the property is violated or not. For the high-level properties we are interested in, violation typically results from improper interactions between a control program and its controlled environment, when physical faults affect the devices. Where formal verification is intractable, testing may be a pragmatic alternative. The test platform has to include a simulator of the environment that mimics the physical devices, the physical laws governing the controlled process, as well as the possible occurrence of faults in the devices. The test selection process should then try to favor those scenarios that will be the most "stressing" with respect to the target property. To address this problem, the proposed test strategy consists of a stepwise construction of test scenarios. Each step explores continuations of the "dangerous" scenarios found at the previous step, using heuristic search techniques.

In order to illustrate the notion of stepwise construction, let us take the example of the boiler case study experimented in (Abdellatif-Kaddour et al., 2003a). The steam boiler problem (Abrial et al. 1996) is a well-known case study for which high-level requirements, control program code, and a software simulator of physical devices were made publicly available. The most critical requirement is the avoidance of boiler explosion, which occurs whenever the water level falls outside a safety range (i.e. it is either too low, or too high). The water level is controlled and monitored by means of four pumps, four pump controllers sensing the state of the pumps (open, closed), a water level sensor, and a steam sensor measuring the quantity of steam which comes out of the boiler. Any of these ten devices may fail, and will remain faulty until repaired by a human operator. The control program has to detect failures, and must either continue to operate in a degraded mode or shut the system down. An overview of this case study, and of our test platform, is provided in Appendix A.

We used our test strategy to search for explosive scenarios due to improper account of device failures. The tested system consisted of the control program connected to the simulator. A test sequence is then a sequence of timed commands sent to the simulator, to tell it to mimic the failure of a given device. As an example, command Steam_fault(3) triggers a steam sensor failure at cycle 3 of operation. Prior to any test experiment, we determined what would constitute a dangerous scenario for this case study: a dangerous scenario would put the system in a state such that the control program has an erroneous perception of its environment. This includes situations such as:

- The control program does not detect the failure of a device.
- The control program wrongly detects the failure of a device that is not faulty.
- The actual water level in the boiler is outside the estimation range calculated by the control program.

Triggering a dangerous situation is adopted as an intermediate test objective at step 1. For example, the test experiments revealed that the control program is unable to detect a steam sensor failure occurring at cycle 3 (cycle 3 corresponds to the control program leaving its initialization mode). We retained dangerous scenario Steam_fault(3) as a prefix for further exploration at step 2. We ended up with explosive scenario: Steam_fault(3), Pump2_fault(4), WaterLevel_fault(8).

In (Abdellatif-Kaddour et al., 2003a), exploration at each step was performed using the simplest heuristic technique, namely random sampling. The technique was surprisingly effective, since we found four different classes of explosive scenarios[1]. However, blind random sampling is not expected to be sufficient in most cases. This motivated our interest in more sophisticated search techniques, and in particular simulated annealing.

The reasons why we chose SA rather than, say, genetic algorithms (GAs) were the following:

- GAs work with a population of solutions (i.e., a population of test sequences in our case), and have to let the population evolve during several generations. In contrast, SA works with one solution at a time. It was perceived that SA would be less demanding in terms of total number of test experiments. This was an important criterion, as applying test sequences to the complete system (including both the control program and the simulator), is quite costly in execution time.

- GAs involve two evolution mechanisms, namely mutation and crossover. The effectiveness of crossover depends on the ability to produce highly fit offspring from highly fit (and possibly dissimilar) parents. How to define meaningful crossover operators was far from intuitive for our testing problem. It was deemed easier to focus on mutation-like operators, exploring a set of "neighbors" of a candidate solution. SA is a typical example of metaheuristic based on this principle.

Several experiments with SA were performed, some of which are reported in (Abdellatif-Kaddour et al., 2003b). The experiments investigated different implementation choices for the boiler problem. We present these choices below, as well as the results we got.

2.2 Implementation of Simulated Annealing

Figure 1 presents the generic SA algorithm for a minimization problem with solution space S. It searches for a solution $s \in S$ that minimizes a cost function $f: S \rightarrow \boldsymbol{R}$. The algorithm works by randomly selecting a new solution s' in the neighborhood of current solution s, using neighborhood operator $N: S \rightarrow 2^S$. Solutions with lower cost ($\delta \leq 0$) are always accepted. Moves to inferior solutions ($\delta > 0$) may be accepted: the probability of acceptance depends on the magnitude of δ and on a control parameter t called temperature. Starting from t_0, the temperature is gradually reduced during the search (according to cooling schedule C) so as to progressively decrease the acceptance rate of inferior

[1] These explosive scenarios are not due to bugs in the control program. They originate from the very definition of the degraded modes in the requirements (specification flaw). To the best of our knowledge, only one of the scenario classes was already identified in the literature.

solutions. The search is stopped when a pre-defined stopping condition becomes true (e.g., the maximal number of iterations is reached).

```
Select an initial solution s
Select an initial temperature $t = t_0 > 0$
Initialize the number of iterations $i = 1$
LOOP while (stopping condition = false)
        Randomly select $s' \in N(s)$
        $\delta = f(s') - f(s)$
        IF ($\delta \leq 0$) THEN
                $s = s'$
        ELSE
                Generate random x uniformly in range [0,1]
                IF ($x < exp(-\delta / t)$) THEN $s = s'$ ENDIF
        ENDIF
        $i = i+1$
        $t = C(t, i)$
END LOOP
```

Figure 1 – Simulated Annealing Algorithm

A number of implementation choices must be made to tune the generic algorithm to a particular search problem. For the boiler case study, the solution space *S* is a set of test sequences, where each sequence is encoded as a list of timed fault commands. The precise definition of *S* (time window of the sequences under consideration, required prefix…) depends on the step of the test strategy, and several spaces are possibly explored at each step. In any case, the aim is to find explosive or dangerous scenarios in the target *S*. The implementation of SA search then involves the following settings.

- Cost function *f* – For some testing problems, the cost function definition may be obvious. For example, WCET problems are associated with measures of the execution time. In our case, determining an adequate cost function is far from trivial. The cost function has to assign the lowest cost to test sequences reaching the test objective, yielding the general form: *IF (boiler explosion OR dangerous situation observed) THEN return (0) ELSE …* Now, the difficulty lies in the determination of the *ELSE* part. How should the cost of two candidate sequences yielding neither an explosion nor a dangerous situation be compared? We experimented with several cost functions. Appendix B gives two examples: Cost1 was used in unpublished experiments while Cost2 is the one used in (Abdellatif-Kaddour et al., 2003b). Both functions involve some constant parameters K_i, whose value was determined empirically.

- Neighborhood *N* – Given a test sequence, it seems reasonable to consider neighbors that differ in only one fault command. The neighborhood operator either removes one fault, or adds one, or changes the date of occurrence (see Appendix B).

- Cooling Schedule *C* – Two commonly used schedules are (1) the geometric variation of temperature: $t_{i+1} = \alpha\ t_i$ where α is chosen close to 1.0, and (2) the schedule proposed by (Lundy and Mees, 1986): $t_{i+1} = t_i / (1+\beta\, t_i)$ where β is chosen close to zero. We experimented with both. Note that calibration of parameters α (resp. β) and t_0 required preliminary executions of SA on the boiler system.

- Stopping Condition – The search is stopped when either the test objective is fulfilled ($f(s) = 0$) or the maximal number of iterations is reached (i = *MAX_ITER*). Since any execution of the boiler system is costly in time, and since the test strategy involves exploration of several search spaces, we cannot afford large values of *MAX_ITER*. We experimented with *MAX_ITER* = 100 and 200. These are small values compared to common usage of SA, which gives more time for the algorithm to converge toward an optimum.

Our experience is that the implementation of SA search requires substantial effort. The tuning procedure is largely ad hoc. Many trials are necessary to investigate implementation choices and to study alternative combinations of parameter settings.

2.3 Experimental results

In spite of the invested effort, SA performance turned out to be quite disappointing for the Boiler example. It never outperformed random sampling. For some of the explored spaces, the high density of zero cost solutions could explain this result: any sophisticated technique was bound to be less cost-effective than random sampling. But poor results were also obtained in less trivial cases.

Let us take the example of one of the search spaces explored at step 2 of the strategy. Table 1 gives comparative results of random sampling and SA. The experiments involved 35 runs of each technique. A run is successful as soon as a zero cost sequence is found. It is stopped at the corresponding iteration. Unsuccessful runs are stopped at *MAX_ITER* = 200. Table 1 provides both the number of successful runs, and the total number of iterations for the 35 runs. Note that the two implementations of SA only differ in the cost function: all the other parameters are set the same.

