A Self-Organized Criticality Mutation Operator for Dynamic Optimization Problems

Carlos M. Fernandes[1,2] cfernandes@laseeb.org
J.J. Merelo[2] jjmerelo@gmail.com
Vitorino Ramos[1] vitorino.ramos@alfa.ist.utl.pt
Agostinho C. Rosa[1] acrosa@laseeb.org

[1]LaSEEB, Technical Univ. of Lisbon
Av. Rovisco Pais, 1, TN 6.21, 1049-001,
Lisbon, PORTUGAL

[2]Departamento ATC, University of Granada
c/ periodista Daniel Saucedo, sn, 18071,
Granada, SPAIN

ABSTRACT

This paper investigates a new method for Genetic Algorithms' mutation rate control, based on the Sandpile Model: Sandpile Mutation. The Sandpile is a complex system operating at a critical state between chaos and order. This state is known as Self-Organized Criticality (SOC) and is characterized by displaying scale invariant behavior. In the precise case of the Sandpile Model, by randomly and continuously dropping "sand grains" on top of a two dimensional grid lattice, a power-law relationship between the frequency and size of sand "avalanches" is observed. Unlike previous off-line approaches, the Sandpile Mutation dynamics adapts during the run of the algorithm in a self-organized manner constrained by the fitness values progression. This way, the mutation intensity not only changes along the search process, but also depends on the convergence stage of the algorithm, thus increasing its adaptability to the problem context. The resulting system evolves a wide range of mutation rates during search, with large avalanches appearing occasionally. This particular behavior appears to be well suited for function optimization in dynamic environments, where large amounts of genetic novelty are regularly needed in order to track the moving extrema. Experimental results confirm these assumptions.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search.

General Terms

Algorithms, Experimentation.

Keywords

Genetic Algorithms, Dynamic Optimization Problems, Self-Organized Criticality.

1. INTRODUCTION

Evolutionary Algorithms (EAs) [3] are bio-inspired metaheuristics used to solve demanding computational problems hardly tackled by deterministic methods. Their simple design and adaptive characteristics makes them suitable for a wide range of applications. However, the convergence behavior and resulting performance is strongly dependent on a proper setting of parameter values, a procedure that may be found hard to deal with by non-expert users. The parameter values control techniques are usually divided into three categories [14]:

1. Deterministic: parameter values are changed by some deterministic rule.
2. Adaptive: values change during the EA run depending on its behavior
3. Self-Adaptive: EAs evolve the parameter values together with the problem solution.

Deterministic methods may be useful when developing an EA for a specific problem for which the algorithm behavior and requirements are known for the different search stages. However, these methods are not robust and usually do not maintain their performance when changing to different problems or even different problem instances. For these purposes, adaptive methods are more suitable since the variation depends indirectly on the problem and the search stage. Self-adaptive methods follow the same intuition that led to EAs, by letting the parameter values evolve in the same way solutions advance towards the optima by natural selection. Nevertheless, self-adaptation, by relying on evolution and natural selection, suffer from the handicap associated with the process: convergence to proper values may be too slow, and the same applies for re-adaptation if the search conditions change and the EA needs to self-adapt the parameters to different values. Besides, it enlarges the search space by including the parameters into it. *Dynamic Optimization Problems* (DOPs) create extra problems to EAs when concerning the variation of parameter values. Deterministic methods are only useful when it is possible to recognize the instant when the fitness function changes (in real-world problems the changes in the fitness functions are not always detectable). Adaptive control methods require variation strategies that may depend, for instance, on population genotypes, phenotypes or fitness. The application of self-adaptive methods is even more critical when solving DOPs since the slow evolution of the parameter values may not be compatible with fast changing extrema.

Another approach is possible when engaging in the issue of parameter control. Systems with *Self-Organized Criticality* (SOC) [4, 5] may be attached to EAs in order to control the parameter values, diversity or population size and possibly overcome the difficulties inherent to deterministic, adaptive and self-adaptive control methods. Previous works [22, 23, 30] suggest that the task is feasible and may improve traditional EA's performance, especially when dealing with DOPs. Following this idea, a modified version of the classical mutation operator using the Sandpile Model as an auxiliary tool to control the mutation of a Genetic Algorithm (GA) is presented.

GECCO'08, July 12-16, 2008, Atlanta, Georgia, USA.

The present work is organized as follows. Section 2 provides a State of the Art on EAs, DOPs and SOC. Section 3 introduces the Sandpile Mutation. Section 4 describes the test environment and discusses the results and Section 5 concludes the paper and outlines future research.

2. STATE OF THE ART

Optimization problems are said to be dynamic when there is a change in the fitness function, problem instance or restrictions, thus making the optimum change as well. When changes occur, the solutions already found may be no longer valid and the process must engage in a new search effort. EAs adaptive characteristics make them good candidate tools to solve DOPs. However, these algorithms typically converge to an optimum and thereby lose the diversity necessary to adapt to a change in the environment when such a change occurs. In EAs research on DOPs, a number of authors have addressed the problem of convergence and subsequent loss of adaptability in many different ways, but most of these approaches may be grouped into one of the following three categories [10].

React on Changes. The EA is run in standard fashion, but as soon as a change in the environment has been detected, explicit actions are taken to increase diversity and thus facilitate the shift to the new optimum. Techniques such as *Hypermutation* [12] pursue the first category, keeping the whole population after a change but increasing population diversity by drastically increasing the mutation rate for some number of generations.

Maintaining diversity. Convergence is avoided all the time and it is hoped that a spread-out population can adapt to changes more easily. The *Random Immigrants Genetic Algorithm* (RIGA) [18] is one well-known example of a strategy that falls in the second category. Other examples of diversity oriented EAs for DOPs may be found in [11], [21] and [27].

Memory schemes. The EA is supplied with a memory to recall useful information from past generations. Memory may be provided in two general ways: *implicitly* [17] by using redundant representations or *explicitly* by introducing an extra memory and formulating strategies to deposit and retrieve solutions later [9].

