International Journal of Computational Intelligence Systems, Vol. 8, No. 4 (2015) 667-689

Algorithm portfolio based scheme for dynamic optimization problems

Jenny Fajardo Calderín [1] *, Antonio D. Masegosa [2,3], David A. Pelta [4]

[1] *Dept. of Artificial Intelligence and Infrastructure and Systems, Polytech. Higher Inst. José A. Echeverría, La Habana, Cuba*

E-mail: jfajardo@ceis.cujae.edu.cu

[2] *Deusto Institute of Technology, University of Deusto, Bilbao, Spain*

E-mail: ad.masegosa@deusto.es

[3] *IKERBASQUE, Basque Foundation for Science, Bilbao, Spain*

[4] *Dept. of Computer Science and AI, Research Center for ICT (CITIC-UGR), University of Granada, Granada, Spain*

E-mail: dpelta@decsai.ugr.es

Received 18 December 2014

Accepted 4 March 2015

Abstract

Since their first appearance in 1997 in the prestigious journal Science, algorithm portfolios have become a popular approach to solve static problems. Nevertheless and despite that success, they have not received much attention in Dynamic Optimization Problems (DOPs). In this work, we aim at showing these methods as a powerful tool to solve combinatorial DOPs. To this end, we propose a new algorithm portfolio for this type of problems that incorporates a learning scheme to select, among the metaheuristics that compose it, the most appropriate solver or solvers for each problem, configuration and search stage. This method was tested over 5 binary-coded problems (dynamic variants of OneMax, Plateau, RoyalRoad, Deceptive and Knapsack) and compared versus two reference algorithms for these problems (Adaptive Hill Climbing Memetic Algorithm and Self Organized Random Immigrants Genetic Algorithm). The results showed the importance of a good design of the learning scheme, the superiority of the algorithm portfolio against the isolated version of the metaheuristics that integrate it, and the competitiveness of its performance versus the reference algorithms.

Keywords: algorithm portfolio, dynamic optimization problems, learning, algorithm selection problem, combinatorial problems

*This work has been supported by the research projects TIN2011-27696-C02-01 and TEC2013-45585-C2-2-R from the Spanish Ministry of Economy and Competitiveness, P11-TIC-8001 from the Andalusian Government (including FEDER funds from the European Union), and PC2013-71A from the Basque Government. Jenny Fajardo has also been supported by a scholarship from the Ibero-American University Association for Post Graduate Studies (AUIP).

1. Introduction

An optimization problem can be considered as "dynamic" when any of its components, namely objective function, constraints, size, variables domain, etc., change with time. Dynamic optimization problems (DOPs) are as follows:

$$\text{DOPs} = \{optimize\ f(x,t); x \in F(t) \subset S; t \in T\}$$

where:

- S is the search space
- t represents time
- $f : S \times T \longrightarrow R$ is the objective or cost function whose definition depends on t.
- $F(t)$, is the set of feasible solutions at time t, $F(t) \subset S$.

Most of the academic research in DOPs has been done using artificial problems, because they allow to properly studying several factors related with the changes, as severity, frequency or dynamism type. Examples of artificial problems are the well-known moving peaks benchmark [28, 29]; dynamic versions of static continuous functions like Sphere, Griewank, Rastrigin, Ackley, etc. [23]; or dynamic versions of combinatorial problems as knapsack problem [3, 21, 35], moving parabola [1], bitwise comparison problems [10, 41], and planning problems [8, 11, 26, 37, 38].

When dealing with DOPs, some aspects are assumed: the changes are gradual; it is not possible to solve the problem from scratch after a change; and the current information should be useful to produce a faster adaptation to the change.

The last years showed an increasing interest on solving them using metaheuristic methods [2, 7, 30, 49, 51]. One can find evolutionary algorithms [50], multiswarm techniques [2], ant colony optimization [19,27], cooperative strategies [17,18,25] and so on.

When solving static problems with metaheuristics, parameter setting is not a trivial task. As a consequence, researchers enhance their methods with some learning features to change the parameters during the run. This is also being done in the context of DOPs, where there is a growing tendency on using learning mechanism to change the parameters, operators, etc. of the method during the search [17, 18, 24, 27, 46, 51].

