Artificially Inducing Environmental Changes in Evolutionary Dynamic Optimization

Renato Tinós[1] and Shengxiang Yang[2(✉)]

[1] Department of Computing and Mathematics, FFCLRP, University of São Paulo, Ribeirão Preto, São Paulo 14040-901, Brazil
rtinos@ffclrp.usp.br

[2] Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, UK
syang@dmu.ac.uk

Abstract. Biological and artificial evolution can be speeded up by environmental changes. From the evolutionary computation perspective, environmental changes during the optimization process generate dynamic optimization problems (DOPs). However, only DOPs caused by intrinsic changes have been investigated in the area of evolutionary dynamic optimization (EDO). This paper is devoted to investigate artificially induced DOPs. A framework to generate artificially induced DOPs from any pseudo-Boolean problem is proposed. We use this framework to induce six different types of changes in a 0–1 knapsack problem and test which one results in higher speed up. Two strategies based on immigrants, which are used in EDO, are adapted to the artificially induced DOPs investigated here. Some types of changes did not result in better performance, while some types led to higher speed up. The algorithm with memory based immigrants presented very good performance.

1 Introduction

In a recent work [11], Steinberg and Ostermeir investigated the hypothesis that environmental changes can help molecular evolution to cross fitness valleys. They experimentally tested four strategies for inducing environmental changes in the evolution of an antibiotic resistance gene (*TEM-15* β-lactamase). One particular strategy, where low antibiotic resistance individuals are selected in the initial steps, produced very interesting results. When the evolutionary pathways were analysed, it was observed that an initially deleterious mutation allowed to access a promising part of the sequence space. This part of the sequence space was very difficult to be reached when environmental changes had not occurred.

The idea that biological and artificial evolution can be speeded up by environmental changes is not new [6,9,12]. Kashtan *et al.* [6] compared two strategies for inducing environmental changes in the *in silico* evolution of five models: (i) logic circuits; (ii) feed-forward logic circuits; (iii) feed-forward artificial neural networks; (iv) feed-forward circuits; (v) RNA structure. The two strategies were modularly varying goals and randomly varying goals. Greater speed

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 225–236, 2016.
DOI: 10.1007/978-3-319-45823-6_21

up was obtained for the first strategy, where subgoals are inserted or removed during the optimization process. Populations can spend long periods around metastable states. Environmental changes modify the fitness landscape and evolution dynamics [13], allowing populations to eventually escape from local optima and plateaus [6][1].

In the evolutionary computation (EC) perspective, the occurrence of environmental changes during artificial evolution generates dynamic optimization problems (DOPs). In recent years, there is an increasing interest in evolutionary dynamic optimization (EDO) [2,8]. However, to the best of the authors' knowledge, the works published in this area deal with DOPs where environmental changes are *intrinsic*. In other words, *artificially induced* DOPs are not considered. Here, we investigate artificially induced DOPs in a perspective of EDO. In [6], strategies for inducing environmental changes in *specific* DOPs were investigated. We propose a general framework to artificially induce environmental changes in *any* pseudo-Boolean optimization problem[2]. The proposed framework is based on the DOP benchmark generator introduced in [14], which will be presented in Sect. 2. The proposed framework will be presented in Sect. 3.

In the experiments used to test the proposed framework, environmental changes are artificially induced in the 0–1 knapsack problem in order to eventually speed up evolution. The experimental results are presented in Sect. 4. It is important to highlight that testing whether environmental changes can speed up evolution is only one of the possible motivations to artificially induced changes in EC. For example, we can artificially induce environmental changes in order to increase the robustness of the solutions [3]. Also, we can control when environmental changes can be inserted in some applications, e.g., those involving cooperation and competition [10]

From a programmer point of view, there are two main differences between artificially induced and intrinsic DOPs. In artificially induced DOPs, the programmer should decide *when* and *how* to change the problem, which is impossible in intrinsic DOPs. To this aim, one needs to answer two questions: (i) *When should the changes be inserted?* (ii) *How the fitness landscape should be modified?* We strongly believe that trying to answer these two questions opens new research possibilities in EDO. Researchers can investigate the best way to change the fitness landscapes from a theoretical point of view.