Estimation of Distribution Algorithms (EDAs) [24, 28] is a class of EAs that replaces the standard crossover and mutation operators by building a probabilistic model of promising solutions and sampling from the corresponding probability distribution. Although dynamic problems have been a subject of EAs research for the last two decades, only recently this issue has started to raise a strong interest on EDAs' researchers. The *Population Based Incremental Learning* [6] was used in [32] to solve DOPs created by a problem generator proposed by the same authors. In [31], Yang proposes the *Univariate Marginal Distribution Algorithm* [26] with enhanced memory. Abass et al. introduced [1] the *Extended Compact Genetic Algorithm* [20] to solve dynamic problems. Their approach is based on random restarts of the population at each change (*React on Changes*).

Some recent proposals have been made using a *Swarm Intelligence* [8] approach to attempt to solve dynamic problems. Examples of Swarm Intelligence applied to dynamic environments may be found in [19] and [29], amongst many others. In [15], Fernandes et al. developed the *Binary Ant Algorithm* (BAA), based on the ACO [13] framework, to take advantage of ACO's ability to solve combinatorial DOPs and generalize it to binary problems. However, this method may be also regarded as a type of EDA, because BAA creates the possible solutions to a problem via transition probability vectors.

In 1987, Bak at al. [4] identified the SOC phenomenon associated with dynamical systems. The first system were SOC was observed was named after its inspiration as the Sandpile model, and consists of a cellular automata where at each cell of the lattice, there is a value which corresponds to the slope of the pile. Grains of sand are randomly "thrown" into the lattice where they pile up and increment the values of the cells. When a value exceeds a specific threshold, an avalanche takes place and four grains belonging to that cell are distributed by the neighboring sites (*von Neumann* neighborhood). If one of those sites also exceeds the threshold value, the avalanche continues, and the grains are also sent to the adjacent cells. The procedure of the Sandpile model is shown in figure 1. With these settings, and depending on the state of the lattice and the position of the new grain, a grain may cause rather different responses. It may not cause any change in the system if it falls in a cell with its value bellow threshold (other than increasing the sand on the cell, of course) and it may generate large avalanches of sand that will strongly redefine the shape of the pile. The likelihood of an avalanche is in power-law proportion to the size of the avalanche, and avalanches are seen to occur at all size scales. Large avalanches are very rare while small ones appear very often. Without any fine-tuning of parameters, the system evolves to a non-equilibrium critical state: SOC.

The *Bak-Sneppen* system [5] is a model of co-evolution between interacting species where SOC is also present. The model consists of a number of species, each one with a fitness value assigned and each connected to two other species (neighbors). Every time step, the specie with the worst fitness and both its neighbors are eliminated from the system and replaced by individuals with random fitness. Such an event is recorded as an avalanche of size 1, and if the next extinction involves one of the newly created species, then the size is incremented. By plotting the size of the extinctions over their frequency, a power-law relationship emerges, like in the Sandpile model.

SOC has already been applied in bio-inspired computation in the past [7, 22, 23, 30]. Since SOC associated with EAs may periodically insert large amounts of new material in the population or completely reorganize a solution to a problem, it soon was adopted by EA researchers in order to provide new means to control parameter values or maintain population diversity, thus avoiding premature convergence to local optima. *Extremal Optimization* (EO) [7] is based on the Bak-Sneppen model of SOC. Unlike EAs and Swarm Intelligence algorithms, it does not work with a population of individuals, but instead it evolves a single solution to the problem by local search and modification. EO removes the worst components of the solution and replaces them with randomly generated material. By plotting the fitness of the solution, it is possible to observe distinct stages of evolution, where improvement is disturbed by brief periods of dramatic decrease in the quality of the solution. In [22] the authors propose a mass extinction model and a mutation model for EAs (later extended to cellular GAs [23]) based on the Sandpile. The Sandpile is previously computed in order to obtain power-law values. Those values are then used during the run to

Procedure Sandpile

Considerer a lattice (x,y) and a function z(x,y) which represents the number o grains in the cells.
Starting with a flat surface $z(x,y) = 0$ for all x and y:
Add a grain of sand: $z(x,y) \rightarrow z(x,y) + 1$
if $z(x,y) > z_c$ then an avalanche occurs

$$z(x,y) \rightarrow z(x,y) - 4$$
$$z(x \pm 1, y) \rightarrow z(x \pm 1, y) + 1$$
$$z(x, y \pm 1) \rightarrow z(x, y \pm 1) + 1$$

if $z(x, y\pm1) = 4$ or $z(x\pm1, y) = 4$
Update z recursively

Figure 1. 2D Bak-Tang-Wisenfeld Sandpile Model

control the number of individuals that will be replaced by randomly generated solutions (SOC mass extinction model) or the mutation rate of the EA (SOC mutation model). In [30] a RIGA associated with the Bak-Sneppen model is presented and tested on DOPs: the *Self-Organized Random Immigrants Genetic Algorithm* (SORIGA). In each generation, SORIGA replaces the worst individual of the population and its neighbors by r_r random solutions (neighborhood relations are determined by the individuals' indexes in the population). To avoid new individuals to be quickly replaced by the fittest chromosomes in the population, the random solutions are stored in a subpopulation and the individuals from the main population are not allowed to replace the new individuals. Furthermore, selection and crossover are allowed only between individuals that belong to the subpopulation. By plotting the extent of extinction events, the authors argue that the model exhibits SOC behavior.

This paper's proposal differs from previous approaches in many ways. Unlike Extremal Optimization, it works on a population-based EA: the SOC mutation is attached to a GA. In addition, power-law values are not previously computed and then used to vary mutation rate, like in [22, 23]. This feature, as it will be shown in Section 4, is very important on DOPs, since large avalanches appear to be connected to changes in the environment: the distribution avalanche sizes varies with DOP characteristics (speed and severity of change); these self-adaptive features of the system would not be achievable if the power-law values were computed off-line. Finally, SORIGA gives new genetic material to the population by inserting r_r new chromosomes in each generation, while the Sandpile Mutation may completely reconfigure the population's alleles in only one generation. In addition, SORIGA always performs $N+r_r$ function evaluations in each time step. When compared with a generational GA with N individuals, SORIGA needs extra computational effort to perform the same number of generations.