Another non trivial aspect is the selection of the metaheuristic to solve the problem at hand. In the context of static problems, one of the most common approaches for this issue is the algorithm portfolio [20, 31, 32]. It consists in a set of algorithms that are executed iteratively or in parallel, in order to solve the problem. A learning scheme is used to select the most suitable algorithm at every stage of the search or to distribute the available execution time among them. This decision is typically based on the algorithm's performance. This type of methods have shown to be very competitive in problems as supply chain optimization [48], numerical optimization [32], vehicle routing [39] or satisfiability [47], among others.

Despite this success in static problems, algorithm portfolios have not received much attention in DOPs. For this reason, in this paper we intend to deepen in this topic, to show algorithm portfolios as a powerful alternative to solve DOPs. Concretely, we propose a new algorithm portfolio for combinatorial DOPs emphasizing the role of the learning scheme.

First, we will analyze if learning makes sense, what learning scheme is the most appropriate and how the learning works along the search process. Then we will study if the portfolio obtains better results than their composing algorithms, and finally, we will compare the best portfolio variant against two reference algorithms: Adaptive Hill Climbing Memetic Algorithm and Self Organized Random Immigrants Genetic Algorithm. The computational experiments are done over five combinatorial optimization problems and the results are analyzed using statistical testing. It is important to remark that we will consider DOPs with dynamism only in the fitness function.

The article is organized as follows. Section 2 discusses the literature related to our proposal. The algorithm portfolio, its learning scheme and the different learning variants considered are presented in Section 3. In Section 4, experimental framework, we show the combinatorial DOPs used to test the

method, the details of the experimentation, the reference algorithms with which our proposal is compared, the composition of the portfolio, the performance measures and the non-parametric tests employed for the statistical assessment of the results. The next part of the paper is devoted to analyze the results of the experimentation and the comparison versus the reference algorithms. Finally, we provide the conclusions of this work in Section 6.

2. Related Work

The idea of algorithm portfolios was proposed by Huberman and coauthors in 1997 [20]. Inspired by basic economic concepts of risk management, these authors presented an approach to allocate a limited amount of resources (CPU time) among a set of algorithms according to its expected "benefit" (average fitness) and "risk" (fitness variance). The allocation is done in such a way that the benefit is maximized and the risk minimized.

The original concept has evolved along these years and nowadays, one of the most accepted definitions is the next one [16]: a collection of different algorithms and/or copies of the same algorithm that are run in one or more CPUs. We can classify algorithm portfolios in different categories depending on: 1) how the algorithms are run and 2) when the available running time is distributed among them [16, 33]. Regarding the first criterion, there are three classes [16]: *parallel*, where all the algorithms are run concurrently in different processors [32]; *interleaved on a single processor*, where the algorithms are run alternatively, simulating parallelism in one processor [14]; and *sequential with restart*, where at each iteration, a randomly selected algorithm is executed for a fixed amount of time (every run of the same algorithm uses a different random seed) [14]. Respect to the second criterion, the algorithm portfolios can be classified in two categories [14,33]: *static*, if the distribution of the available CPU time among the algorithms is fixed before the run [32, 48], or *dynamic*, if it is done along the search process [14].

We can find in the literature some references about the application of algorithm portfolios to solve DOPs. For example, in [40], the authors propose a static portfolio to solve the Inventory Routing Problem with stochastic demands. In this case, the algorithms are run in parallel without information feedback, and the portfolio is composed by variants of the genetic algorithm. Another work on this topic is found in [39], where an algorithm portfolio is used to solve the Dynamic Vehicle Routing Problem with stochastic demands. The paper presents a static algorithm portfolio that combines trajectory-based and population-based algorithms that are run in parallel with no information exchange.

A similar approach to algorithm portfolios that has been used in DOPs are the hyperheuristics [5]. They can be defined as search methods or learning mechanisms for the selection and generation of heuristics in order to solve a particular optimization problem. They can also be seen as high-level methods that given a set of low-levels heuristics for a particular problem, are able to automatically produce a proper combination of these low-level heuristics to solve the instance at hand. Among the high-level methods for the selection/generation/combination of heuristics we can find tabu search, variable neighborhood search, genetic algorithms or data mining techniques [5]. In the context of static problems, these methods have been applied to graph coloring, production planning, work-force scheduling, constraint satisfaction or vehicle routing [5, 6].