From a practical point of view, knowing beforehand when the changes occur, new algorithms and operators can be designed. For example, hypermutation re-introduces diversity by increasing the mutation rate after a change. Knowing when a change will occur allows to apply hypermutation some generations before the change. The development of new algorithms and operators is also important

[1] The idea of changing the static fitness landscape in order to make the optimization process easier is also present in other approaches. For example, in multi-objectivization, a single-objective problem is transformed into a multi-objective problem [7]. Another example is adding noise to the fitness function [5].

[2] In a pseudo-Boolean optimization problem P, the fitness function is $f_P(\mathbf{x}) \in \mathbb{R}$, where $\mathbf{x} \in \mathbb{B}^l$ is a candidate solution vector with dimension l.

because the goal in intrinsic and artificially induced DOPs can be different. In intrinsic DOPs, the goal is to track the moving optima, while in artificially induced DOPs we can be interested in finding the optima only for the static problem. Here, we test two very simple strategies to deal with artificially induced DOPs in Sect. 3.

2 DOP Benchmark Generator

Based on the analysis of fitness landscape changes in some DOPs, a benchmark generator for dynamic pseudo-Boolean optimization problems was proposed in [14]. The generator allows to create DOPs from any pseudo-Boolean optimization problem P with (static) fitness function $f_P(\mathbf{x})$, where $\mathbf{x} \in \mathbb{B}^l$. A DOP is considered as a sequence of static landscapes (environments) modified by changes [8]. In the DOPs created by the generator, the fitness function is given by:

$$f(\mathbf{x}, e) = f_P\big(g(\mathbf{x}, e)\big) + \Delta f\big(g(\mathbf{x}, e), e\big), \tag{1}$$

where e is the index of environment, i.e., it indicates a static fitness landscape between two consecutive changes [13]. Instead of computing the static fitness $f_P(.)$ at position $\mathbf{x}$, it is computed at position $g(\mathbf{x}, e)$. Besides, a deviation $\Delta f\big(g(\mathbf{x}, e), e\big)$ is added to $f_P(.)$. The generator allows to create 6 different types of DOPs based on the choice of $g(\mathbf{x}, e)$ and $\Delta f\big(g(\mathbf{x}, e), e\big)$, as described below.

2.1 DOP Type 1: DOP with Permutation

In this case, $\Delta f\big(g(\mathbf{x}, e), e\big) = 0$ and $g(\mathbf{x}, e)$ is given by a permutation of $\mathbf{x}$. In the generator, 3 different ways to permute $\mathbf{x}$ are employed.

DOP Type 1.1 (Permutation of the XOR Type): The candidate $\mathbf{x}$ is permuted according to: $g(\mathbf{x}, e) = \mathbf{x} \oplus \mathbf{m}(e)$, where:

$$\mathbf{m}(e) = \begin{cases} \mathbf{0}_l, & \text{for } e = 1 \\ \mathbf{m}(e-1) \oplus \mathbf{r}(e), & \text{for } e > 1 \end{cases} \tag{2}$$

where "$\oplus$" is the XOR operator and $\mathbf{r}(e)$ is a binary template that is randomly created in each environment e and contains $\lfloor \rho \cdot l \rfloor$ ones, where $0 \leq \rho \leq 1$. The change severity is controlled by ρ. DOP Type 1.1 produces the same type of change as the XOR DOP generator [14].

DOP Type 1.2 (Permutation Defined by a Permutation Matrix): The permutation is given by: $g(\mathbf{x}, e) = \mathbf{B}(e)\mathbf{x}$, where the permutation matrix $\mathbf{B}(e)$ is incrementally modified according to:

$$\mathbf{B}(e) = \begin{cases} \mathbf{I}_l, & \text{for } e = 1 \\ \mathbf{C}(e)\mathbf{B}(e-1), & \text{for } e > 1 \end{cases} \tag{3}$$

where $\mathbf{C}(e)$ is a permutation matrix obtained by randomly exchanging $\lfloor \rho \cdot l \rfloor$ lines of the l-dimensional identity matrix $\mathbf{I}_l$. In fact, the use of matrices would imply in a computational cost $O(l^2)$. This cost can be reduced to $O(l)$ by using an integer vector to record the positions of the permuted variables of $\mathbf{x}$. Similar strategies are adopted for other DOP types.