3. THE SANDPILE MUTATION

The Sandpile Mutation, uses the original 2D Sandpile model with minor modifications in order to evolve self-regulated mutation rates. The pseudo-code is shown in figure 2 (maximization is assumed). The Sandpile Mutation replaces the traditional mutation in the Standard GA (SGA). However, while traditional mutation is done at bit level, that is, mutation is attempted in every bit and it is performed every time a gene passes the probability test, Sandpile Mutation procedure is performed every generation in the way described in figure 2. The procedure works as follows.

Sandpile Mutation

The Sandpile is evolved on a $n \times l$ lattice with $n = 1, 2, \ldots N$ and $l = 1, 2, \ldots L$. The lattice is randomly initialized with values between 1 and 3. This way, a first stage of search with rare avalanche events is avoided. The critical threshold z_c is set to 4 (equal to the number of grains that topple when a avalanche occurs). In the occurrence of an avalanche, the 4 grains drop into the cell's *von Neumann* neighborhood. If z reaches threshold z_c, but the mutation does not occur, then the grain is discarded.

for g grains **do**
  drop grain at random $z(n,l) \rightarrow z(n,l) + 1$
  compute normalized fitness:

$$f_n = \frac{fitness(i) - worstFitness}{bestFitness - worstFitness} \quad (1)$$

where i is the index of the chromosome associated with the cell (n, l), *worstFitness* is the lowest fitness in the current population and *bestFitness* is the best fitness in the population. Note: if *bestFitness* = *worstFitness*, f_n is set to 0.5

  if *randomValue*(0, 1.0) > f_n and cell (n, l) not active
    mutate (bit flip mutation**)**
    avalanche

$$z(n,l) \rightarrow z(n,l) - 4$$
$$z(n \pm 1, l) \rightarrow z(n \pm 1, l) + 1$$
$$z(n, l \pm 1) \rightarrow z(n, l \pm 1) + 1$$

    and **update** lattice z recursively

Figure 2. The Sandpile Mutation pseudo-code.

After a new population is generated by selection and crossover, the individuals are ranked according to its fitness. Then each individual is mapped into a $n \times l$ lattice with $n = 1,\ldots N$ and $l = 1,\ldots L$, where N is the population size and L is the chromosome length − see figure 3 for a detail of the Sandpile. The Sandpile starts evolving in the first generation (a specific initialization of the lattice is done in order to avoid a preliminary stage without events). In each generation g grains are randomly "thrown" into the lattice thus incrementing the cells values (z). When a cell reaches the critical value (z_c) an avalanche occurs only if a randomly generated value (between 0 and 1.0) is higher than the normalized fitness of the individual on which the avalanche is located. This way, fitter individuals have lower chances of being mutated while the genes of poor solutions have a high probability of being mutated when a z value reaches z_c (and please remember that ranking is done every generation, supplying the lattice with a sort of "slope"). After a first avalanche, the neighboring cells are updated in a recursively manner and the avalanche may proceed through the lattice as long as the state of the Sandpile and the fitness of the solutions are favorable to that progression. Threshold z_c is set to 4, equal to the number of grains that topple to the *von Neumann* neighborhood when an avalanche occurs.

The Sandpile Mutation has one restriction that is not present in original model: if a cell is already involved in an avalanche (*active cell*), and the recursive nature of the process has not allowed it to complete its sequence, then the cell is ignored. This restriction eliminates hypothetical avalanche cycles and several mutations of the same gene. Although there is no empirical evidence that this restriction improves the performance of the model, it is adopted in order to reduce the computational time of the mutation process. (With very complex fitness functions, the cost of a Sandpile Mutation with cycles may become irrelevant.)

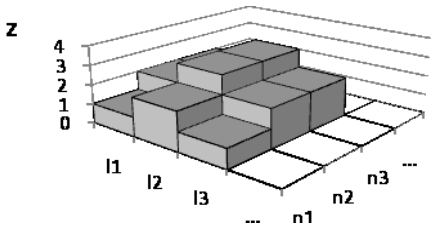

Figure 3. A section of a sandpile attached to a population: genes l_1, l_2, l_3 from chromosomes n_1, n_2, n_3. Please note that cell (l_2, n_2) is close to $z_c = 4$. If one grain falls in that site, an avalanche will take place (if fitness test is passed). Then, four grains will topple to the cell's *von Neumann* neighborhood and (l_2, n_3) will reach threshold. If conditions are favorable, avalanche and mutations may proceed indefinably.

Two more details must be referred. First, if a z value reaches z_c but the mutation does not occur (due to the fitness test) then the grain is discarded. In addition, if an avalanche takes place in a cell on the edge of the lattice, then the grains will "fall off" the lattice.

With this process, different purposes are achieved. By using a Sandpile associated with the population, avalanches of different sizes are generated, thus changing the population in diverse manners, from minor changes to radical reconfigurations of the genetic material. Associating the avalanche and consequent mutations with the quality of solutions, better individuals are favored, preventing them from being subject to extreme mutations, this way introducing an element of natural selection in the system. In the following experiments, the Sandpile Mutation replaces the traditional mutation of an SGA. The resulting algorithm is the *Sandpile Mutation Genetic Algorithm* (SMGA). Its characteristics may place it on the *maintaining diversity* category [10] described in section 2.