Regarding DOPs, its application is fairly recent. In [45], Gonul et al. proposed a framework for DOPs that hybridizes hyperheuristics and Population Based Incremental Learning. The selection of low-level heuristics is done through a scoring system and reinforcement learning. They tested two variants of their proposal on dynamic binary functions with successful results. Topcuoglu and coauthors present a hyperheuristic to solve DOPs in [43] whose high-level heuristic selection method is an evolutionary technique known as memory/search algorithm. The experimentation done over the Dynamic Generalized Assignment Problem and the Moving Peaks Benchmark showed the better performance of this method versus the canonical memory/search algorithm. In [22], the authors test, using also the Moving Peaks Benchmark, several hy-

perheuristics that combine five high-level heuristic selection methods (simple random, greedy, choice function, reinforcement learning, random permutation descent) with seven move acceptance criteria (all moves, only improving, improving and equal, exponential Monte Carlo with counter, great deluge, simulated annealing, simulated annealing with reheating). The best performing variant (choice function - improving and equal) improved the results obtained by state-of-the-art algorithms for this problem.

As we have seen, the use of algorithm portfolios and hyperheuristics is becoming more popular in DOPs. In this sense, although algorithm portfolios and hyperheuristics share a similar approach (they combine a set of heuristics), it is important to highlight that they present two important differences. On one hand, hyperheuristics usually work with low-level heuristics whereas algorithm portfolios do it with higher-level methods as metaheuristics. On the other hand, and in our opinion the main difference, hyperheuristics works over a single solution of the problem (they can be seen as trajectory-based methods) while algorithm portfolios works over a set of independent solutions at the same time. Taking into account these two aspects, the characteristics of our proposal, which we describe in the next section, fits better to the category of algorithm portfolios, since our method deals with metaheuristics that works over their own and independent solutions.

3. Algorithm Portfolio

The main motivation to build an algorithm portfolio is to avoid deciding on a single algorithm to solve the problem at hand [20]. In this contribution, our portfolio is made of a set of metaheuristics. Using a credit based approach, the portfolio selects which metaheuristic to run at every stage of the search. Then, a credit assignment is done based on the performance of the selected metaheuristic. According to the categories of algorithm portfolios showed in the last section, our proposal can be classified as a dynamic algorithm portfolio that interleaves the run of the metaheuristics on a single processor.

Algorithm 1: Portfolio Scheme Pseudocode

```
1  n = size of the portfolio;
2  for (j = 1; j ⩽ n; j++) do
3  |  initialize a_j ;
4  end
5  while (not stop-condition) do
6  |  if change is detected then
7  |  |  for (j = 1; j ⩽ n; j++) do
8  |  |  |  re-evaluate x_curr^{a_j} or p_curr^{a_j};
9  |  |  |  recalculate x_best;
10 |  |  end
11 |  end
12 |  a_i ← select an algorithm from A with
   |  probability P_selec^{a_i};
13 |  x_c ← run a_i for one iteration;
14 |  if f(x_c) > f(x_best) then
15 |  |  x_best ← x_c;
16 |  |  for (j = 1; j ⩽ n; j++) do
17 |  |  |  update x_curr^{a_j} or p_curr^{a_j} with x_best;
18 |  |  end
19 |  end
20 |  perform credit assignment to a_i;
21 |  for (j = 1; j ⩽ n; j++) do
22 |  |  update selection probability P_selec^{a_j};
23 |  end
24 end
```

More formally, our portfolio is composed by a set of metaheuristics A = $\{a_1, \ldots, a_n\}$, being each a_i a trajectory or population-based metaheuristic. Every a_i has a single current solution $x_{curr}^{a_i}$ (if we consider a trajectory-based method) or a current population $p_{curr}^{a_i}$ (in population-based metaheuristics).

Algorithm 1 shows the inner working of the portfolio. After the initialization of the algorithms, the method goes into the main loop. Firstly, if a change is detected, $x_{curr}^{a_i}$ or $p_{curr}^{a_i}$ are re-evaluated, and the new global best solution (x_{best}) is calculated.

Subsequently, an a_i is selected and run for one iteration (the definition of what we consider by an iteration is given at the end of this section). The selection strategy employed in this step is the well-known roulette-wheel method, where the portion or selection probability of each a_i is assigned according to its credit w_i. This probability is calculated as:

$$P_{selec}^{a_i} = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{1}$$

Table 1. Details of the steps the algorithm portfolio accomplishes in a different way depending on whether the selected algorithm is trajectory-based or population-based.