DOP Type 1.3 (Permutation According to a Set of Templates): The permutation is defined by:

$$g(\mathbf{x}, e) = \begin{cases} \mathbf{x} \oplus \mathbf{m}_j(e), \text{ if } \mathbf{x} \in \mathbf{s}_j(e), j = 1, \ldots, n_s \\ \mathbf{x}, \qquad\qquad \text{otherwise} \end{cases} \tag{4}$$

where $\mathbf{s}_j(e)$ is a template defining a hyperplane in $\mathbb{B}^l$ and n_s is the number of templates. The templates, or schemata in the genetic algorithms terminology, are composed of digits 0, 1 and * (do not care) and can be associated with subsets of solutions. Each template $\mathbf{s}_j(e)$ is given by:

$$\mathbf{s}_j(e) = \begin{cases} \mathbf{0}_l, & \text{for } e = 1 \\ \mathbf{r}_j, & \text{for } e = 2 \\ \mathbf{D}(e)\mathbf{s}_j(e-1), & \text{for } e > 2 \end{cases} \tag{5}$$

where $\mathbf{r}_j$ is a random template with order equal to o_s, and $\mathbf{D}(e)$ is a permutation matrix obtained by randomly exchanging o_s lines of the l-dimensional identity matrix. The template $\mathbf{m}_j(e) \in \mathbf{s}_j(e)$ contains $\frac{l-o_s}{2}$ ones generated in random non-fixed positions of $\mathbf{s}_j(e)$. The order of the template $\mathbf{s}_j(e)$ is equal to o_s for $e > 1$. The following combinations $(o_s, n_s) \in \{(3,1),(2,1),(1,1),(1,2),(1,3)\}$, corresponding to $\rho \in \{0.125, 0.25, 0.5, 0.75, 0.875\}$, are used.

2.2 DOP Type 2: Copying Decision Variables

Here, $\Delta f\big(g(\mathbf{x}, e), e\big) = 0$ and $g(\mathbf{x}, e)$ is a transformation that produces decision variables that are copies of other decision variables. Two ways of copying the variables are considered: one where the variables in $\mathbf{x}$ are copied from other variables in $\mathbf{x}$ and another where the variables are copied from those in a template.

DOP Type 2.1 (Copying Decision Variables Using a Linear Transformation): The candidate solutions are linearly transformed by: $g(\mathbf{x}, e) = \mathbf{L}(e)\mathbf{x}$, where $\mathbf{L}(e)$ is a binary matrix generated according to:

$$\mathbf{L}(e) = \begin{cases} \mathbf{I}_l, & \text{for } e = 1 \\ \mathbf{Q}(e), & \text{for } e > 1 \end{cases} \tag{6}$$

where $\mathbf{Q}(e)$ is a matrix obtained by randomly copying $\lfloor \rho \cdot \frac{l}{2} \rfloor$ lines of the l-dimensional identity matrix into other lines.

DOP Type 2.2 (Copying Decision Variables from a Template): The transformation is given by:

$$g(\mathbf{x}, e) = \begin{cases} \mathbf{m}(e), \text{ if } \mathbf{x} \in \mathbf{s}(e) \\ \mathbf{x}, \quad \text{if } \mathbf{x} \notin \mathbf{s}(e) \end{cases} \tag{7}$$

where $\mathbf{s}(e)$ is a template given by:

$$\mathbf{s}(e) = \begin{cases} \mathbf{0}_l, \quad \text{for } e = 1 \\ \theta(e), \text{ for } e > 1 \end{cases} \tag{8}$$

The order of the random template $\theta(e)$ is $l - \lfloor \rho \cdot \frac{l}{2} \rfloor$. The binary template $\mathbf{m}(e) \in \mathbf{s}(e)$ is randomly generated at each environment e.

2.3 DOP Type 3: Adding Fitness Deviation by a Set of Templates

In this DOP type, $\mathbf{x}$ is not transformed, i.e., $g(\mathbf{x}, e) = \mathbf{x}$. The fitness deviation $\Delta f\big(g(\mathbf{x}, e), e\big) = \Delta f(\mathbf{x}, e)$ is given by:

$$\Delta f(\mathbf{x}, e) = \sum_{j=1}^{n_s} a(\mathbf{x}, \mathbf{s}_j(e), e), \tag{9}$$

where n_s is the number of templates. The order of each template $\mathbf{s}_j(e)$ is o_s. The parameters o_s and n_s are defined in the same way as in DOP Type 1.3. In Eq. (9), $a\big(\mathbf{x}, \mathbf{s}_j(e), e\big)$ is given by:

$$a\big(\mathbf{x}, \mathbf{s}_j(e), e\big) = \begin{cases} \Delta f_j(e), \mathbf{x} \in \mathbf{s}_j(e) \\ 0, \qquad \mathbf{x} \notin \mathbf{s}_j(e) \end{cases} \tag{10}$$

where $\Delta f_j(e)$ is the fitness deviation for $\mathbf{s}_j(e)$. Here, $\Delta f_j(e)$ is randomly generated from a uniform distribution in the range $[-\rho f_{range}, \rho f_{range}]$ in each environment e. The value of f_{range} is given by the difference between the best and mean fitness in the initial population (or, if this difference is too small, by the best fitness in the initial population).

3 Framework for Inducing Environmental Changes

Here, changes are artificially induced in order to test whether they can speed up evolution. Changes are inserted according to one of the 6 DOP types described in Sect. 2; we want to test which one produces the best results for speeding up evolution of a genetic algorithm applied to the 0–1 knapsack problem. In fact, a little modification is introduced in DOP types 1.3, 2.2, and 3, as described in Sect. 3.2. The framework for inducing environmental changes in EDO is described in Sect. 3.1. Variants of the standard genetic algorithm used in EDO are also tested (Sect. 3.3).

3.1 Framework

As the objective is to speed up evolution for problem P, we propose a framework where the static environment for problem P is modified every τ iterations of the algorithm (generations). The DOP is seen as a sequence of environments, where the type of each environment is indicated by $d(e)$. While $d(e) = 0$ indicates that $f(\mathbf{x}, e) = f_P(\mathbf{x})$ (i.e., the e-th environment is equal to the static environment for problem P), $d(e) = c \neq 0$ indicates an environment produced by DOP type c, where $c \in \{1.1, 1.2, 1.3, 2.1, 2.2, 3\}$. When e is odd, i.e., $\text{mod}(e, 2) = 1$, the e-th environment is equal to the static environment for problem P, i.e., $d(e) = 0$. When e is even, i.e., $\text{mod}(e, 2) = 0$, two strategies are compared: (i) Static, where $d(e) = 0$; (ii) Dynamic, where $d(e) = c$ and c identifies the DOP Type for environments where $\text{mod}(e, 2) = 0$. Experiments with each one of the six DOP types will be presented in Sect. 4.

3.2 DOP Types

Some properties of the DOP types produced by the generator are described [14]:

- Neighbourhood relations: the transformation of the fitness landscapes for DOP Types 1.1 and 1.2 preserves the neighbourhood relations in the search space. In other words, instead of transforming the fitness landscape, we could move the population according to the respective transformation only one time after the change and compute $f(\mathbf{x}, e) = f_P(\mathbf{x})$ during τ generations. The neighbourhood relations are not preserved for the other DOP types.
- All solutions of the search space are changed for DOP Type 1.1. For DOP Types 1.3. and 3, the fractions of the search space affected by a change are equal to $\rho \in \{0.125, 0.25, 0.5, 0.75, 0.875\}$. For the remaining DOP types, the number of solutions of the search space affected by a change varies from 2^{l-1} to $2^l - 2$ for $\rho > 0$.

A consequence of the last property is that the change can have no effect in the dynamics of the population. For example, no effect will be observed when the solutions in the population are not among those affected by the fitness landscape modification. As we want to change the dynamics of the population here, we will use the knowledge about the best current solution in order to change the fitness landscape for DOP Types 1.3, 2.2 and 3. In this way, a small modification is introduced. In the DOP generator presented in [14], the first template $\mathbf{s}_j(e)$ for DOP Types 1.3, 2.2 and 3 is randomly chosen with no restriction. Here, the template is chosen assuring that $\mathbf{x_b}(e-1) \in \mathbf{s}_j(e)$, where $\mathbf{x_b}(e-1)$ is the best solution found in the e-th environment. Thus, DOPs produced by changes of types 1.3, 2.2 and 3 have the time-linkage property, i.e., knowing the current best solution influences the future dynamics of the problem [8].