4. EXPERIMENTS AND RESULTS

The test environment proposed in [32] was used to investigate the Sandpile Mutation on DOPs. Given a stationary problem $f(\boldsymbol{x})$ $(\boldsymbol{x} \in \{0,1\}^l)$ where l is the chromosome length, the dynamic environments may be constructed by applying a binary mask $\mathbf{M} \in \{0,1\}^l$ to each solution before its evaluation in the following manner:

$$f(\boldsymbol{x}, t) = f\left(\boldsymbol{x} \text{ XOR } \mathbf{M}(k)\right) \tag{2}$$

Where t is the generation index, $k = t/\tau$ is the period index and $f(\boldsymbol{x}, t)$ is the fitness of solution $\boldsymbol{x}$. $\mathbf{M}(k)$ can be incrementally generated as follows:

$$\mathbf{M}(k) = \mathbf{M}(k-1) \text{ XOR } \mathbf{T}(k) \tag{3}$$

where $\mathbf{T}(k)$ is an intermediate binary mask for every period k. This mask $\mathbf{T}(k)$ has $\rho \times l$ ones, where ρ is a value between 0 and 1.0 which controls the intensity or severity of change. Notice that $\rho = 0$ corresponds to a stationary problem since T vectors will carry only 0's and no change will occur in the environment. On the other hand, $\rho = 1$ guarantees the highest degree of change. Therefore, by changing ρ and τ in the previous set of equations it is possible to control two of the most important features when testing algorithms on DOPs: severity (ρ) and speed (τ) of change [2]. Following the experiments in [30], a *Royal Road* function and two *Deceptive* functions were used as stationary problems where the dynamic frame was applied.

Royal Road functions [25] were specifically designed to study GA's performance on the level of building block interactions, and are widely used for GAs test and analysis. From the set of *Royal Road* functions, R1 was selected. R1 is defined by:

$$f(\boldsymbol{x}) = \sum_{i=1}^{q} c_s \delta_s(\boldsymbol{x}) \tag{4}$$

where q is the number of schemata $S = \{s_1, \ldots, s_q\}$ and, for the function in this paper, $\delta_s(x)$ is set as 1 if $\boldsymbol{x}$ is an instance of S and 0 otherwise, and $c_s = 8$ for all s; a 64-bit string was used and each schema is composed of 8 contiguous bits.

Deceptive functions are designed to be unfriendly for optimization via GAs. Instead of combining low order building blocks in order to form higher order building blocks, deceptive functions conduce the search towards deceptive attractors. To build the deceptive functions a trap function can be used:

$$f(\boldsymbol{x}) = \begin{cases} \frac{a}{z}(z - u(\boldsymbol{x})), & \text{if } u(\boldsymbol{x}) \leq z \\ \frac{b}{l-z}(u(\boldsymbol{x}) - z), & \text{otherwise} \end{cases} \tag{5}$$

where u(x) is the unitation function, l is length of x, a is local optimum, b is the global optimum and z is the slope-change location which separates the attraction basin sizes of the two optima. Following [30], two deceptive functions were defined by setting l, a, b and z values as. *Deceptive 1*: $l = 10$, $a = 0.82$ and $z = 8$; *Deceptive 2:* $l = 50$, $a = 0.8$, $z = 48$; b is set to 1.0 in both functions. One characteristic of these functions must be noted: the Hamming distance between global and local optimum is equal to the chromosome length, that is, to change between the two optima is sufficient to flip all bits. This attribute, as explained latter, may benefit the Sandpile Mutation performance and induce wrong conclusions about the behavior of the model.

For that reason, the *Massively Multimodal Deceptive Problem* (MMDP) [16] was added to the test set. MMDP is function composed of k deceptive subproblems (s_i) of 6 bits each whose values depend on the unitation of the bit string as shown in table 1. The fitness of the binary string is then computed in the following way:

$$f_{MMDP}(\boldsymbol{x}) = \sum_{i=1}^{K} f_s(\boldsymbol{x}) \tag{6}$$

Please note that MMDP has two global optima and unlike the deceptive functions previously described, the Hamming distance between the global and local optimum is lower than the chromosome length and to change between global and local optima is not enough to flip all bits.

Before proceeding to an analysis of SMGA performance on dynamic environments it is important to observe how the algorithm behaves on a static and scalable problem. For that purpose, MMDP function was used. Although there are other trap

Table 1. MMDP function.

$u(\mathbf{x})$	0	1	2	3	4	5	6
$f_s(\mathbf{x})$	1.0	0	0.360384	0.640576	0.360384	0	1.0

and deceptive functions which have been a subject of more exhaustive and widespread studies amongst the community of EA researchers, those functions may lead to wrong conclusions about Sandpile Mutation performance. Consider for instance a deceptive function as the one described above. The function's global optimum consists of a 1's string while the local optimum is the 0's string. If a population converges to a local optimum, a massive avalanche of Sandpile Mutation may invert all the bits of an individual, thus immediately acquiring the global optimum. That is, SMGA may take advantage of the deceptive function's structure. MMDP does not hold the same structure and a large avalanche over a population that has converged to local optima is not likely to create a global optimum string.

As already stated, the Sandpile model is a system with SOC and exhibits a power-law relationship between the frequency and size of the avalanches. Although SMGA is based on the Sandpile model, modifications that arise from the attachment of the Sandpile to a GA do not guarantee that the resulting system also exhibits SOC. However, the general idea behind Sandpile Mutation is to take advantage of large avalanches that reconfigure the system from time to time. Large avalanches may lead to large mutation rates and in between those catastrophic events, lower rates give the algorithm a more steady behavior. The non-periodic manner in which avalanches of any size emerge in the system is also explored by SMGA because in dynamic problems the landscapes may change in many ways. To check if SMGA exhibits the expected behavior, it is necessary to observe the relationship between the frequency and size of the avalanches (and mutations). Figure 4 shows that relationship for MMDP with different length L. Although it is not the aim of this paper to prove that SMGA is a SOC system, it is evident that the Sandpile evolves (at least) near the desired state: small avalanches are frequent and large avalanches are scarce – see graphs at the left hand side of figure 4. The way this distribution affects the mutation rate may be seen on the right hand side of figure 4. Although the shape of the curves differs from the log-log graphs of the avalanches, they show SMGA's general tendency to evolve small mutation rates in many generations and large rates on few generations. The algorithm was executed with p_c = 0.7, 2-point crossover, binary tournament selection and $g = 10 \times L$.