Step (Pseudocode line in Algorithm 1)	*Trajectory-based*	*Population-based*
Algorithm initialization (3)	random generation of a single solution following a uniform distribution	random generation of the population following a uniform distribution
Re-evaluation (8)	re-evaluate $x_{curr}^{a_i}$	reevaluate the whole population $p_{curr}^{a_i}$
One iteration (13)	application of the neighborhood operator to $x_{curr}^{a_i}$ and evaluation of the acceptance criterion for the move	perform a sequence of applications of the corresponding operators
x_c (13)	solution resulting from applying the neighborhood operator to $x_{curr}^{a_i}$	individual generated in the sequence of applications of the operators
$x_{curr}^{a_i}/p_{curr}^{a_i}$ update with x_{best}(17)	replace $x_{curr}^{a_i}$ by x_{best}	x_{best} replace the worst individual in the population $p_{curr}^{a_i}$

The solution generated by the algorithm selected in one iteration, x_c, is then compared versus the current global best x_{best}. If x_c is better than x_{best}, the algorithm portfolio refreshes it and then, it updates the current solution or population of the rest of algorithms ($a_j \neq a_i$) with the new x_{best} (the details of the obtaining of x_c and the updates of the current solutions and populations are also described at the end of the section).

In the last stage of the main loop, the method assigns a certain credit to a_i and recalculates the selections probabilities of the algorithms according to Equation 1. The amount of credit assigned to a_i depends on the quality of the solution x_c and the credit scheme employed. The details of this part of the algorithm portfolio will be described in the next subsection.

Some of the steps mentioned above are accomplished in a different way depending on whether the considered algorithm is trajectory-based or population-based. Table 1 shows how the algorithm portfolio carries out these steps in each case. The most important differences appears in how an iteration is performed, how x_c is obtained and how the current solution or population is updated. Regarding the iteration process and the obtaining of x_c, in trajectory-based algorithms, one iteration corresponds to one application of the neighborhood operator to $x_{curr}^{a_i}$ and the evaluation of the acceptance criterion for the move (e.g., tabu and aspiration criteria in Tabu Search, acceptance probability in Simulated Annealing, etc.). x_c represents the solution resulting from the application of the neighborhood operator. In population-based methods, one iteration corresponds to the sequential application of their operators (e.g., selection $\rightarrow$ crossover $\rightarrow$ mutation $\rightarrow$ replacement, in Genetic Algorithms). When a crossover operator is applied, only one of the two individuals obtained (randomly selected according to a uniform distribution) is considered for the next step of the sequence of operator applications. In case the method considers replacement, the individual generated replaces the worst parent. x_c corresponds to the individual generated in this process.

The update of $x_{curr}^{a_i}$ for trajectory-based algorithms, performed in the step 17 of Algorithm 1, consists on replacing $x_{curr}^{a_i}$ by x_{best}. Regarding population-based algorithms, in this step, x_{best} replaces the worst individual of the population.

3.1. Learning scheme

The credit assignment mechanism implemented by the portfolio is, in fact, a learning scheme whose objective is to learn which the best performance method/s for the problem at hand is/are. For this reason, we will also refer to the credit assignment mechanism as learning scheme.

The credit assigned to a metaheuristic a_i at time $t+1$ is calculated as:

$$w_i(t+1) = w_i(t) + r_i(t) - l_i(t) \tag{2}$$

where:

- t is the current time.

- $w_i(t+1)$ is the credit obtained by a_i at time step $t+1$.
- $w_i(t)$ is the current credit of a_i.
- $r_i(t)$ is the reward assigned to a_i. It is a value greater or equal to zero that is determined as a function of the quality of the solution generated by a_i (x_c) and the quality of x_{best}.
- $l_i(t)$ is the penalization assigned to a_i if the generated solution (x_c) is worse than x_{best}. It is defined as follows:

$$l_i(t) = w_i(t) * Q \tag{3}$$

where: Q is a penalty term and $Q \in [0,1]$.

We assume here that every possible definition or variant for the reward, penalization and credit update, lead to a different learning scheme.

3.2. Learning scheme variants

When facing dynamic optimization problems, it is critical to decide what to do with the learning gained (credit assigned) by the portfolio. The potential definitions of the three components of the learning scheme, namely $w_i(t)$, $r_i(t)$ and $l_i(t)$, lead to variations that will be analyzed next in order to detect the best alternatives. The potential definitions are described next:

Current credit ($w_i(t)$):

- Restart (RS): current credit is set to zero when a change is detected. The rationale behind this idea is that we need to face a new problem and we need to detect from scratch which are the good methods for the new situations (we forget everything we learnt).
- Keep it or no-restart (NRS): we assume that the new situation is similar to the previous one and if a method was good in the past, it will be good in the future.