3.3 Algorithms

The influence of artificially inducing changes in the 0–1 knapsack problem optimized by a standard genetic algorithm (GA) is investigated here. We also test variants of approaches used in EDO that replaces part of the population by immigrants. Two types of immigrants are tested:

- *Random immigrants* (RIs) [1]: when this strategy is used, 20 % of the population is replaced by randomly generated individuals.
- *Memory immigrants* (MIs) [15]: when this strategy is used, 10 % of the population is replaced by individuals stored in a memory population.

Instead of inserting immigrants in every generation, they are inserted only after a change. Also, as we want to optimize problem P, the memory is formed by the best individuals found in environments with $d(e) = 0$. As all the individuals in the memory were generated in environments with the same fitness landscape, it is not necessary to re-evaluate them when they are re-introduced in the population. The MIs are re-introduced in environments where $d(e) = 0$. The maximum size of the memory population is equal to the size of the GA population (*popsize*). When the maximum size is reached, a random individual of the memory population is replaced by the new individual, with exception for the best individual in the memory population. One can observe that we are using the knowledge about the changes in the problem in order to design the memory immigrants approach.

4 Experiments

4.1 Experimental Design

The fitness function for the 0–1 knapsack problem [4] is given by:

$$f_P(\mathbf{x}) = \sum_{i=1}^{l} p_i x_i - R(\mathbf{x}) \tag{11}$$

where $\mathbf{x} \in \mathbb{B}^l$ defines the subset of items in the knapsack and p_i is the profit of the i-th item. The penalty $R(\mathbf{x})$ is equal to zero if the sum of the weights in the knapsack is less than the knapsack capacity C. Otherwise, the penalty is:

$$R(\mathbf{x}) = \alpha \left(\sum_{i=1}^{l} w_i x_i - C \right) \tag{12}$$

where w_i is the weight of the i-th item and $\alpha = \max_{i=1,\ldots,l}(p_i/w_i)$. The profits and weights are integers randomly generated in the beginning of each run.

The profits are in the range $[40, 100]$, while the weights are in the range $[5, 20]$. The capacity C is equal to 50% of the sum of all weights. The objective is to maximize the fitness given by Eq. (11).

For all algorithms, the population size ($popsize$) is 100. Tournament selection, elitism, bit flip mutation, and uniform crossover are employed. The mutation rate is $1/l$, while the crossover rate is 0.6. In tournament selection, the best among 3 individuals randomly chosen is selected. Results for 4 algorithms, where RIs and MIs are inserted or not, are tested. The 2 strategies described in Sect. 3.1 are tested. For the dynamic strategy, results for runs with each one of the 6 DOP types are presented. The results of 50 runs for each combination of algorithm, dimension (l), change severity (ρ), and DOP strategy are presented. In the runs, the change period (τ) is equal to 500 generations. Each algorithm is run for l seconds. As the execution time is fixed and the number of evaluations for the algorithms can be different, the number of generations can also be different.

The best fitness obtained in each run is compared to the evaluation of the global optimum obtained by dynamic programming. The complexity of dynamic programming for the 0–1 knapsack problem is $O(lC)$, i.e., if C is polynomial, the algorithm runs in polynomial time. However, for the general case, the problem is NP-complete. In the experiments presented in the next section, the best fitness is stored only for the environments where $d(e) = 0$. In this way, the best from all generations are considered for the static strategy. However, for the dynamic strategy, only the results for environments where the index is odd are recorded.

4.2 Experimental Results

Table 1 shows the average error for the experiments. The average error is obtained by comparing, for each run, the static fitness of the global optimum with the static fitness of the best solution found by the algorithm. In order to test whether the best results are due to the use of immigrants (instead of due to changing the environment), results for runs of the static case with RIs and MIs are also presented. The results for the dynamic (with different DOP types) strategy are compared to the respective results for the static strategy. The Wilcoxon signed-rank test with the confidence level equal to 0.95 is used to test the statistical significance of the results.

Changing the environments resulted in better performance for some DOP types, but not for all. The worse results were obtained for DOP Types 1.1 and 1.2. As commented in Sect. 3.2, neighbourhood relations in the search space are preserved for changes in DOP Types 1.1 and 1.2. The changes produce the same effect that uniformly moving the solutions to other regions of the search space. Uniformly moving the individuals of the algorithm to new regions of the search space did not result in a better performance in the experiments.