The test environment generator described above was used to determine how effective SMGA is when dealing with DOPs. DOP versions of the *Royal Road* and *Deceptive* functions were implemented and tested. SMGA's performance is compared with an SGA and two RIGAs – see table 2. As stated above the *Deceptive* functions described above may favor SMGA due to its specific search space. By modifying the Sandpile Mutation with the purpose of avoiding the convergence to the global optimum via catastrophically mutation of local optimum chromosomes, it is then possible to compare the algorithms with fairness. The modification was done in the mutation step of the Sandpile procedure – see figure 2. Instead of flipping the bit every time the conditions are satisfied, the modified SMGA (SMGA*) flips the bit with probability 0.5. To compensate the expected decrease in the resulting mutation rates, SMGA* parameter g value was set to $g^* = 5 \times g$.

For each DOP several degrees of severity (τ) and speed (ρ) were set: $\tau = 10$, $\tau = 200$ and $\tau = 1000$; $\rho = 0.05$, $\rho = 0.6$ *and* $\rho = 0.95$

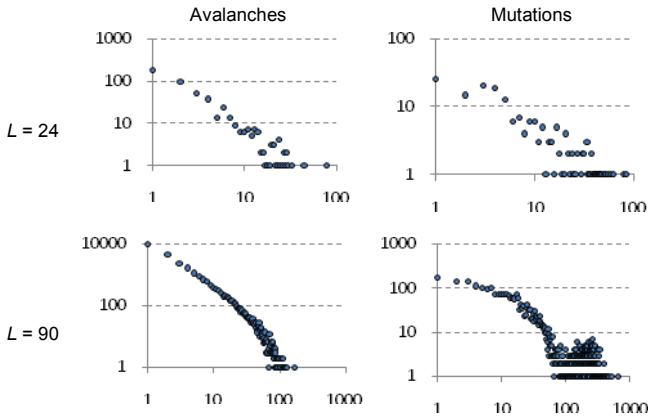

Figure 4. Log-log graphs of size and frequency distribution of SMGA's avalanches and mutations. *L* is chromosome length.

(if $\rho \times L$ is not integer, the value is rounded). Each algorithm was executed for 10 periods of environmental changes and, for each configuration, 30 runs were performed. The fitness was measured and averaged over the 30 runs. Evaluation of GAs' performance is done by comparing the mean best-of-generation values (this is the standard procedure for DOPs). Results and parameters are shown in Table 2. A statistical comparison was carried out by *t-tests* with 58 degrees of freedom at a 0.05 level of significance. If SGA or RIGA are significantly worse than SMGA, then the sign + is shown in parenthesis; else, the sign − appears; if the GAs are statistically equivalent, then the ~ symbol is used. A general analysis of the results reveals that there are only a small number of combinations of τ and ρ for which any of the GAs (SGA, RIGA 1 or RIGA 2) is significantly better than SMGA. (Please note that RIGAs perform $N+r_r$ function evaluations in each generation.) In addition, the results show that, in general, SMGA's efficiency increases with τ and ρ, especially in *Deceptive* functions 1 and 2. This behavior may be explained by SMGA's dynamic and non-deterministic characteristics. Sandpile Mutation main feature (and possibly its major force) consists on its capacity to reconfigure the population in a non-deterministic manner. Large mutations appear from time to time, but it is impossible to determine or set the exact moment when those avalanches take place. Fast changes in the environment mean that the optimum is moving without giving any chance for SMGA to evolve large mutation rates that might release the population from previous optimal regions of the search space. From table 2 it is clear that DOPs with $\tau = 10$ do not require such a dynamic variation of the mutation rate as the one provided by SMGA. On the other hand, with slower rates of change ($\tau > 10$), SMGA's overcomes the difficulties faced by SGA and RIGA and generally outperforms the other GAs. The observation of SMGA's behavior in *Royal Road* with different ρ also reveals a pattern: SMGA tends to perform better (that is, to outperform SGA and RIGA) with higher ρ. Low ρ means that the changes are not severe. SMGA ability to generate large amounts of genetic novelty, providing the algorithm with means to escape previous optimal regions, is not so useful when the fitness function changes only by a small amount. However, when ρ increases, SMGA's exposes it skills to solve DOPs with severe changes.

Table 2. Mean best-of-generation and standard deviation. General parameters: $N = 120$, 2-point crossover, tournament selection ($k_{ts} = 0.9$). SGA: $p_m = 1/L$, $p_c = 0.7$; RIGA 1: $p_m = 1/L$, $p_c = 0.7$, $r_r = 3$; RIGA 2: $p_m = 1/L$, $p_c = 0.7$, $r_r = 12$; SMGA: $p_c = 0.7$; $g = 10 \times L$ (Royal Road) and $g^* = 50 \times L$ (Deceptive 1 and 2). Royal Road: $L = 64$; Deceptive 1: $L = 10$; Deceptive 2: $L = 50$