Penalization ($l_i(t)$):

- Active Penalization (AP): $Q = 0.9$ if a_i generates worse solutions than x_{best}, and $Q = 0$ otherwise. In this way, we lower the credit of a method if it is not allowing the improvement of the best found solution. The value $Q = 0.9$ was chosen after empirical observations.
- Inactive Penalization (IP): $Q = 0$ during the whole run.

Reward ($r_i(t)$):

- Better (RB): $r_i(t) = 1$ if the fitness of the generated solution (x_c) is strictly better than $f(x_{best})$.
- Equal or Better (REB): $r_i(t) = 1$ if the fitness of the generated solution (x_c) is equal or better than $f(x_{best})$.

We analyze the eight possible variants derived from the combination of the previous alternatives (shown in Table 2). As we stated above, every combination can be considered as a "learning scheme".

Table 2. Learning scheme variants

Variant	$w_i(t)$	$l_i(t)$	$r_i(t)$
RS-AP-RB	RS	AP	RB
RS-AP-REB	RS	AP	REB
RS-IP-RB	RS	IP	RB
RS-IP-REB	RS	IP	REB
NRS-AP-RB	NRS	AP	RB
NRS-AP-REB	NRS	AP	REB
NRS-IP-RB	NRS	IP	RB
NRS-IP-REB	NRS	IP	REB

4. Experimental Framework

We describe here the problems, performance measure, comparison techniques and the details of the experimentation we will follow to evaluate our proposal.

4.1. Problems

The portfolio will be tested on dynamic versions of static binary-coded problems constructed using the XOR-DOP generator [51]. The generator operates generating masks that are applied to the solution using a bitwise XOR operator. The objective function is changed every τ evaluations of the fitness function. In the k^{th} change, a new XOR mask $M(k)$ is generated as follows:

$$M(k) = M(k-1) \oplus T(k) \tag{4}$$

where $\oplus$ is the XOR operator ($a \oplus b = 1 \iff a \neq b$, and $a \oplus b = 0$ otherwise) and $T(k)$ is an intermediate binary mask randomly generated with $\rho \times m$ values set to 1 at change k (being m the dimension of the problem). When $k = 1$, $M(1) = \{0 \ldots 0\}$. The cost of a solution x at evaluation e is done as follows:

$$f(x,e) = f(x \oplus M(k)) \tag{5}$$

where $k = \lceil e/\tau \rceil$ is the current change and $\lceil \rceil$ is the integer part operator.

Figure 1 shows an example of the evaluation process. Through changes in the mask, the generator produces changes in the optimum position, and using different values for τ y ρ, we can control the frequency and severity of the changes, respectively. High values of ρ imply more severe changes, while low values of τ mean more frequent changes.

As stated before, using this idea of masking, any binary encoded static problem can be converted to its dynamic version. In what follows, the base static problems used here are described.

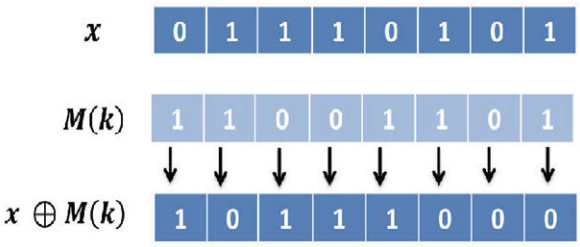

Fig. 1. Given the current mask $M(k)$ and a solution x, the objective function is applied to $f(x \oplus M(k))$, where $\oplus$ is the XOR (exclusive OR) operator.

4.1.1. *Knapsack problem*

The knapsack problem is a well-known NP-Hard combinatorial optimization problem [21]. Given a set of m elements the knapsack problem is described as follows:

$$\max f(x) = \sum_{i=1}^{m} p_i x_i \tag{6}$$

$$\text{subject to} \sum_{i=1}^{m} w_i x_i \leqslant C \quad x_i \in \{0,1\}\ i = 1 \ldots m \tag{7}$$

where $x = (x_1, \ldots, x_m)$ and $x_i = 1$ if object i is selected or $x_i = 0$, otherwise. Values p_i and w_i represent the profit and weight of object i, respectively, and C is the capacity of the knapsack. It is believed that knapsack is one of the easiest NP-Hard problems. Several exact algorithms are available and for them, the hardness of random instances, increases with the correlation between weights and profits [34].