	Random sampling	SA Cost1	SA Cost2
Successful search	26	25	16
Iterations (35 runs)	4187	3778	4453

Table 1 – Experimental results for one search space of the Boiler example

The best SA implementation (the one with Cost1) does not have a greater number of successful runs than random sampling. By comparing the total number of iterations, it might be concluded that SA speed is slightly higher (it takes 3778 iterations to find 25 zero cost sequences, versus 4187 iterations and 26 sequences). But the "improvement" is too insignificant to be worth the effort.

In (Abdellatif-Kaddour et al., 2003b), such results led us to propose a variant of the SA algorithm (i.e., the core algorithm in Figure 1 was modified), which performed well on the Boiler example. But we are aware that this specific variant might be inadequate for other examples. Each time a new control system is studied, the tuning effort needs to be made again and again.

This motivates our interest in investigating improvements of the tuning procedure. Indeed, for the testing problems we address, SA search cannot be a realistic option unless systematic approaches are proposed to support implementation choices. For example, the experiments of Table 1 *a posteriori* show that Cost2 is a worse choice than Cost1. Intuition is a poor guide to predict this result. There is a need for more reliable means to assess the quality of alternative settings, and this is all the more crucial as the testing time has to be kept reasonable.

It turns out that fundamental work on search techniques has settled a framework to address this issue. A number of measures have been proposed to characterize search problems and the behavior of heuristics applied to them. In this paper, we investigate the predictive power of such measures for tuning SA search in the case of testing problems.

3 Measures for Characterizing Search Problems and Metaheuristics

Fundamental research on metaheuristics aims to establish relations between measures of problems and the dynamics of metaheuristic search. The studied problems are most often NP complete problems considered in complexity theory, like graph problems or constraint satisfaction problems.

Existing measures can be classified into three broad categories, corresponding to the structure they characterize (Belaidouni and Hao, 2000). The underlying structures are:

- the *search space*, defined by a couple *(S, f)* where *S* is a solution space and *f* a cost function assigning a real number to each point in the space;

- the *search landscape*, defined by a triplet *(S, f, N)* where N is a neighborhood operator connecting each point in S to a non-empty set of neighbors;
- the *process landscape*, defined by a quadruplet *(S, f, N, φ)* where φ is a search process based on the notion of neighborhood.

Examples of measures for each structure are provided in Sections 3.1 to 3.3. A more complete description of measures and of related work can be found in (Belaidouni, 2001). Section 3.4 discusses applicability to our testing problem.

3.1 Characterizing the Search Space

For the search space structure, typical examples of measures are the variation range $[f_{min}, f_{max}]$ of cost values, or the number of optimal solutions in S. A finer measure concerns the *density of states* (dos), giving the frequency of each cost value in the search space (Rosé et al., 1996). For some problems, dos may be determined analytically. But most often it has to be approximated by using a sampling procedure, which can be expensive to properly account for the least frequent values.

3.2 Characterizing the Search Landscape

The search landscape has been the most studied structure. The introduction of a neighborhood operator N makes it possible to define the distance between points s and s' in S. It corresponds to the minimal number of times N must be applied to move from s to s'. The notion of distance is underlying all measures for characterizing the search landscape.

3.2.1 Diameter

The diameter D of a search landscape is the maximal distance between two points in S. Intuitively, small diameters should be more favorable than large ones, because any point can potentially be reached quicker (in terms of successive applications of N). Suppose D is large. Then, it may take a large number of iterations to get to an optimal solution, even in the ideal case where the search process would select the shortest trajectory between the starting point and the target solution.

The diameter can usually be determined analytically (for landscapes of typical NP-complete problems, see examples of analytical expressions in Angel and Zissimopoulos, 2000).

3.2.2 Autocorrelation

The autocorrelation measure has been introduced by (Weinberger, 1990) to characterize the *ruggedness* of a landscape. Intuitively, 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 dissimilar. The latter situation indicates that *f* and *N* are not mutually adequate to guide the search.

The autocorrelation ρ_d measures the variation of cost for points that are at distance *d*. Most often, measurement is focused on ρ_1, which is deemed the most important value to know. Exact evaluation of the autocorrelation is not possible unless the distribution of cost values is known. In practice, ρ_1 is estimated by means of a random walk whose steps consist in moving to a new point chosen randomly among the neighbors of the current point (see Figure 2). The underlying assumption is that the statistics of the cost sequences generated by a random walk are the same, regardless of the starting point chosen: the landscape has to be *statistically isotropic* (Weinberger, 1990). An estimate of ρ_1 is then (Hordijk, 1996):

$$\hat{\rho}_1 = \frac{\sum_{i=0}^{T-2}\left(f(s_i)-\overline{f}\right)\left(f(s_{i+1})-\overline{f}\right)}{\sum_{i=0}^{T-1}\left(f(s_i)-\overline{f}\right)^2}$$

where T is the size of the sample and $\overline{f}$ the arithmetic mean of the cost values in the sample. If ρ_1 is close to one, the landscape is smooth. If ρ_1 is close to zero, neighboring cost values are unrelated and neighborhood search techniques are not expected to be effective.

Autocorrelation (or derived measures) has been used to explain search performance on landscapes associated with NP-complete problems (Stadler and Schnabl, 1992) (Angel and Zissimopoulos, 1998) (Angel and Zissimopoulos, 2000) as well as on artificial landscapes whose ruggedness can be tuned by control parameters (Hordijk, 1996).

```
Select a starting point $s_0$
Sample[0] = $f(s_0)$
FOR i=1 to T-1 DO
        Randomly select $s_i \in N(s_{i-1})$
        Sample[i] = $f(s_i)$
END FOR
```

Figure 2 – Random Walk Algorithm

3.2.3 Other Measures

While autocorrelation characterizes the variation of costs at a fixed distance, other measures are focused on the variation of distances at a fixed cost. For example, such measures are used by (Belaidouni and Hao, 2000) to study landscapes of the MAX-CSP problem. A practical difficulty is that their experimental evaluation requires the production of a sample of solutions for each cost level. In (Belaidouni and Hao, 2000), the authors have to apply metaheuristic search to produce the samples. There are also measures characterizing the joint variation of distances and costs, like the Fitness Distance Correlation (FDC) (Jones and Forrest, 1995), used in the framework of genetic algorithms and genetic programming (Vanneschi et al., 2003). The FDC definition involves the notion of distance to the nearest global optimum, which restricts its applicability to problems with known optima.

Finally, it is worth mentioning work studying the topology of landscapes in terms of local optima and plateaus, which are regions the search can be trapped in (Frank et al., 1997). For small instances of problems, the number and size of such regions can be exhaustively determined, as in (Yokoo, 1997). Also, for some simple landscapes, the number of local optima can be approximated analytically (see e.g. the TSP landscape analysis by Stadler and Schnabl, 1992). However, in the general case, approximation has to be achieved via expensive experiments (Eremeev and Reeves, 2003).

3.3 Characterizing the Process Landscape

The process landscape should be the most informative structure, since it accounts for all parameters of the search implementation. It is also the most difficult (and the most expensive) to characterize, because analytical analysis is not possible and empirical analysis involves running the search process.

Most of the measures defined for the search space and search landscape can be transferred to the process landscape. For example, the process cost density (pcd) (Belaidouni and Hao, 2002) is analogous to the density of states (dos) defined for search spaces (cf. Section 3.1), using process φ as the sampling procedure. It is worth noting that dos and pcd are not the same, because φ introduces a bias in the exploration of solutions. Similarly, the number and size of basins of attractions for φ should be related to the local optima of the search landscape, but the relation is not trivial.

When characterizing the process landscape, an acute problem is the variability of behavior from one run of φ to the next: obtaining statistically meaningful measures can be practically impossible. The problem already existed for the characterization of search landscapes using a random walk. However, it is much more acute when the samples are produced by sophisticated metaheuristics. Indeed, for the boiler example, we empirically observed variability of SA behavior, depending on the starting point of the search.

3.4 Discussion

While a number of measures have been proposed in the literature, there is currently no consensus on which ones to use in which case. Generally speaking, a single measure can only have a limited explanatory power. For example, autocorrelation is one of the most acknowledged measures but can be insufficient to distinguish between two search landscapes of different difficulty.

In spite of these limitations, we will show that a measurement approach can still be useful to implement SA search for our testing problem. The choice of measures has to be guided by our specific needs. First, we cannot afford a large number of experiments to get an estimation of measures, because execution of the complete system is computationally costly. Second, our aim is to tune SA search to get zero-cost solutions, not just low cost solutions. This is so because non-zero solutions do not fulfill the test objective: in the boiler example, they correspond to safe scenarios triggering neither a boiler explosion nor a dangerous situation. Let us discuss the implication of these needs.

The constraint in the number of system executions leads us to exclude the most costly measures for characterizing the search landscape (e.g., the measures mentioned in Section 3.2.3), as well as all measures for characterizing the process landscape (see Section 3.3). As regards the latter measures, let us recall that their estimation is anyway problematic due to the variability of SA behavior.