		Royal Road				Deceptive 1				Deceptive 2			
τ	ρ	SGA	RIGA 1	RIGA 2	SMGA	SGA	RIGA 1	RIGA 2	SMGA*	SGA	RIGA1	RIGA2	SMGA*
10	0.05	32.99 ±4.29 (~)	31.20 ±3.29 (~)	31.76 ±3.89 (~)	**31.41 ±3.55**	0.837 ±0.007 (+)	0.831 ±0.010 (+)	0.829 ±0.081 (+)	**0.855 ±0.043**	0.753 ±0.004 (~)	0.665 ±0.009 (+)	0.667 ±0.007 (+)	**0.752 ±0.007**
10	0.60	11.2 ±1.32 (+)	10.69 ±1.34 (+)	11.06 ±1.07 (+)	**13.40 ±1.01**	0.811 ±0.023 (~)	0.802 ±0.028 (~)	0.802 ±0.026 (~)	**0.793 ±0.020**	0.562 ±0.006 (+)	0.588 ±0.004 (+)	0.593 ±0.005 (~)	**0.594 ±0.006**
10	0.95	17.16 ±1.96 (~)	16.62 ±1.96 (~)	16.35 ±1.79 (~)	**17.12 ±1.48**	0.979 ±0.020 (−)	0.960 ±0.021 (−)	0.956 ±0.021 (−)	**0,922 ±0.031**	0.504 ±0.004 (+)	0.559 ±0.004 (~)	0.562 ±0.005 (−)	**0.558 ±0.005**
200	0.05	60.03 ±1.46 (−)	59.26 ±1.42 (−)	59.67 ±2.05 (−)	**57.80 ±2.38**	0.823 ±0.008 (+)	0.824 ±0.010 (+)	0.824 ±0.010 (+)	**0.957 ±0.026**	0.7980 ±0.0002 (−)	0.7394 ±0.006 (+)	0.7413 ±0.005 (+)	**0.7973 ±0.0005**
200	0.60	37.87 ±1.41 (+)	37.26 ±1.27 (+)	37.70 ±1.50 (+)	**42.15 ±1.45**	0.842 ±0.018 (+)	0.843 ±0.020 (+)	0.840 ±0.021 (+)	**0.908 ±0.025**	0.7751 ±0.0006 (+)	0.7155 ±0.004 (+)	0.7185 ±0.004 (+)	**0.7832 ±0.0007**
200	0.95	31.87 ±1.23 (+)	30.96 ±1.17 (+)	31.82 ±1.25 (+)	**46.43 ±2.15**	0.976 ±0.012 (−)	0.971 ±0.016 (−)	0.964 ±0.017 (−)	**0.939 ±0.016**	0.7579 ±0.0008 (+)	0.7145 ±0.004 (+)	0.7160 ±0.003 (+)	**0.7808 ±0.0009**
1000	0.05	62, 87 ±0.78 (~)	62.80 ±0.82 (~)	62.90 ±0.99 (~)	**62.36 ±1.14**	0.823 ±0.008 (+)	0.826 ±0.012 (+)	0.821 ±0.051 (+)	**0.994 ±0.003**	0.79960 ±0.00005 (−)	0.76793 ±0.00216 (+)	0.77155 ±0.00324 (+)	**0.79947 ±0.00007**
1000	0.60	54. 20 ±1.35 (~)	54.38 ±1.17 (~)	54.38 ±1.45 (~)	**54.85 ±1.40**	0.854 ±0.021 (+)	0.846 ±0.017 (+)	0.850 ±0.021 (+)	**0.984 ±0.006**	0.79506 ±0.00012 (+)	0.76181 ±0.00334 (+)	0.76390 ±0.00256 (+)	**0.79670 ±0.00014**
1000	0.95	51. 85 ±1.21 (+)	51.81 ±1.10 (+)	51.59 ±1.03 (+)	**56.62 ±1.63**	0.973 ±0.014 (+)	0.978 ±0.014 (+)	0.966 ±0.017 (+)	**0.988 ±0.005**	0.79646 ±0.00918 (~)	0.76207 ±0.00281 (+)	0.76406 ±0.00286 (+)	**0.79623 ±0.00016**

Figure 5 shows SGA and SMGA's best fitness during the run on a dynamic *Royal Road* with $\tau = 200$. The graphics illustrate previous statements. When $\rho = 0.05$, SGA is slightly faster at tracking the moving optimum. SMGA's mutation rate distribution is not so useful in these conditions, because the landscape changes in small amounts. When ρ increases, SMGA's discloses its abilities to solve DOPs. With $\rho = 0.95$ (meaning that the landscape is suffering dramatic changes every 200 generations), SMGA replicates, in each period, the *best-of-generation* curve of the first environment (from $t = 0$ to $t = 200$). SGA looses diversity during the first stage of convergence and is unable to reach the same fitness values it has achieved at $t = 200$. SMGA's mutation rate in each generation when solving the *Royal Road* with $\tau = 10$ and several ρ values is shown in figure 6. Please note how the shape of the curves change when increasing ρ from 0 (meaning that no changes occurs in the fitness landscape, that is, the problem is static) to 0.95. With $\rho = 0.05$ there are already some hints of a relation between the beginning of a new period and the mutation bursts. With $\rho = 0.95$ a rough periodicity starts to emerge. The algorithm self-adapts to the environment changes. Computing the Sandpile Model off-line in order to use the power-law values to control mutation rate would not provide the algorithm with these self-adjusting capabilities. One may argue that increasing the mutation rate when a change occurs (like in Hypermutation [12]) may have the same effect. However, that would mean that changes were detectable, which is not always possible. A more general framework is considered here, were dynamic optimization is assumed to be "blind": changes in the landscape are not detectable (or the detection is too costly).