Our test instance has $m = 100$ objects, and the weights, benefits and capacity are defined as:

$$w_i = U(1,50) \tag{8}$$

$$p_i = w_i + U(1,5) \tag{9}$$

$$C = 0.6 * \sum_{i=1}^{m} w_i \tag{10}$$

where $U(a,b)$ is a function returning a uniformly distributed random value in the $[a,b]$ interval. The definition for w_i y p_i led to an instance with strong correlation between both values. As stated in [34]:

The strongly correlated instances are hard to solve for two reasons: (a) The instances are ill-conditioned in the sense that there is a large gap between the continuous and integer solution of the problem; (b) Sorting the items according to decreasing efficiencies correspond to a sorting according to the weights. Thus, for any small interval of the ordered items (i.e. a "core") there is a limited variation in the weights, making it difficult to satisfy the capacity constraint with equality.

Possible unfeasible solutions arising in the search are penalized as in [51]:

$$f(x) = \begin{cases} \sum_{i=1}^{m} p_i x_i & \text{if } C' \leqslant C \\ 10^{-10} \cdot \left(\left(\sum_{i=1}^{m} w_i \right) - C' \right) & \text{otherwise} \end{cases} \tag{11}$$

being $C' = \sum_{i=1}^{m} w_i x_i$

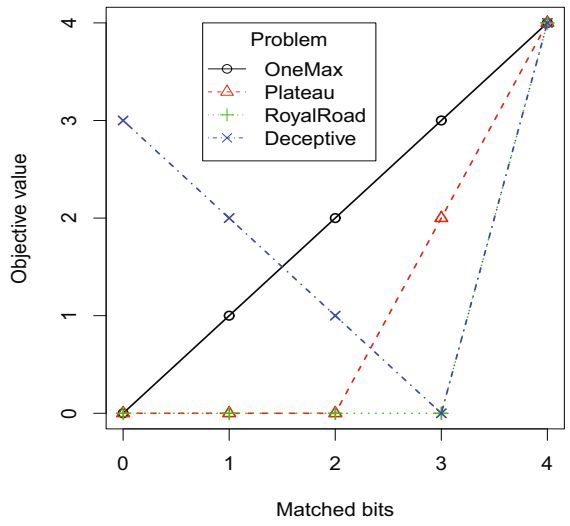

Fig. 2. Fitness contribution of every 4-bits block with respect to the number of correctly matched bits.

4.1.2. *Problems using base binary functions*

We use four additional problems and all of them consist on finding solutions that match all the bits of a target optimal solution. This target solution is initially considered to be the solution where all its bits are set to 1. To evaluate a solution, we consider blocks of 4 bits where each block contributes a given amount to the final objective value. The contribution of every block of 4 bits depends on the considered functions that are described below:

- **OneMax**: Each matched bit adds 1 to the fitness.
- **Plateau**: Three matched bits add 2 to the fitness while four matched bits add 4 and any other amount of bits matched leads to a 0 contribution.
- **RoyalRoad**: Each perfectly matched block adds 4 to the fitness. Partially matched blocks have fitness 0.
- **Deceptive**: Fitness is 4 if all the 4 bits are matched. If not, the fitness for the block is 3 minus the number of matched bits.

Figure 2 shows the contribution of every 4-bits block to every function in terms of the number of correctly matched bits.

4.1.3. *Additional information*

The dimension of all the problems was defined as 100 (25 blocks of 4-bits for functions described in Section 4.1.2). We considered five different change frequencies ($\tau \in \{1200, 3000, 6000, 9000, 12000\}$ fitness function evaluations) , and four different severities ($\rho \in \{0.1, 0.2, 0.5, 0.9\}$). The selected ($\tau$) should be understood as the number of fitness function evaluations allowed between consecutive changes, in other words, it accounts for the number of evaluations that the algorithm has in order to "catch" the optimum until the next movement.

We performed 30 independent runs for every algorithm, problem and combination of τ and ρ, where every run consisted of 100 changes of the fitness function.

4.2. *Reference algorithms*

To better assess the performance of our portfolio, we will perform comparisons against some reference algorithms. In a recent review [7], two algorithms are shown to be competitive in the DOPs considered here: Adaptive Hill Climbing Memetic Algorithm (AHMA) [46] and Self Organized Random Immigrants Genetic Algorithm (SORIGA) [42]. Although there exist more recent methods for the same problems used in this work [13, 44], AHMA and SORIGA are still used in comparisons [9, 25, 44], there is a general agreement on their quality, they are easy to understand and more important, their source code is available, thus running them on our benchmarks is much easier. We consider that in this way the comparison is fairer than taking values from published tables.