The fact that we are not interested in sub-optimal solutions limits the insight that can be gained from some measures. Let us take the example of dos and pcd. A search space will be considered all the easier as the cost distribution exhibits a low mean and high variance (i.e., it is not difficult to find low cost solutions, and very low values departing from the mean are not too scarce). Similarly, a process landscape will be considered better than another if pcd analysis shows that it tends to generate lower cost solutions. Such results are relevant to problems for which low cost, but possibly sub-optimal, solutions are quite acceptable. They are not relevant in our case because sub-optimal solutions are useless with respect to the test objective, as previously explained.

We finally retain two measures for characterizing the search landscape: the diameter D and autocorrelation ρ_1. They will be used in Section 4 to guide the choice of “good” combinations of a cost function f and neighborhood operator N, for SA search.

How to characterize the process landscape remains an open issue. In Section 5, we will adopt a less ambitious approach. We will not attempt to refine tuning of the process landscape. Rather, we will attempt to make the best possible use of a given process landscape. Since behavior depends on the starting point of the search, we will try to detect runs that are unlikely to succeed in the allowed number of iterations, so as to stop them and restart search from another point. This will lead us to propose a new measure to monitor the behavior of a run and implement stopping criteria.

4 Characterizing the Search Landscape

The diameter and autocorrelation measures are used to identify search landscapes that are potentially well-suited to SA search. The tuning process shall retain settings of f and N such that:

- Diameter D is significantly lower than *MAX_ITER*, the allowed number of iterations per run.
- Autocorrelation ρ_1 is high.

The diameter criterion is intended to ensure that an optimal solution has a chance to be reached before the end of the run, whatever the starting point of the search. The autocorrelation criterion identifies landscapes for which the principle of neighborhood search is not irrelevant. Equipped with these criteria, our challenge is to revisit the boiler example and to determine an adequate parameterization of SA search for it.

Before addressing the boiler example, we study the relevance of the criteria for simpler problems. The first one, QAP (Section 4.1), has been extensively studied in the literature of metaheuristic search. It is considered here for comparison purposes. It provides a typical example of the combinatorial problems to which diameter and autocorrelation measures are usually applied. It is also one of the problems for which SA search is known to be efficient. The next two problems, Cal1 and Cal 2 (Section 4.2), are loosely inspired from a calendar testing problem. They are used to perform a large number of controlled experiments on "good" and "bad" landscapes. We let f and N vary, and try to establish a relation between the corresponding D and ρ_1 measures on the one hand, and SA efficiency on the other hand. This makes it possible to study the discriminative power of the criteria. The boiler example is then revisited in Section 4.3.

4.1 The QAP Landscape

The Quadratic Assignment Problem (QAP) is a typical representative of combinatorial problems studied in the literature of metaheuristic search. It is a NP-hard problem generalizing the Traveling Salesman Problem. A number of search techniques have been experimented on QAP instances (Connolly, 1990) (Taillard, 1991) (Maniezzo et al., 1995) (Merz and Freisleben, 2000). SA search is known to be very efficient for QAP, using some standard settings of f and N. Actually, cost function f is already determined by the problem definition, and it is only N that needs to be adequately chosen. The standard N for QAP is the so-called *2-exchange neighborhood* (see e.g., Angel and Zissimopoulos, 2000, for a definition of this neighborhood operator). Thus, the QAP example provides us with the opportunity to investigate characterization of a landscape known to be "good".

The experiments involve the NUG15 instance of QAP proposed by (Nugent et al., 1968)[2]. It has four solutions supplying an optimal cost, in a search space of 15! elements. In order to check our ability to reproduce known results, we retain the same SA implementations as the ones described in (Connolly, 1990). The maximal number of iterations is taken as *MAX_ITER* = 5000 to keep the same order of magnitude as in Connolly's experiments (which involved 5250 iterations for this problem instance). Before experimenting with SA search, we first characterize the underlying search landscape.

Diameter criterion – With the 2-exchange neighborhood, $D = 14$ which is obviously much lower than the allowed number of iterations.

Autocorrelation criterion – The experimental evaluation of ρ_1 involved 100 random walk runs (see Figure 2 for the random walk algorithm), with 1000 iterations per run. Note that this effort is unusually high as a single run should be necessary to perform estimation: we wanted to confirm the expected stability of estimates for this landscape. We obtained $\rho_1 = 0.74$ (standard deviation 0.01). This gives an idea of what should be considered a sufficiently high autocorrelation measure.

For this "good" landscape, we confirm the expected efficiency of SA search. In Table 2, L&M1 and L&M2 correspond to the two SA implementations described in (Connolly, 1990): L&M1 is standard SA with the Lundy and Mees cooling schedule, and L&M2 is a variant that yielded better results for a range of QAP instances. Whatever the SA implementation, any of the four optimal solutions is found by at least 21% of successful runs. Whereas random sampling fails to produce any optimal solution, SA search is efficiently guided by the structure of the underlying landscape.

	Random sampling	L&M1	L&M2
Percentage of successful runs (1000 runs, *MAX_ITER* = 5000)	0%	8%	11.1%

Table 2 – SA efficiency for QAP (NUG15 instance)

4.2 Landscapes for two Calendar Problems

After having characterized a known landscape, we now focus on landscapes for two artificial problems, loosely inspired from a calendar problem experimented by (Tracey, 2000b) in the field of software testing. For these two artificial problems, noted Cal 1 and Cal 2, the solution space is the set of dates from January 1, 1900 up to December 31, 3000. Each date is encoded by a triplet (*day, month,*

[2] This problem instance, as well as many other ones, is archived on Eric Taillard's page: http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/qap.dir/qap.html

year). We make the simplifying assumption that there are exactly 365 days per year (there is no account for leap years), hence the space contains 401,865 solutions. The search problem is then to find a zero cost date in the space. Cal 1 has exactly one optimum to be found (arbitrarily taken as December 31, 2000), while Cal 2 has two (December 31, 2000 and February 2, 2500). In each case, we experimented with 24 different search landscapes, by making *f* and *N* vary. Cal 1 is used to illustrate the results below. A complete report of all experiments can be found in (Abdellatif-Kaddour, 2003c).

The four cost functions and six neighborhood operators used for Cal 1 are described in Appendix C. Basically, *Cost1* is a random cost function that is expected to yield poor results whatever the neighborhood definition. *Cost2*, as well as neighborhood operators *N1* and *N2*, are inspired from Tracey's work, and should fit well together. *Cost3* and *Cost4* have been designed in the same spirit as *N3-N6*, so that any combination of them is expected to supply a high correlation of neighboring costs. The Cal 2 cost functions are similar to the Cal 1 ones, but are slightly adapted to account for the additional optimum. The neighborhood operators are kept exactly the same.

Let us now characterize the landscapes resulting from all *f* x *N* combinations. The maximal number of iterations per run is set to 2000.

Diameter criterion – Table 3 shows the diameter values for all neighborhood operators. Bold values indicate that the criterion is passed, while italic values denote diameters that are judged too large compared to the allowed number of iterations. As can be seen, *N3* and *N4* are rejected as bad choices for *MAX_ITER* = 2000.

Autocorrelation criterion – Table 4 shows ρ_1 estimates for the 24 landscapes of Cal 1. In each case, experimental evaluation involved 100 random walk runs, with 1000 iterations per run. A cell of the table indicates the mean value of ρ_1, followed by the standard deviation put in brackets. Ten landscapes (with ρ_1 in italic numbers) do not pass the autocorrelation criterion. Not surprisingly, all landscapes involving *Cost1* are rejected. Moreover, we learn that *N3-N6* are not well-suited to *Cost2*, while *N1-N2* fit well with *Cost3-Cost4* although they have not been designed for that purpose. Cal 2 landscape analysis yields similar conclusions.

Putting together the diameter and autocorrelation criteria, 10 landscapes are retained for each of the Cal 1, Cal 2 problems, and 14 landscapes are rejected. We then assess the relevance of these decisions by measuring SA search efficiency for all landscapes. Table 5 gives the percentage of successful runs for the Cal 1 problem, observed from a sample of 1000 SA runs per landscape. It can be seen that the 14 rejected landscapes (whose results are displayed in italic characters) exactly correspond to the 14 worst scores. This result is encouraging, in spite of the fact that the diameter and autocorrelation measures have a limited discriminating power: the 10 retained landscapes do supply the best scores but these scores exhibit a large variation (ranging from 6.3% to 65.1%). Note, however, that the lowest score (6.3%) is still significantly higher than that of random sampling, which was 0.4% in the Cal 1 case study. Similar results are obtained for Cal 2.

Hence, despite limitations, the proposed approach should be relevant to reject bad landscapes, and retain only those that are reasonably well suited to SA search (while possibly not "best" suited). We are now equipped to revisit the Boiler example.