The best results were obtained by DOP Type 2.2, followed by DOP Type 3. It is interesting to observe that, even directly optimizing $f_P(.)$ in approximately

Table 1. Average error (over 50 runs) for static and dynamic environments. The symbol s indicates that the results are statistically different according to the Wilcoxon signed-rank test. Bold face indicates that the results for the changing environments are statistically better than the respective results for the static environment. Italic face indicates the best result for each dimension.

RI	MI	ρ	DOP type						
			Static	1.1	1.2	1.3	2.1	2.2	3
$l = 200$									
No	No	0.125	0.8 ± 1.4	0.7 ± 1.1	0.7 ± 1.3	**0.1 ± 0.4 (s)**	**0.2 ± 0.4 (s)**	**0.1 ± 0.2 (s)**	**0.1 ± 0.4 (s)**
		0.500		2.3 ± 1.8 (s)	1.5 ± 1.6 (s)	**0.2 ± 0.5 (s)**	0.8 ± 1.1	**0.1 ± 0.3 (s)**	**0.2 ± 0.5 (s)**
		0.875		3.1 ± 2.2 (s)	1.9 ± 2.4 (s)	0.5 ± 1.1	1.3 ± 1.4	**0.1 ± 0.4 (s)**	**0.2 ± 0.5 (s)**
No	Yes	0.125	0.7 ± 1.1	1.4 ± 1.7 (s)	1.5 ± 2.0 (s)	**0.3 ± 0.5 (s)**	0.9 ± 1.2	**0.2 ± 0.6 (s)**	0.6 ± 1.1
		0.500		1.6 ± 1.7 (s)	1.6 ± 1.9 (s)	0.5 ± 0.9	1.5 ± 1.8 (s)	**0.3 ± 0.5 (s)**	0.7 ± 1.2
		0.875		1.4 ± 1.9 (s)	1.5 ± 2.0 (s)	0.7 ± 1.4	1.1 ± 1.5 (s)	**0.1 ± 0.5 (s)**	0.8 ± 1.2
Yes	No	0.125	1.1 ± 1.7	0.7 ± 1.4	0.9 ± 1.2	**0.2 ± 0.6 (s)**	**0.4 ± 0.7 (s)**	***0.0* $\pm$ *0.2* (s)**	**0.2 ± 0.5 (s)**
		0.500		1.9 ± 1.9 (s)	2.0 ± 2.5	**0.2 ± 0.5 (s)**	0.8 ± 1.1	**0.1 ± 0.4 (s)**	**0.2 ± 0.5 (s)**
		0.875		2.2 ± 1.9 (s)	1.6 ± 2.0	**0.4 ± 0.7 (s)**	1.2 ± 1.5	***0.0* $\pm$ *0.2* (s)**	**0.1 ± 0.4 (s)**
Yes	Yes	0.125	0.7 ± 1.2	1.2 ± 1.5	1.4 ± 1.9 (s)	**0.3 ± 0.5 (s)**	0.9 ± 1.4	**0.1 ± 0.4 (s)**	0.7 ± 1.3
		0.500		1.4 ± 1.7 (s)	1.8 ± 1.9 (s)	0.5 ± 1.0	1.2 ± 1.8	**0.2 ± 0.6 (s)**	0.7 ± 1.3
		0.875		1.2 ± 1.6 (s)	1.0 ± 1.5	0.7 ± 1.1	1.5 ± 1.8 (s)	**0.1 ± 0.3 (s)**	0.6 ± 1.0
$l = 500$									
No	No	0.125	4.4 ± 2.8	61.6 ± 9.6 (s)	60.8 ± 10.0 (s)	13.0 ± 3.5 (s)	27.8 ± 5.0 (s)	4.5 ± 2.0	7.1 ± 2.8 (s)
		0.500		80.7 ± 15.5 (s)	72.0 ± 18.6 (s)	9.1 ± 3.1 (s)	31.4 ± 7.4 (s)	4.5 ± 1.5	4.6 ± 1.7
		0.875		124.0 ± 15.5 (s)	65.9 ± 11.7 (s)	16.2 ± 5.6 (s)	75.1 ± 10.0 (s)	7.1 ± 2.4 (s)	6.8 ± 2.2 (s)
No	Yes	0.125	6.3 ± 3.4	**4.9 ± 2.3 (s)**	5.7 ± 3.0	5.7 ± 2.7	**4.9 ± 2.6 (s)**	**2.9 ± 1.9 (s)**	5.6 ± 3.1
		0.500		5.3 ± 2.8	8.2 ± 3.1 (s)	**4.1 ± 2.7 (s)**	5.4 ± 3.5	**2.0 ± 1.6 (s)**	**3.6 ± 2.3 (s)**
		0.875		7.7 ± 3.2 (s)	7.3 ± 3.5	**4.7 ± 2.7 (s)**	7.1 ± 3.9	**2.0 ± 1.6 (s)**	**4.2 ± 2.3 (s)**
Yes	No	0.125	5.2 ± 2.4	31.6 ± 7.4 (s)	76.1 ± 11.2 (s)	16.5 ± 4.3 (s)	29.5 ± 4.8 (s)	7.1 ± 2.0 (s)	**4.3 ± 2.2 (s)**
		0.500		75.3 ± 14.8 (s)	63.6 ± 13.3 (s)	19.8 ± 4.3 (s)	62.2 ± 9.4 (s)	**3.9 ± 1.6 (s)**	4.7 ± 2.1
		0.875		74.2 ± 16.5 (s)	71.6 ± 11.7 (s)	28.7 ± 6.9 (s)	83.6 ± 10.6 (s)	8.0 ± 2.6 (s)	6.9 ± 3.3 (s)
Yes	Yes	0.125	5.0 ± 2.6	5.5 ± 3.0	6.0 ± 3.3	**3.8 ± 2.2 (s)**	7.6 ± 4.6 (s)	**2.7 ± 1.9 (s)**	5.7 ± 2.6
		0.500		5.7 ± 3.4	4.8 ± 2.7	4.4 ± 2.5	7.5 ± 3.4 (s)	**3.2 ± 2.1 (s)**	6.2 ± 3.1 (s)
		0.875		$5.7 \pm \pm 3.4$	5.5 ± 2.8	4.7 ± 3.2	7.6 ± 4.1 (s)	***1.8* $\pm$ *1.6* (s)**	6.6 ± 3.3 (s)