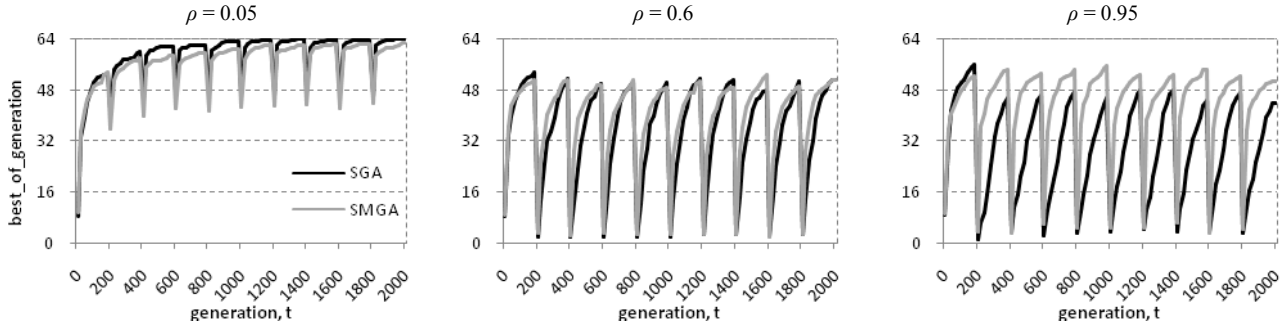

Figure 5. Comparing SGA and SMGA's dynamic behavior on Royal Road. $\tau = 200$

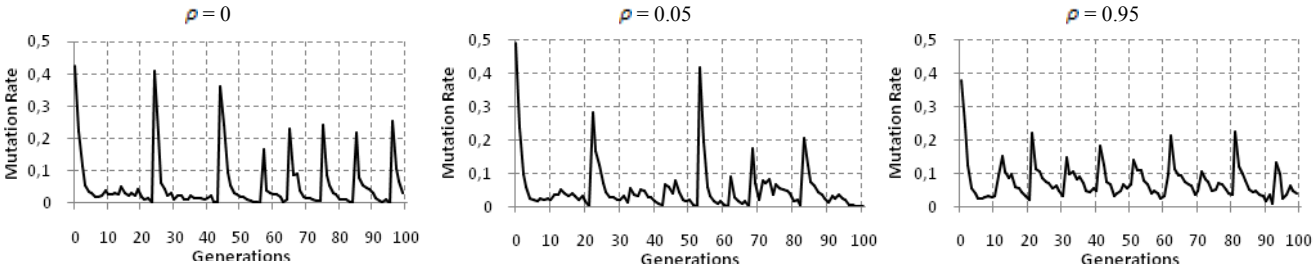

Figure 6. Royal Road. SMGA mutation rate variation with τ = 10. Mutation rate is measured every generation by comparing all the alleles in the population before and after *g* grains are thrown into the lattice.

Table 3 shows Sandpile Mutation results compared to those attained by SORIGA in [30]. SMGA outperforms SORIGA in eight of the nine scenarios in which *Deceptive 1* was tested. On *Royal Road* and *Deceptive 2*, SMGA tends to outperform SORIGA when the speed of change is high, while SORIGA behaves better when speed is lower. (There is not enough information in [30] to perform statistical analysis). Please note that, like traditional Random Immigrants GAs, SORIGA performs $N+r_r$ function evaluations in each generation, meaning that setting $r_r = 24$ (like in some experiments described in [30]) causes a significant increase of computational effort. In addition, r_r value affects the performance, which means that SORIGA's parameter space is more complex and more difficult to fine-tune. Sandpile Mutation needs a new parameter *g*, but that replaces the mutation rate (p_m) of traditional GAs. Since SORIGA is the closest approach to Sandpile Mutation, further experiments are obligatory in order to investigate how both algorithms behave on dynamic environments, and if the observed SMGA's mutation bursts when fitness changes may help to track the optimum of hard dynamic problems.

5. CONCLUSIONS AND FUTURE WORK

This paper presents a new method to control Genetic Algorithms' mutation rate in order to tackle Dynamic Optimization Problems. The new strategy is based on the Sandpile model and Self-Organized Criticality and it is named Sandpile Mutation. Genetic Algorithms' populations are attached to a Sandpile with the objective of generating a wide range of mutation rates − from small mutations to catastrophic events that reconfigure large amounts of the population. The Sandpile Mutation Genetic Algorithm has been tested and compared with other algorithms on three dynamic problems with different speed and severity settings. Results showed that the proposed algorithm outperforms *Standard Genetic Algorithm* and *Random Immigrants Genetic Algorithm* in the majority of the tests. Due to its characteristics, Sandpile mutation is expected to work better when the environment experiences medium or severe changes, and the results confirmed this assumption. Sandpile Mutation was also compared to another Self-Organized Criticality GA: the *Self-Organized Criticality Random Immigrants GA*, but further experiments are needed in order to properly evaluate the performance of the two algorithms. Results also showed that the Sandpile model self-regulates the mutation rate, and gives rise to mutation bursts when the environment changes. This particular behavior may be useful when tackling problems that need large amounts of genetic novelty to deal with the fitness changes, but that do not permit Hypermutation strategies, because changes are not be detectable, or are too costly to detect. Unlike some memory schemes and other methods that react to changes, the Sandpile Mutation does not need an explicit mechanism to respond to a shift in the optimum and introduce genetic novelty. The increase in the mutation rate is done in a self-regulated manner, without information from the environment itself, other than the fitness of the individuals.

Table 3. Comparison of SMGA and SORIGA. SORIGA: N = 120; p_m = 0.01, p_c = 0.7; r_r = 3; SMGA: N = 120; p_c = 0.7; g = 10×L (Royal Road) and g^* = 50×L (Deceptive 1 and 2)

		Royal Road		Deceptive 1		Deceptive 2	
τ	ρ	SORIGA	SMGA	SORIGA	SMGA*	SORIGA	SMGA*
10	0.05	30.91	**31.41**	0.825	**0.855**	**0.756**	0.753
10	0.60	12.05	**13.40**	0.768	**0.793**	0.576	**0.594**
10	0.95	15.44	**17.12**	0.886	**0,922**	0.554	**0.558**
200	0.1	**60.60**	57.80	0.901	**0.957**	**0.798**	**0.798**
200	0.60	**42.72**	42.15	0.881	**0.908**	0.780	**0.783**
200	0.95	39.73	**46.43**	**0.947**	0.939	**0.781**	0.780
1000	0.05	**63.34**	62.36	0.975	**0.994**	**0.800**	0.799
1000	0.60	**58.62**	54.85	0.959	**0.984**	0.796	**0.797**
1000	0.95	**57.84**	56.62	0.979	**0.988**	**0.797**	0.796

Due to the Sandpile's structure, avalanches and resulting mutations affect adjacent genes. However, the Sandpile may evolve in a grid structure other than a lattice. A small-world network should allow the avalanches to spread more quickly to distant bits in the chromosome and distant chromosomes in the ranked population. Further work will focus on the design of small-world grids for the Sandpile Mutation. In addition, parameter *g* needs to be addressed in order to understand how its variation affects the algorithm's performance and also to detect possible optimal values. Besides deceptive trap functions, the Sandpile will also be tested on hard combinatorial problems, with the aim of understanding how could the model behave when dealing with real-world problems.