	N1	N2	N3	N4	N5	N6
Diameter	**550**	**1100**	*6700*	*13400*	**250**	**500**

Table 3 – Diameter of the calendar landscapes

	N1	N2	N3	N4	N5	N6
Cost1	*0 (0)*	*0 (0)*	*0 (0)*	*0 (0)*	*0 (0)*	*0 (0)*
Cost2	**0.82 (0.08)**	**0.95 (0.04)**	*0.22 (0.09)*	*0.22 (0.09*	*0.36 (0.19)*	*0.33 (0.17)*
Cost3	**0.99 (<10^{-2})**	**0.99 (<10^{-2})**	**0.89 (<10^{-2})**	**0.89 (<10^{-2})**	**0.98 (0.02)**	**0.98 (0.01)**
Cost4	**0.99 (<10^{-2})**	**0.99 (<10^{-2})**	**0.99 (<10^{-2})**	**0.99 (<10^{-2})**	**0.99 (<10^{-2})**	**0.99 (<10^{-2})**

Table 4 – Autocorrelation of the Cal 1 landscapes: mean (stand. dev.)

	N1	N2	N3	N4	N5	N6
Cost1	*0%*	*0%*	*0%*	*0%*	*0.2%*	*0.2%*
Cost2	**6.3%**	**16.1%**	*0%*	*0%*	*1.4%*	*1.4%*
Cost3	**47.6%**	**65.1%**	*2.1%*	*2.1%*	**16.9%**	**16.8%**
Cost4	**60.8%**	**9.5%**	*3.7%*	*3.7%*	**40.3%**	**40.2%**

Table 5 – Percentage of successful SA runs on the Cal 1 landscapes (1000 runs, *MAX_ITER*=2000)

4.3 The Boiler Search Landscape

Compared to the previous case studies, the Boiler example is one for which random sampling is fairly efficient. The most difficult search space is the one mentioned in Section 2.3, where random sampling found 26 zero-cost solutions using a total number of 4187 iterations (see Table 1). We will focus on the corresponding search problem. As regards SA efficiency, we observed that the choice of the cost function had a strong impact, but the best function proposed so far did not significantly outperform random sampling. We will try to improve the situation by tuning the cost function. The size of the target search space is 10,077,695. The allowed number of iterations is *MAX_ITER* = 200.

Let us now characterize the two landscapes studied in Section 2.3.

Diameter criterion – The chosen neighborhood operator yields $D = 9$, hence the diameter criterion is passed.

Autocorrelation criterion – For both landscapes, the experimental evaluation of ρ_1 involved 10 random walk runs, with 100 iterations per run. Table 6 shows the results. The autocorrelation measure does not discriminate between Cost1 and Cost2. In both cases, neighboring costs are not unrelated but ρ_1 values are smaller than the ones accepted for the QAP, Cal 1 and Cal 2 problems (the retained landscapes always had $\rho_1 \geq 0.74$). Hence, the results do not reveal strong inadequacy of the landscapes, but suggest that it may be preferable to design a cost function with higher autocorrelation.

	Mean	Stand. Dev.
Cost1	0.52	0.17
Cost2	0.53	0.13

Table 6 – Autocorrelation of the Boiler landscapes

Rather than inventing an entirely new cost function, we go back to the generic definitions of Cost1 and Cost2 in Appendix B, expressed in terms of symbolic parameters K_i. Let us recall that the K_i calibration was based on trials and empirical judgment. We now try a more systematic approach, by making the K_i values vary, and retain the valuation supplying the best autocorrelation.

As regards Cost1, we observed the best autocorrelation for $K_1 = 100$, $K_2 = 0$, $K_3 = 10000$. Improvement is not drastic, since $\rho_1 = 0.62$ (stand. dev. 0.14). Still, we retain the corresponding cost function, noted $\text{Cost1}_{\text{bis}}$.

As regards Cost2, we failed to obtain any significant improvement. This may suggest that the generic definition is not well suited to the search problem and/or the chosen neighborhood operator.

We now study whether the new Cost 1_{bis} function manages to improve SA efficiency. Table 7 provides the results obtained by 35 runs (*MAX_ITER* = 200). The previous results of random sampling and Simulated Annealing are recalled for comparison purposes. It is clear that the Cost1_{bis} landscape is the best for SA search. Moreover, observed performance is now significantly better than that of random sampling. Considering our initial failure to properly tune SA search, these results confirm that the proposed measurement criteria should be much sounder than *ad hoc* judgment.

	Random sampling	SA Cost1	SA Cost2	**SA Cost1_{bis}**
Successful search	26	25	16	**33**
Iterations (35 runs)	4187	3778	4453	**2573**

Table 7 – SA search improvement for the Boiler example

5 Stopping Criteria for Process Runs

Contrary to what has just been done on the search landscape, we do not try to improve the process landscape. For the reasons explained in Section 3.4, our objective is more modest: we attempt to make the best possible use of a given process landscape. This is done by monitoring the process runs. Each run corresponds to a trajectory in the landscape, which may or may not reach an optimal solution depending on the starting point of the search and the allowed number of iterations. The monitoring approach aims to decide whether the current run is likely to succeed before the allowed number of iterations. If the decision is negative, the run is stopped and search is restarted from another point.

Stopping criteria have also been studied by others in the framework of search-based testing. In (Lammermann and Wegener, 2005), the allowed search effort is tuned to structural coverage goals: the stopping criteria are *a priori* determined from software measures. In (0'Sullivan et al., 1998), on-line monitoring of search convergence is used to determine whether the current run should be stopped or continued. As an *a priori* estimation is impossible for us, our approach is more in the spirit of the latter work. However, its implementation is quite different since (0'Sullivan et al., 1998) use genetic algorithms and analyze convergence for populations of solutions. Section 5.1 presents the principle of our monitoring approach and the related stopping criteria. Section 5.2 uses the QAP and calendar problems to calibrate the criteria. A trade-off must be found between too sensitive criteria (stopping runs that would have been successful) and too restrictive ones (not stopping unsuccessful runs). The calibrated criteria are then applied to the Boiler example in Section 5.3.

5.1 Principle of the Approach

A new measure is introduced to characterize a process run, called the Generation Rate of Better Solutions (GRBS):

$$\text{GRBS} = \frac{\text{number of generated solutions decreasing the cost}}{\text{total number of generated solutions}}$$

The GRBS value is not very interesting *per se* because it is unstable, evolves during a run, and varies from one run to the next. Actually, it is precisely the evolution during a run that is the focus of our monitoring approach.

Whatever the case study (QAP, calendar problems, Boiler), we observed that GRBS evolution for *successful* runs can be schematized as in Figure 3. Three phases may be identified. Phase 1 typically characterizes the beginning of the search at a point where cost is high and easy to improve: lower cost solutions are produced at a high rate. At the end of Phase 1, low cost solutions are reached and improvement becomes difficult: the steep decrease of GRBS (Phase 2) indicates that search is approaching the optimal solution that will be found by this successful run. Phase 3 characterizes the last iterations of the run, exploring a small subspace in the neighborhood of the optimal solution. Of course, Figure 3 is only a schematic view. The actual duration of each phase depends on the run, and it may be the case that a phase is not present. Concrete examples of runs with no Phase 1 or no Phase 3 are shown in Figure 4.a.

An interesting observation is that the duration of phases can – to a certain extent – be used to distinguish successful and unsuccessful runs. We observed empirically that *unsuccessful* runs always correspond to long durations of either Phase 1 or Phase 3, where "long" refers to the maximal number of iterations allowed for the runs. Concrete examples are shown in Figure 4.b.