4.2.1. *Adaptive Hill Climbing Memetic Algorithm (AHMA)*

AHMA, firstly proposed in [46], is essentially a genetic algorithm coupled with a local search that has two neighborhood operators available: Greedy Crossover Hill Climbing (GCHC) and Steepest Mutation Hill Climbing (SMHC). GCHC applies a crossover operator using the elite solution and an individual from the population (chosen by the roulette

wheel selection). It returns the best child obtained. SMHC operator flips a number of bits in the elite solution (a sort of macro-mutation). The original solution is replaced if the new one is better.

These operators are selected with a probability distribution that is adjusted during the run. After a change, the probability values are kept. AHMA also includes two mechanisms to manage population diversity during the run: Adaptable Dual Mapping (ADM) and Triggered Random Immigrants (TRI).

4.2.2. SORIGA

SORIGA (*Self Organized Random Immigrants Genetic Algorithm*), proposed in [42], is a Genetic Algorithm in which, after the initialization, its population is split into two sub-populations: the main and the secondary one. The best individuals are moved to the main subpopulation whereas the worst individuals are replaced by randomly generated ones ("Random Immigrants") and moved to the secondary subpopulation. Both sub-populations are co-evolved independently until the worst individual considering the two sub-populations belongs to the main one. In this moment, both sub-populations are joined again, evolved for one generation and split in the same way explained before. This process is repeated iteratively until the stopping condition is reached.

4.2.3. Porfolio Composition

To implement the algorithms that compose the portfolio we use the Java library BiCIAM [12] which contains standard versions of the most common metaheuristics. For the evaluation of the algorithm portfolio we selected the next methods: Best-First Hill Climbing, Random Search, Simulated Annealing, Tabu Search, Genetic Algorithm, Evolutionary Strategy and Univariate Marginal Distribution Algorithm. It is important to note that we chose the methods basing just on theirs features (trajectory/population based, search pattern, evolutionary/non-evolutionary method, etc.) and without knowing a priori their performance in the DOPs showed above. We aimed at having a diverse set of algorithm with heterogeneous searching behaviors. Table 3 displays the parameter settings of the methods that integrate the algorithm portfolio. The current version of the Java library BiCIAM, which includes the metaheuristics mentioned above and the implementation of the algorithm portfolio presented in this paper, is available in the next link: `http://modo.ugr.es/algorithmportfolio/index.html`

Table 3. Methods that integrate the algorithm portfolio and their parameter settings

Method	Parameter	Setting
Hill Climbing	hill climbing type	Best-first
	neighborhood operator	one-bit flip mutation
Simulated Annealing	initial temperature t_0	20
	final temperature t_n	0
	number of iterations T	50
	α	0.93
	annealing scheme	$t_n = \alpha * t_0$
	neighborhood operator	one-bit flip mutation
Tabu Search	tabu list size	20
	tabu list content	solutions
	neighborhood operator	one-bit flip mutation
Evolutionary Strategy	population size	50
	mutation probability	0.9
	selection operator	truncation (20)
	mutation operator	uniform
Genetic Algorithm	population size	50
	mutation probability	0.5
	crossover probability	0.9
	selection operator	truncation (20)
	crossover operator	uniform
	mutation operator	uniform
Estimation of Distribution UMDA	population size	50
	selection operator	truncation (20)
	probability distribution	UMDA [36]

4.3. Performance measure

The algorithms will be evaluated using the "offline performance" [4] which is defined as follows:

$$F_{BG} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{G}\sum_{e=1}^{G}F_{B_{ie}^k}\right) \tag{12}$$

where N is the number of runs; in the original definition, G stands for the number of generations (which has perfect sense when talking about evolutionary

algorithms) but here, $G = 100 \cdot \tau$ is the number of fitness function evaluations allowed for the portfolio in one run; the value $k = \lceil e/\tau \rceil$, $k \in \{0 \dots 100\}$ is the current change period; and $F^k_{B_{ie}}$ is the fitness of the best solution found in the k-th change period of the i-th run up to the e-th evaluation (or $(e-(k-1)\tau)$-th evaluation of the k-th change period).