6. ACKNOWLEDGEMENTS

Authors wish to thank FCT, *Ministério da Ciência e Tecnologia*, the Research Fellowships SFRH/BD/18868/2004 (also supported by *Fundação para a Ciência e a Tecnologia*, ISR/IST plurianual funding, through the POS_Conhecimento Program that includes FEDER funds). This work has also been supported by the Spanish MICYT project TIN2007-68083-C02-01, Junta de Andalucia CICE P06-TIC-02025 and Granada University PIUGR 9/11/06. Authors are grateful to R. Tinós and S. Yang for providing documentation on SORIGA that will be very useful in future work.

7. REFERENCES

[1] Abbass, H. A., Sastry K., and Goldberg, D. E. 2004. Oiling the wheels of change: The role of adaptive automatic problem decomposition in non-stationary environments. Technical Report 2004029, Illigal, University of Illinois at Urbana-Champaign, IL, USA

[2] Angeline, P. 1997. Tracking Extrema in Dynamic Environments. Proceedings of the 6th International Conf. on Evolutionary Programming, Springer, 335-345.

[3] Back, T. 1996.Evolutionary Algorithms in Theory and Practice. Oxford University Press.

[4] Bak, P., Tang, C., K. Wiesenfeld, K. 1987. Self-organized criticality: an explanation of 1/f noise. Physical Review of Letters, 381-384.

[5] Bak, P., Sneppen. K. 1993. Punctuated equilibrium and criticality in a simple model of evolution. Physical Review of Letters 71, 4083-4086.

[6] Baluja, S. 1994. Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, USA.

[7] Boettcher, S., Percus A.G. 2003. Optimization with extremal dynamics. Complexity 8(2), 57-62.

[8] Bonabeau, E., Dorigo, M., Threraulaz, G. 1999. Swarm intelligence: from natural to artificial systems. Oxford University Press

[9] Branke, J. 1999. Memory enhanced evolutionary algorithms for changing optimization problems. Proceedings of the 1999 Congress on Evolutionary Computation, 1875-1882

[10] Branke, J., Schmeck, H. 2002. Designing evolutionary algorithms for dynamic optimization problems. Theory and Application of Evolutionary Computation: Recent Trends, 239-262

[11] Cedeno, W., Vemuri. V.R. 1997. On the use of niching for dynamic landscapes. Proceedings of the 1997 Conference on Evolutionary Computation, IEEE, 361-367

[12] Cobb H.G. 1990. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA

[13] Colorni, A., Dorigo M., and Maniezzo, V. 1992. Distributed Optimization by Ant Colonies. Proceedings of the 1st European Conference on Artificial Life, MIT Press, 134-142

[14] Eiben, A.E., Hinterding R., Michalewicz Z. 1999. Parameter Control in Evolutionary. IEEE Transaction on Evolutionary Computation 3(2), 124-141

[15] Fernandes, C., Rosa A.C., and Ramos V. 2007. Binary Ant Algorithm. Proceedings of the 2007 Genetic and Evolutionary Computation Conference, ACM, 41-48

[16] Goldberg, D.E, Deb, K., Horn, J. 1992. Massively multimodality, deception and genetic algorithms. Parallel Problem Solving from Nature II, North-Holland, 37-46

[17] Goldberg, D.E., Smith, R.E. 1987. Nonstationary function optimization using genetic algorithms with dominance and diploidy. Proceedings of the 2nd International Conference on Genetic Algorithms, ACM, 59-68

[18] Grefenstette, J.J. 1992. Genetic algorithms for changing environments. Parallel Problem Solving from Nature II. North-Holland, 137-144,

[19] Guntsch, M., Middendorf, M. 2002. Applying population based ACO to dynamic optimization problems. Proceedings of 3rd International Workshop ANTS 2002, LNCS 2463, 111-122

[20] Harik, G. R. 1999. Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, Illigal, University of Illinois at Urbana-Champaign, IL, USA

[21] Huang, C., Kaur, J., Maguitman, A., and Rocha, L. 2007 Agent-based model of genotype editing. Evolutionary Computation 15(3), 253-289.

[22] Krink, T., Rickers P., René T. 2000. Applying self-organised criticality to evolutionary algorithms. Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, 375-384

[23] Krink, T., Thomsen, R. 2001. Self-Organized Criticality and Mass Extinction in Evolutionary Algorithms. Proceedings of the 2001 Congress on Evolutionary Computation, vol.2, 1155-1161

[24] Lorrañga, P., Lozano J.A. 2002. Estimation of distribution algorithms: A new tool for evolutionary computation. Boston: Kluwer Academic Publishers, Boston

[25] Mitchell, M. 1991. When will a GA outperform Hilclimbing? Advances in Neural Information Processing Systems 6, 51-58

[26] Muehlenbein, H. 1998. The equation for response to selection and its use for prediction. Evolutionary Computation 5(3), 303-346

[27] Ochoa, G., Madler-Kron C., Rodriguez R., Jaffe K. 2005. Assortative mating in genetic algorithms for dynamic problems. Proceedings of the 2005 EvoWorkshops, 617-622

[28] Pelikan, M., Goldberg D., Lobo F. 1999. A Survey of Optimization by Building and Using Probabilistic Models, Computational Optimization and Applications 21(1), 5-20

[29] Ramos, V., Fernandes C., and Rosa A.C. 2005. On self-regulated swarms, societal memory, speed and dynamics, Proceedings of ALifeX, MIT Press, 393-399

[30] Tinós, R. and Yang, S. 2007. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Genetic Programming and Evolvable Machines 8, 255-286

[31] Yang, S. 2005. Memory-enhanced univariate marginal distribution algorithms. Proceedings of the 2005 Congress on Evolutionary Computation, 2560-2567

[32] Yang, S. and Yao, X. 2005. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing 9(11), 815-834