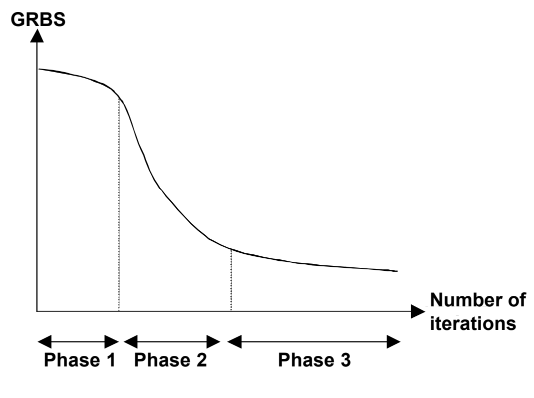
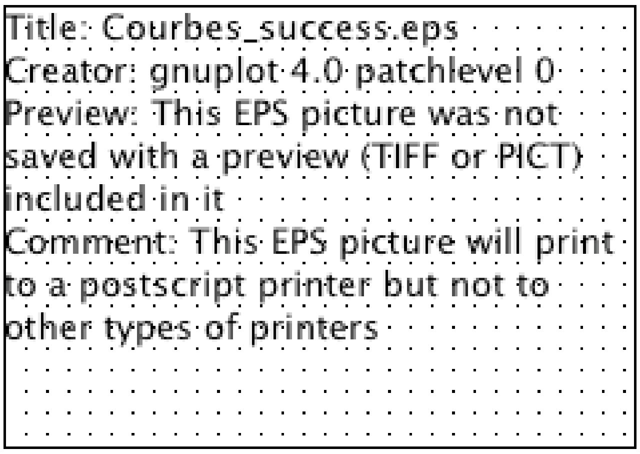
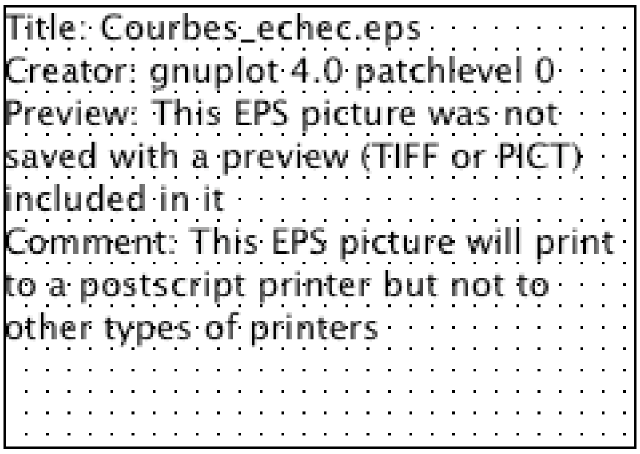


Figure 3 – Generic GRBS evolution for successful runs


Title: Courbes_success.eps
Creator: gnuplot 4.0 patchlevel 0
Preview: This EPS picture was not saved with a preview (TIFF or PICT) included in it
Comment: This EPS picture will print to a postscript printer but not to other types of printers

(a) Examples of successful runs


Title: Courbes_echec.eps
Creator: gnuplot 4.0 patchlevel 0
Preview: This EPS picture was not saved with a preview (TIFF or PICT) included in it
Comment: This EPS picture will print to a postscript printer but not to other types of printers

(b) Examples of unsuccessful runs

Figure 4 – Instances of GRBS evolution (Cal 1, Cost 4 x N6 landscape)

A long Phase 1 yields high GRBS values to be collected during the whole run. The starting point of the search has a high, easy to improve cost. Still, it turns out that the process moves fail to reach a low cost configuration: better solutions are continuously produced at a high rate, but each solution only provides a slight improvement compared to the previous one. Hence, when the maximum number of iterations is reached, the achieved cost value is still far away from the optimum.

For unsuccessful runs with a long Phase 3, it is observed that after transient phases 1 and 2, better solutions are scarcely produced during a large number of iterations (until *MAX_ITER* is eventually reached). The stagnation may indicate that the search process is stuck at a local optimum or trapped in a flat region.

It would be desirable to stop such runs as soon as possible, to avoid useless iterations. Two complementary stopping criteria are proposed, based on the on-line monitoring of the GRBS. The first (resp. second) criterion determines whether Phase 1 (resp. Phase 3) is too long for the run to possibly succeed within the allowed number of iterations. Both checks are performed in parallel, and the run is stopped whenever one criterion becomes true.

- **Phase 1 too long –** $GRBS \geq Threshold\_High$ **during** $max(\alpha\, MAX\_ITER, D)$ **iterations.**
  In the proposed formulation, Phase 1 of the search is identified by a GRBS value exceeding a threshold high value. The judgment that Phase 1 is "too long" must then be relative to the maximal number of iterations: this is tuned by means of parameter α with $0 < \alpha < 1$. In addition to this, it seems reasonable not to stop the run before the process has spent some minimal amount of time to explore the landscape. In the proposed formulation, the exploration time cannot be shorter than diameter *D* of the landscape.

- **Phase 3 too long –** ***GRBS ≤ Threshold_Low*** **during** ***max (β MAX_ITER, |N|)*** **iterations.**

  As above, the phase duration is tuned to the maximal number of iterations: this is done by means of parameter β with $0 < \beta < 1$. In addition to this, we impose that a Phase 3 search cannot be stopped before it has spent some time to explore the neighborhood of the solution reached at the end of Phase 2. This is why the triggering number of iterations cannot be smaller than the size |N| of the neighborhood operator, where |N| is defined as the maximal number of neighbors connected to any point.

In order to implement the checks, parameters α, β, *Threshold_High* and *Threshold_Low* must be instantiated. We experimented with a number of values using the QAP, Cal 1 and Cal 2 case studies. We also tried several monitoring schemes: on-line monitoring is started at the first iteration of the run, or after an initial number of iterations has been performed.

5.2 Calibration using QAP, Cal 1 and Cal 2

The best monitoring schemes, that consistently supplied satisfactory results for all case studies, were the following:

<u>Scheme 1</u>:	Monitoring is started after *MAX_ITER/10* iterations.
	Phase 1 too long: *Threshold_High* = 50%, $\alpha = 0.1$
	Phase 3 too long: *Threshold_Low* = 10%, $\beta = 0.1$
<u>Scheme 2</u>:	Monitoring is started at the first iteration.
	Phase 1 too long: *Threshold_High* = 50%, $\alpha = 0.25$
	Phase 3 too long: *Threshold_Low* = 10%, $\beta = 0.25$

It is worth noting that according to Scheme 1 (resp. 2), the search can never be stopped before *MAX_ITER/5* (resp. *MAX_ITER/4*) iterations.

The experiments involved the two process landscapes of QAP, and the 20 landscapes of Cal 1 and Cal 2 that passed the diameter and autocorrelation criteria (Abdellatif-Kaddour, 2003c). Exemplary results for the two retained schemes are shown in Table 8. They have been obtained for the QAP landscapes, by re-playing the runs from Section 4.1 with the monitoring schemes. The replay allows determination of the percentage of unsuccessful – but also successful – runs that are stopped by each scheme. Global search performance is then measured by the ratio (number of successes) / (total number of iterations). It accounts for the combined effect of false positives (successful runs are stopped) and false negatives (unsuccessful runs are not stopped).

	L&M1			L&M2		
	Without monitoring	**Monitoring scheme 1**	**Monitoring scheme 2**	**Without monitoring**	**Monitoring scheme 1**	**Monitoring scheme 2**
% unsucc. runs stopped	—	100%	100%	—	100%	100%
% succ. runs stopped	—	76.25%	50%	—	60.36%	18.92%
Success/iteration	$1.7\ 10^{-5}$	$1.9\ 10^{-5}$	$2.4\ 10^{-5}$	$2.4\ 10^{-5}$	$4.3\ 10^{-5}$	$5.5\ 10^{-5}$

Table 8 – Monitoring schemes applied to the QAP landscapes

From Table 8, it can be seen that all unsuccessful runs are detected and stopped. Unfortunately, the monitoring schemes may also stop a high percentage of *successful* runs. But we experimentally observed that those runs are most often the ones that succeed lately, i.e. towards the end of the 5000 allowed iterations. This explains why global search performance is not degraded.

For the two QAP landscapes, Scheme 2 turns out to be the best one. The supplied improvement is the same order of magnitude as the one supplied by using Connolly's L&M2 – rather than L&M1 – search process. However, considering the whole set of 22 landscapes, no scheme exhibits superiority over the other. The conclusions drawn from the 22 landscapes are the following:

- Both schemes manage to stop a satisfactory percentage of unsuccessful runs (> 50% for 12 landscapes).
- Successful runs are also stopped (< 20% for 13 landscapes), but these are generally runs that would have consumed almost *MAX_ITER* iterations before an optimal solution is found.
- Global search performance is either improved or unchanged, never degraded.

Hence, it seems that the monitoring schemes can significantly improve performance in some cases (as for the QAP example), and have a neutral effect in the worst case. It remains to be studied whether similar results can be obtained for the Boiler problem.

5.3 Experimentation with the Boiler Example

The retained schemes have been experimented on three Boiler landscapes, namely the ones with Cost1, Cost2 and Cost1$_{bis}$. The results shown in Table 9 correspond to the same runs as the ones in Section 4.3, replayed with monitoring.