half of the generations, the algorithms eventually obtained better results for the changing environments. With few exceptions, the dynamic strategy with DOP Type 2.2 resulted in better performance than the static strategy. Table 2 shows the percentage of successful runs, i.e., where the global optimum was found. For $l = 200$, the best result for the static case is 68%, while the global optimum was found in 96% of the runs of the algorithm with RIs for the dynamic strategy with DOP Type 2.2. The best results for the changing environments generally were obtained when the immigrants strategies were employed. However, immigrants generally did not result in better performance for the static environment. In particular, the best results for the changing environments for the experiments with $l = 500$ were obtained when MIs were inserted.

Table 2. Percentage of runs where the global optimum was found. Bold face indicates that the result for the dynamic environment is better than the respective result for the static environment. Italic face indicates the best result for each dimension.

RI	MI	ρ	DOP Type ($l = 200$)							DOP Type ($l = 500$)						
			Static	1.1	1.2	1.3	2.1	2.2	3	Static	1.1	1.2	1.3	2.1	2.2	3
No	No	0.125	62	62	62	**88**	**84**	**94**	**92**	4	0	0	0	0	2	0
		0.500		22	38	**88**	54	**94**	**82**		0	0	0	0	2	2
		0.875		16	40	**70**	40	**92**	**86**		0	0	0	0	0	0
No	Yes	0.125	64	46	50	**78**	54	**84**	**68**	2	2	2	2	2	**16**	0
		0.500		36	38	62	44	**76**	62		0	0	**8**	**4**	**22**	**6**
		0.875		44	48	**68**	52	**90**	56		0	2	2	0	**24**	2
Yes	No	0.125	56	**62**	54	**82**	**76**	***96***	**84**	0	0	0	0	0	0	**2**
		0.500		32	34	**80**	56	**88**	**84**		0	0	0	0	**2**	0
		0.875		24	42	**74**	42	***96***	**88**		0	0	0	0	0	0
Yes	Yes	0.125	68	44	48	**76**	58	**90**	62	0	**2**	0	**8**	**2**	**14**	0
		0.500		42	40	**72**	54	**86**	66		**2**	**4**	**2**	0	**6**	0
		0.875		44	54	58	46	**92**	66		**2**	0	**4**	0	***26***	0

5 Conclusions

We investigated artificially induced DOPs in this paper. Environmental changes can be artificially induced for different reasons, e.g., for speeding up evolution. A framework for generating artificially induced DOPs from any pseudo-Boolean problem was presented. Six different types of changes can be induced in the framework proposed here. The experiments with DOPs generated based on the 0–1 knapsack problem showed that better performance was obtained only for some change types and change severities.