	Cost1			Cost2		
	Without monitoring	Monitoring scheme 1	Monitoring scheme 2	Without monitoring	Monitoring scheme 1	Monitoring scheme 2
# unsucc. runs stopped	—	10 (out of 10)	10 (out of 10)	—	5 (out of 19)	5 (out of 19)
# succ. runs stopped	—	11 (out of 25)	12 (out of 25)	—	0 (out of 16)	0 (out of 16)
Success/iteration	$6.6\ 10^{-3}$	$7.2\ 10^{-3}$	$8.2\ 10^{-3}$	$3.6\ 10^{-3}$	$4.0\ 10^{-3}$	$4.0\ 10^{-3}$

(b) Cost1 and Cost2 landscapes

	Cost1$_{bis}$		
	Without monitoring	Monitoring scheme 1	Monitoring scheme 2
# unsucc. runs stopped	—	1 (out of 2)	1 (out of 2)
# succ. runs stopped	—	2 (out of 33)	2 (out of 33)
Success/iterations	$12.8\ 10^{-3}$	$13.0\ 10^{-3}$	$13.3\ 10^{-3}$

(a) Cost1$_{bis}$ landscape

Table 9 – Monitoring schemes applied to the Boiler landscapes

For these landscapes, search performance is similar whether or not the monitoring schemes are used. Note that, for the Cost1$_{bis}$ landscape, no drastic improvement could be expected because SA search is already quite efficient (indeed, there are only 2 unsuccessful runs). At least, our results confirm that performance is not degraded, which is also the case for Cost1 and Cost2 landscapes. This outcome was not granted, because the allowed number of iterations (200) is much lower than the one experimented for QAP (5000) or the calendar problems (2000). For such comparatively short runs, we could not be sure that the monitoring schemes would still be relevant. The fact that behavior remains consistent with previous observations is a positive point in favor of the reusability of the retained schemes. As previously, the stopped successful runs are the ones that succeed lately.

Our conclusions are then the following. Since the monitoring schemes do not degrade search performance, and might improve it in some cases, their use can be recommended. Scheme 1 or Scheme 2 can indifferently be chosen, as the supplied results seem quite similar. Note, however, that the monitoring approach cannot act as a replacement for finding adequate settings of SA parameters (like the cost function). At best, it may help not to waste test effort where search convergence is too slow. But the intrinsic difficulty of the landscape for SA search remains unchanged.

6 Conclusion

As argued by (Harman and Jones, 2001), the development of empirical studies is a crucial step to establish a body of knowledge in search-based software engineering. This paper is intended to contribute to this step. We adopted a *measurement* perspective, building upon fundamental work on metaheuristic search and combinatorial problems. To the best of our knowledge, the related literature has been little explored by the testing community. We are not aware of other studies investigating measures to tune search-based test data generation.

Our empirical study involved a testing problem (issued from previous work on testing control systems), as well as three simple problems chosen to experiment with a variety of landscapes. This allowed us to consolidate our approach and investigate whether consistent results could be obtained for several examples. Work was focused on simulated annealing search, and was driven by the two constraints of our original testing problem: 1) we cannot afford a large number of system executions and 2) sub-optimal solutions are useless with respect to the test objective. We believe these constraints are shared by many other testing problems. One characteristic of search-based test data generation is that the cost of a candidate solution has to be evaluated by supplying it to the system under test. Except for unit testing of simple functions, this evaluation may become very computationally costly. Moreover, as soon as the search process is intended to serve a test coverage objective (e.g., activate a program branch, or trigger an exception), sub-optimal solutions are not of interest. Hence, it is hoped that the retained approach could be relevant to a number of other testing problems.

We use the diameter and autocorrelation measures to study the adequacy of alternative settings of the cost function and neighborhood operator. These measures can only have a limited explanatory power, but are still useful to reject inadequate landscapes and retain those that are reasonably well suited to SA search. In our study, the approach allowed the Boiler problem to be successfully revisited, yielding a better cost function than the ones obtained by *ad hoc* trials. Our future work will consider additional empirical studies to refine the criteria for accepting a landscape. In particular, the determination of typical threshold values for the current measures requires more investigation.

The diameter and autocorrelation measures characterize the *search landscape*, which does not include the search process. It would be more interesting to work on the *process landscape*, since ultimately we are only interested in the search process efficiency. However, characterization of the process landscape is much more difficult, and still remains an open issue for testing problems. Up to now, our work on the process landscape has addressed less ambitious issues, using two alternative approaches.

The first one, followed in (Abdellatif-Kaddour et al., 2003b), consisted in modifying the core SA algorithm to tailor it for the problem under consideration. At the expense of *ad hoc* trials, we obtained a specific SA variant that worked well on the Boiler example. However, this variant is not expected to be reusable. Indeed, additional experiments in (Abdellatif-Kaddour, 2003c) showed that it poorly performs on the QAP and calendar problems.

The second approach, explored in this paper, was intended to be less specific. It consists in monitoring search convergence, so as to stop runs that are unlikely to succeed within the allowed number of iterations, and restart the search from another point. The proposed monitoring schemes supplied consistent results for the four case studies we experimented with. The schemes are not costly to implement, and may improve search performance while having a neutral effect in the worst case. We will conduct further experiments to confirm these results. In particular, the relevance of our schemes relies on the assumption that the GRBS measure evolves as shown in Figure 3, with possibly different phase durations. We will study whether GRBS evolution could exhibit different patterns for other instances of testing problems (e.g., exhibit peaks, …).

Finally, we hope that the first results presented in the paper will encourage more experimental studies based on the measurement of testing landscapes. As a first step, the proposed approach could provide a useful framework to revisit existing case studies.

Aknowledgements

The authors would like to thank the anonymous referees for their detailed and helpful comments on a previous version of this paper.

References

Abdellatif-Kaddour, O., Thevenod-Fosse, P., and Waeselynck, H. 2003a. Property-Oriented Testing: A Strategy for Exploring Dangerous Scenarios. *Proc. ACM Symposium on Applied Computing (SAC'2003)*, Melbourne, USA, 1128-1134.

Abdellatif-Kaddour, O., Thevenod-Fosse, P., and Waeselynck, H. 2003b. An Empirical Investigation of Simulated Annealing Applied to Property-Oriented Testing. *Proc. ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03)*, Tunis, Tunisia.

Abdellatif-Kaddour, O. 2003c. Property-Oriented Testing of Control Systems: Stepwise Construction of Test Scenarios Generated by Simulated Annealing Search. Doctoral Dissertation, Polytechnic National Institut of Toulouse, France, LAAS Report n° 03573. (In French).

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

Adamopoulos, K., Harman, M., and Hierons, R.M. 2004. How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. *Proc. Genetic and Evolutionary Computation Conference (GECCO 2004)*, LNCS 3103, Springer Verlag, 1338-1349.

Angel, E. and Zissimopoulos, V. 1998. Autocorrelation Coefficient for the Graph Bipartitioning Problem. *Theoretical Computer Science*, 191: 229-243.

Angel, E. and Zissimopoulos, V. 2000. On the Classification of NP-Complete Problems in terms of their Correlation Coefficient. *Discrete Applied Mathematics*, 99(1-3): 261-277.

Belaidouni, M. and Hao, J.K. 2000. Landscapes of the Maximal Constraint Satisfaction Problem. *Proc. 4th European Conference on Artificial Evolution (EA'99)*, LNCS 1829, Springer Verlag, 244-255.

Belaidouni, M. 2001. Metaheuristics and Search Landscapes. Doctoral Dissertation, University of Angers, France. (In French).

Belaidouni, M. and Hao, J.K. 2002. SAT, Local Search Dynamics and Density of States. *Proc. 5th European Conference on Artificial Evolution*, LNCS 2310, Springer Verlag, 192-204.

Clarke, J., Dolado, J.J., Harman, M., Hierons, R., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., and Shepperd, M. 2003. Reformulating Software Engineering as a Search Problem. *IEE Proceedings Software*. 150(3): 161-175.

Connolly, D.T. 1990. An Improved Annealing Scheme for the QAP. *European Journal of Operational research*. 46(1): 93-100.

Eremeev, A.V. and Reeves, C.R. 2003. On Confidence Intervals for the Number of Local Optima. *Proc. EvoWorkshops 2003*, LNCS 2611, Springer Verlag, 224-235.

Frank, J., Cheeseman, P. and Stutz, J. 1997. When Gravity Fails: Local Search Topology. *Journal of Artificial Intelligence Research*. 7: 249-281.

Gross, H.G., Jones, B. and Eyres, D.E. 2000. Structural Performance Measure of Evolutionary Testing Applied to Worst-Case Timing of Real-Time Systems. *IEE Proceedings Software*. 147(2): 161-175.

Harman, M. and Jones, B.F. 2001. Search-Based Software Engineering. *Information and Software Technology*, 43(14): 833-839.

Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hordijk, W. 1996. A measure of landscapes. *Evolutionary Computation*. 4(4): 335-360.

Jones, T. and Forrest, S. 1995. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. *Proc. Int. Conf. on Genetic Algorithms (ICGA'03)*, 184-192.

Jones, B.F., Sthamer, H-H. and Eyres, D.E. 1996. Automatic Structural Testing Using Genetic Algorithms. *Software Engineering Journal*. 11(5): 299-306.

Kirkpatrick, S., Gellat, C.D., and Vecchi, M.P. 1983. Optimization by Simulated Annealing. *Science*. 220(4598):671-680.

Lammermann F. and Wegener J. 2005. Test-Goal-Specific Termination Criteria for Evolutionary White-Box Testing by Means of Software Measures, *Proc. 6th Metaheuristics International Conference (MIC'2005).*

Lundy, M. and Mees, A.I. 1986. Convergence of an annealing algorithm. *Mathematical Programming*. 34(1): 111-124.

Maniezzo, V., Dorigo, M. and Colorni, A. 1995. Algodesk: an Experimental Comparison of Eight Evolutionary Heuristics Applied to the Quadratic Assignment Problem. *European Journal of Operational research.* 81(1): 188-204.

McMinn, P. 2004. Search-Based Software Test Data Generation: A Survey. *Software Testing, Verification & Reliability*. 14(2): 105-156.

Merz, P. and Freisleben, B. 2000. Fitness Lanscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. *IEEE Trans. on Evolutionary Computation.* 4(4): 337-352.

Michael, C.C., McGraw, G., Schatz, M.A. 2001. Generating Software Test Data by Evolution. *IEEE Trans. on Software Engineering*. 27(12): 1085-1110.

Nugent, C.E., Vollman, T.E. and Ruml, J. 1968. An Experimental Comparison of Techniques for the Assignment of Facilities to Locations. *Operations Research*, 16: 150-173.

O'Sullivan, M., Vössner, S. and Wegener, J. 1998. Testing Temporal Correctness of Real-Time Systems – a New Approach using Genetic Algorithms and Cluster Analysis. *Proc. 6th European Conference on Software Testing, Analysis & Review (EuroSTAR 1998).*

Pargas, R.P., Harrold, M-J. and Peck, R.R. 1999. Test Data Generation Using Genetic Algorithms. *Software Testing, Verification & Reliability*. 9(4): 263-282.

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

Rosé, H., Ebeling, W. and Asselmeyer, T. 1996. The density of States – a Measure of the Difficulty of Optimisation Problems. *Proc. Parallel Problem Solving from Nature (PPSN IV)*, LNCS 1141, Springer Verlag, 208-217.

Schultz, A.C., Grefenstette, J.J. and De Jong, K.A. 1995. Learning to Break Things: Adaptative Testing of Intelligent Controllers. *Handbook on Evolutionary Computation*, chapter G3.5. IOP Publishing Ltd. and Oxford University Press.

Stadler, P.F. and Schnabl, W. 1992. The Landscape of the Traveling Salesman Problem. *Physics Letters A*, 161(4): 337-344.

Taillard, E.D. 1991. Robust Tabu Search for the Quadratic Assignment Problem. *Parallel Computing*. 17(4&5): 443-455.

Tracey, N., Clark, J., Mander, K. and McDermid, J. 1998a. An Automated Framework for Structural Test-Data Generation. *Proc. 13th IEEE Conference on Automated Software Engineering (ASE*). Hawaii, USA, 285-288.

Tracey, N., Clark, J., and Mander, K. 1998b. Automated Program Flaw Finding Using Simulated Annealing. *Proc. ACM Int. Symp. on Software Testing and Analysis (ISSTA'98*). Clearwater Beach, Florida, USA, 73-81.

Tracey, N., Clark, J., Mander, K. and McDermid, J. 2000a. Automated Test-Data Generation for Exception Conditions. *Software – Practice and Experience*. 30(1): 61-79.

Tracey, N. 2000b. A Search-Based Automated Test Data Generation Framework for Safety-Critial Software. PhD Dissertation, University of York, UK.

Vanneschi, L., Tomassini, M., Collard, P. and Clergue, M. 2003. Fitness Distance Correlation in Structural Mutation Genetic Programming. *Proc. Europ. Conf. on Genetic Programming (EuroGP'03)*, LNCS 2610, Springer Verlag, 455-464.

Wegener, J., Sthamer, H.H., Jones, B.F. and Eyres, D.E. 1997. Testing Real-Time Systems Using Genetic Algorithms. *Software Quality Journal*. 6(2): 127-135.

Wegener, J., Buhr, K. and Pohlheim, H. 2002. Automatic Test Data Generation for Structural Testing of Embedded Software Systems by Evolutionary Testing. *Proc. Genetic and Evolutionary Computation Conference (GECCO-2002)*. New York, USA, 1233-1240.

Wegener, J. and Buehler, O. 2004. Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System. *Proc. Genetic and Evolutionary Computation Conference (GECCO-2004)*. Springer Verlag, LNCS 3103, 1400-1412.

Weinberger, E. 1990. Correlated and Uncorrelated Landscapes and How to Tell the Difference. *Biological Cybernetics*. 63: 325-336.

Yokoo, M. 1997. Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms: Analyzing Landscapes of CSPs. *Proc. Int. Conf. on Principles and Practice of Constraint Programming (CP'97)*. LNCS 1330, Springer Verlag, 356-370.

Appendix A – The Boiler Case Study

The steam boiler problem was proposed as a challenge for formal methods for safety-critical control systems (Abrial et al. 1996). The target system consists of a plant, a control program and a message transmission system connecting them. The plant has the following physical units: the steam-boiler, 4 pumps to provide water, 4 pump controllers indicating whether or not water is flooding from the pump, a sensor to measure the level of water in the boiler, a sensor to measure the quantity of steam which comes out, an operator desk. The program serves to control the level of water. Its functional requirement is to maintain the level between two nominal values N_1, N_2. Its safety requirement is to maintain the level between two safety limits M_1, M_2, where $M_1 < N_1$ and $M_2 > N_2$, otherwise the boiler explodes. The program has a cyclic behavior, and executes the following actions every five seconds:

- Reception of messages from the plant. There are 33 possible messages, including sensor data (e.g., LEVEL(v), where v measures the water level in the boiler) or messages from the operator desk (e.g., STOP, when an emergency shutdown is requested by the operator).
- Analysis of the information received from the plant. The control program has to update its view of the physical environment. As any device may become faulty at any time, the program cannot trust the received data. Plausibility and consistency checks are performed. The checks are based on the view elaborated at the previous cycle, as well as on expectations on the system dynamics. Depending on the check results, the program decides which devices are working correctly, and computes a new estimation range for the water level.
- Synthesis of messages and transmission to the plant. The program determines its operating mode, and sends it to the plant. The *emergency stop* mode is sent if the program considers that safety is endangered. The other modes are *initialization*, *normal*, *degraded* or *rescue* depending on the (perceived) states of the devices. Failure detection messages may also be sent to inform the operator of the faulty devices. Commands to open or close the pumps are sent to control the level. Overall, 27 messages are possible.

A more detailed description of the case study can be found in (Abrial et al. 1996). Many contributions were made, some of them proposing an implementation of the control program. A plant simulator was provided by the FZI to allow contributors to run and test their implementation.

Our test experiments involved the implementation proposed by J-R. Abrial and Steria. It is a C program of approximately 4800 lines of code (including comments) that can be interfaced with the FZI simulator. The test environment is shown in Figure A.1.

The tested system is composed of the control program and the plant simulator. Once started, the system autonomously operates. The closed-loop control is then perturbed by simulating the occurrence of faults in the devices, at chosen cycles of operation: our test inputs are timed fault commands for the simulator. For a time window of n cycles, and a maximum of 10 faults (affecting the 4 pumps, the 4 pump controllers, the water sensor and the steam sensor), the number of possible input sequences is:

$$\sum_{f=1}^{10} \binom{10}{f} n^{f} .$$

When an input sequence is supplied to the system, an execution trace is recorded. It contains the data needed to identify a boiler explosion or a dangerous situation. An explosion is easily identified because the simulator notifies it. Dangerous situations require more analysis. Whether the control program correctly diagnosed the faulty devices is determined from the failure detection messages sent to the plant. Whether the program correctly estimated the water level is determined by comparing its estimation range with the "actual" value in the simulator.