Particularly, changes generated in DOP Type 2.2 resulted in better performance. For static environments, the best percentages of successful runs were: 68% ($l = 200$) and 4% ($l = 500$). For DOP Type 2.2, the best percentages of successful runs were: 96% ($l = 200$) and 26% ($l = 500$). Results not shown here for experiments with $l = 300$ and $l = 400$ also indicate better performance for the dynamic strategy[3]. The best results were obtained when random and memory immigrants were employed. The memory immigrants approach employed here makes use of the knowledge about the sequence of changes in the problem. This is an example of designing strategies to deal with artificially induced DOPs. The knowledge about the changes and their impact are usually not known in intrinsic DOPs. In artificially induced DOPs, the designer controls when and how to change the environments.

[3] The best percentages of successful runs for DOP Type 2.2 were 66% ($l = 300$) and 42% ($l = 400$), against 28% ($l = 300$) and 18% ($l = 400$) for the static environments.

Several future works are possible. Concerning the framework proposed here, it is necessary to better understand the impact of the different change types in different problems and state-of-art algorithms developed for static and dynamic optimization. In artificially induced DOPs, it is necessary to theoretically investigate how and when to change the environment according to the objectives of the programmer. Also, it is necessary to investigate new algorithms and operators that make use of the knowledge about the changes.

Acknowledgments. This work was funded partially by FAPESP under grant 2015/06462-1 and CNPq in Brazil, and partially by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under grant EP/K001310/1.

References

1. Cobb, H.G., Grefenstette, J.J.: Genetic algorithms for tracking changing environments. In: Proceedings of 5th International Conference on Genetic Algorithms, pp. 523–530 (1993)
2. Cruz, C., González, J., Pelta, D.: Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput. **15**, 1427–1448 (2011)
3. Fu, H., Sendhoff, B., Tang, K., Yao, X.: Robust optimization over time: problem difficulties and benchmark problems. IEEE Trans. Evol. Comp. **19**(5), 731–745 (2015)
4. Han, K.H., Kim, J.H.: Genetic quantum algorithm and its application to combinatorial optimization problem. In: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 2, pp. 1354–1360 (2000)
5. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments-a survey. IEEE Trans. Evol. Comp. **9**(3), 303–317 (2005)
6. Kashtan, N., Noor, E., Alon, U.: Varying environments can speed up evolution. Proc. Natl. Acad. Sci. **104**(34), 13711–13716 (2007)
7. Knowles, J.D., Watson, R.A., Corne, D.W.: Reducing local optima in single-objective problems by multi-objectivization. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, p. 269. Springer, Heidelberg (2001)
8. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol. Comp. **6**, 1–24 (2012)
9. Parter, M., Kashtan, N., Alon, U.: Facilitated variation: how evolution learns from past environments to generalize to new environments. PLOS Comput. Biol. **4**(11), e1000206 (2008)
10. Richter, H.: Coevolutionary intransitivity in games: a landscape analysis. In: Mora, A.M., Squillero, G. (eds.) EvoApplications 2015. LNCS, vol. 9028, pp. 869–881. Springer, Heidelberg (2015)
11. Steinberg, B., Ostermeier, M.: Environmental changes bridge evolutionary valleys. Sci. Adv. **2**(1), e1500921 (2016)
12. Tan, L., Gore, J.: Slowly switching between environments facilitates reverse evolution in small populations. Evolution **66**(10), 3144–3154 (2012)
13. Tinós, R., Yang, S.: Analyzing evolutionary algorithms for dynamic optimization problems based on the dynamical systems approach. In: Yang, S., Yao, X. (eds.) Evolutionary Computation for Dynamic Optimization Problems. SCI, vol. 490, pp. 241–267. Springer, Heidelberg (2013)

14. Tinós, R., Yang, S.: Analysis of fitness landscape modifications in evolutionary dynamic optimization. Inf. Sci. **282**, 214–236 (2014)
15. Yang, S.: Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evol. Comput. **16**(3), 385–416 (2008)