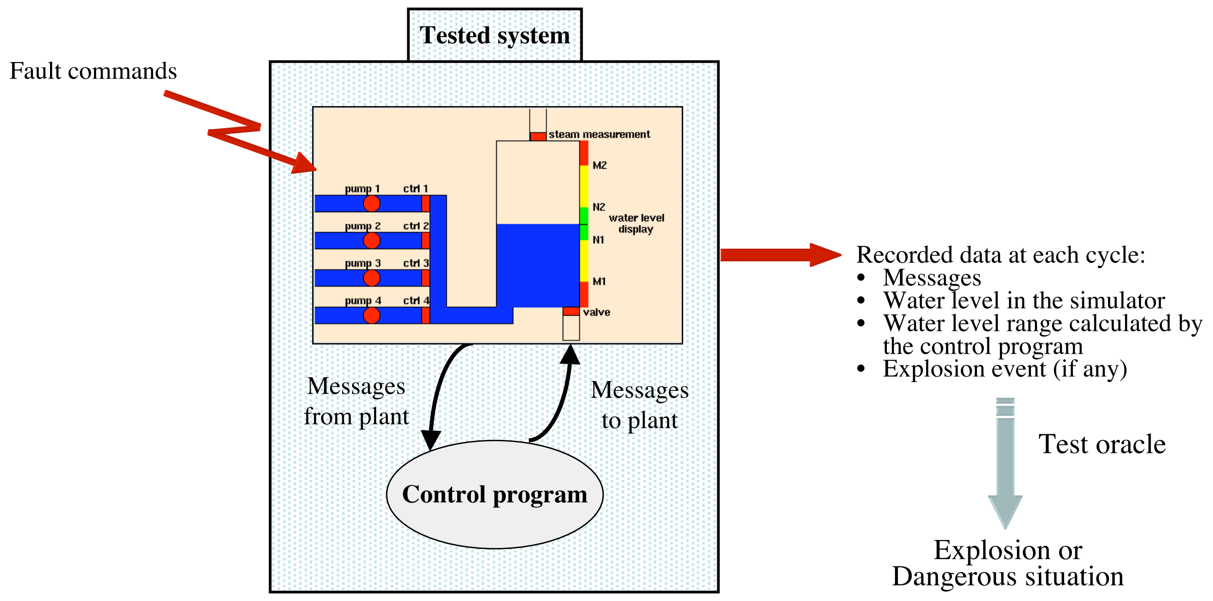


Figure A.1 – Test environment used in our experiments

Appendix B – Landscapes for the Boiler Problem

There are 2 landscapes, by combining 2 cost functions x 1 neighborhood operator.

B.1 Cost1

Test sequences fulfilling the test objective (i.e. yielding a boiling explosion or a dangerous situation) are assigned a zero cost. Other test sequences are assigned a positive cost, equals to the minimum of penalties associated with the three dangerous situations mentioned in Section 2.1. Intuitively, cost f_i is intended to penalize test sequences that are not "stressful" with respect to dangerous situation *i*.

- Cost f_1 is related to the control program not detecting the failure of a device. When none of the devices is faulty, this situation cannot occur: hence f_1 is assigned a maximum penalty K_1. Then, we let f_1 decrease as the number of faulty devices increases.
- Cost f_2 is related to the control program wrongly detecting the failure of a device that is non-faulty. The Boiler environment includes 10 devices (4 pumps, 4 pump controllers, 1 water level sensor, 1 steam sensor). When the test sequence makes all these devices faulty, wrong detection cannot occur: f_2 is assigned a maximum penalty K_2. Then, we let f_2 decrease as the number of non-faulty devices (10 – *nbFaultyDevices*) increases.
- Cost f_3 is related to bad estimation of the water level. At each cycle, the control program computes a lower bound qc_1 and upper bound qc_2 of the water level ($qc_1 < qc_2$). A dangerous situation is reached when the actual water level (observed from the boiler simulator) is outside the range $[qc_1, qc_2]$. Sequences not fulfilling this objective are considered all the more stressful as the uncertainty about the actual level is high, i.e. f_3 is decreasing with $(qc_2 - qc_1)$. In this way, the maximum penalty is when the estimation range is both correct (no dangerous situation) and accurate at the highest possible precision.

```
IF (boiler explosion OR dangerous situation observed) THEN
        Return (0)
ELSE
        Return (Min ($f_1, f_2, f_3$))
        Where:
                $f_1 = \frac{K_1}{nbFaultyDevices + 1}$
                $f_2 = \frac{K_2}{10 - nbFaultyDevices + 1}$
                $f_3 = \frac{K_3}{qc_2 - qc_1}$
ENDIF
```

The experiments reported in Section 2.3 involved the following values for constant parameters K_i: $K_1 = K_2 = 250$, $K_3 = 4000$. Calibration was performed empirically, with the aim of providing similar variation ranges for all f_i. This yielded [22, 250] for f_1 and f_2, [7, 160] for f_3.

B.2 Cost2

The positive costs returned by Cost2 are primarily based on the consideration of water level estimation. The first term corresponds to function f_3 defined for Cost1. It rewards test sequences that increase the control program's uncertainty about the water level. The second term is intended to discriminate between test sequences yielding the same uncertainty. It favors scenarios that make the actual level q get close to the safety limits M_1 and M_2 (the boiler explodes when the water level gets lower than M_1 or greater than M_2).

```
IF (boiler explosion OR dangerous situation observed) THEN
        Return (0)
ELSE
        Return $\left(\frac{K_4}{qc_2 - qc_1} + K_5.Min\left(|q - M_1|, |q - M_2|\right)\right)$
ENDIF
```

The experiments reported in Section 2.3 involved the following values for constant parameters K_i: K4 = 10,000 and K5 = 0.2. These values were intended to give higher weight to the first term than to the second one. The variation ranges are respectively [20, 400] and [0, 70].

B.3 Neighborhood Operator

The neighbors of a test sequence T are obtained from T by either: (i) changing the date of one fault of T; or (ii) adding one fault to T at an allowed date; or (iii) removing one fault from T. The allowed dates depend on the space explored by the test strategy. The experiments reported in Section 2.3 correspond to the space of sequences with prefix Steam_fault(3) and additional faults occurring at cycles [4, 8]. The neighborhood operator is applied only to these additional faults. For example, starting from Steam_fault(3), Pump2_fault(5), WaterLevel_fault(8) it may generate:

- Steam_fault(3), Pump2_fault(4), WaterLevel_fault(8) by changing the date of Pump2-Fault.
- Steam_fault(3), Pump1_fault(5), Pump2_fault(5), WaterLevel_fault(8) by adding a Pump1-Fault.
- Steam_fault(3), Pump2_fault(5) by removing the water level fault.

Overall, there are 8 + 35 + 2 = 45 neighbors of the original sequence.

Appendix C – Landscapes for the Cal 1 problem

There are 24 landscapes, corresponding to the combinations of 4 cost functions x 6 neighborhood operators.

C.1 Cost Functions

	Definition
Cost1	If (*day, month, year*) = (31, 12, 2000) then Cost1 = 0 Else Cost1 is randomly chosen in [1, 1000]
Cost2	Cost2 = $\|31-day\| + \|12-month\| + \|2000-year\| + K_{day} + K_{month} + K_{year}$
Cost3	Cost3 = $\|31-day\| + 31*\|12-month\| + 365*\|2000-year\|$
Cost4	Exact number of days between date (*day, month, year*) and date (31, 12, 2000)

Cost1 assigns a random cost value to any solution, except triplet (31, 12, 2000) which is assigned a zero cost.

Cost2 is inspired from the cost function used by (Tracey, 200b). It considers each triplet parameter, and measures its deviation from the target value in the optimal triplet. For example, $|31-day|$ measures deviation of the *day* parameter value, while $|2000-year|$ measures deviation of the *year* parameter value. Furthermore, as soon as deviation of parameter i is not zero, a constant penalty K_i is added, where *year* deviation is more penalized than *day* or *month* one:

- If $day = 31$ then $K_{day} = 0$ else $K_{day} = 10$
- If $month = 12$ then $K_{month} = 0$ else $K_{month} = 10$
- If $year = 2000$ then $K_{year} = 0$ else $K_{year} = 100$

The total cost is then the sum of individual costs for *day*, *month* and *year*. The variation range of Cost2 is [0, 1167].

Cost3 and *Cost4* make the cost value depend on the number of days between the current date and the optimal one. *Cost3* corresponds to a crude approximation of this number of days, while *Cost4* is exact. The variation ranges are respectively [0, 365371] and [0, 364999].

Note that the cost functions designed for Cal 2 are similar to the ones for Cal 1. They have just been adapted to account for the fact that Cal 2 has two optimal solutions. For example, *Cost4* computes the minimal number of days between the current date and any one of the two optimal dates.

C.2 Neighborhood Operators

	Definition	**|N| = maximal number of neighbors a date may be connected to**
N1	Any date obtained by letting one, two or three triplet parameters (*day, month, year*) vary ± 1, modulo the maximal parameter value.	26
N2	Same as N1, without modulo (e.g. a December date is no more connected to a January one).	26
N3	Any date no more than 15 days apart from the current date, modulo the solution space boundaries, that is, (1, 1, 1900) and (31, 12, 3000) are connected.	30
N4	Same as N3, without modulo.	30
N5	Any date no more than 400 days apart from the current date, modulo the solution space boundaries.	800
N6	Same as N5, without modulo.	800

Intuitively, *N1* and *N2* should be well-suited to cost function *Cost2*, while N3-N6 are more in the spirit of *Cost3* and *Cost4*.

Note that the neighborhood operators designed for Cal 2 are exactly the same as the ones for Cal 1.