4.4. Statistical assessment of results

Nowadays, it is widely assumed that any comparison among a set of algorithms over a set of problems should be supported by statistical testing. In this article we follow the guidelines proposed in [15] where non-parametric statistical testing is suggested in situations like the one faced in this contribution (several problems, algorithms and configurations).

Firstly, we will apply Friedman's test to check if significant differences exist among a set of algorithms. Besides this, Friedman's average rank allows to sort the algorithms in terms of performance. Secondly, if such significant differences are detected, we use Wilcoxon's test for pairwise comparisons of algorithms, and Holm's and Finner's post-hoc tests for one-to-many comparisons between the best algorithm (as indicated by Friedman's rank output) and the rest.

Commercial software SPSS was used for Friedman's and Wilcoxon's tests and KEEL tool [15] for Holm's and Finner's post-hoc tests.

5. Results

The computational experiments are oriented to analyze the following questions:

- Does the portfolio obtain better results when using a learning scheme than when not? Which variant of the learning scheme leads to better results? How does the learning scheme influence the selection of the algorithms?
- Does the best portfolio learning variant obtain better results than the individual components when they are run isolated?
- How the performance of the algorithm portfolio is with respect to good reference algorithms (i.e. SORIGA and AHMA)?

Every algorithm in this contribution is analyzed over 5 problems (One-Max, Plateau, Royal-Road, Deceptive and Knapsack), 4 levels of severity (ρ) and 5 different change frequency (τ), that is, it is tested over a total of 100 problem configurations. For the sake of simplicity and understanding, the offline performance obtained by the different methods, in each problem configuration considered in this experimentation, is not displayed during the analysis of the results. The interested reader can refer to Appendix 1 to check this information.

5.1. Analysis of the learning scheme

In this subsection we will analyze the learning schemes presented in Section 3.1 to check if their use makes sense (if they lead to better results than an algorithm portfolio without learning scheme), which of them obtains the best performance and how the learning scheme influences the selection of the algorithms.

5.1.1. Learning vs No Learning

The aim of this first analysis is to verify whether or not, the learning schemes proposed lead the portfolio to obtain better results than a strategy without learning (AP-NoLearn), i.e. a portfolio where the constituent methods have the same selection probability along the whole run.

In first place we compare, using Wilcoxon's test at a significance level $\alpha = 0.05$, every learning scheme against AP-NoLearn and the results are shown in Table 4. The first column indicates the learning scheme under consideration. The second column "Global" states the result of the comparison over all the problems and configurations. The rest of the columns correspond to the results in every problem. The '$>$' sign states that the considered scheme is statistically better than AP-NoLearn, '$<$' means the opposite, and '$-$' indicates the difference in performance is not significant.

Considering the "Global" column, just half of the learning schemes tested lead to significantly better performance against AP-NoLearn, thus making relevant the need of a careful design of the learning strategy. In other words, a bad learning strategy may lead

Table 4. Learning vs. no learning. Results of the pairwise comparisons, using the Wilcoxon's non-parametric test at a significance level of $\alpha = 0.05$, between the learning schemes proposed and the variant of the algorithm portfolio without learning scheme (AP-NoLearn). The first column indicates the credit assignment scheme under consideration. The second column states the result of the comparison over all the problems and configurations. The rest of the columns correspond to the results in every problem. The '$>$' sign states that the considered scheme is statistically better than AP-NoLearn, '$<$' means the opposite, and '$-$' indicates that the difference in performance is not significant.

Learning scheme	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
RS-AP-RB	$>$	$>$	$>$	$>$	$>$	$>$
RS-AP-REB	$-$	$>$	$>$	$>$	$-$	$-$
RS-IP-RB	$>$	$>$	$>$	$>$	$-$	$>$
RS-IP-REB	$<$	$-$	$-$	$<$	$<$	$>$
NRS-AP-RB	$<$	$<$	$<$	$-$	$-$	$-$
NRS-AP-REB	$>$	$>$	$>$	$>$	$-$	$<$
NRS-IP-RB	$>$	$>$	$>$	$>$	$-$	$>$
NRS-IP-REB	$<$	$<$	$<$	$<$	$<$	$>$

to worse results than having no learning. It is not clear which component of the learning scheme have a higher impact on the performance, although those schemes using 'RB' (giving reward when the generated solution is strictly better than the reference one) provide better results.

Considering the results disaggregated by problem, the cases for OneMax, Plateau and RoyalRoad are quite similar to those in the Global case. Besides, the signs in the table are almost equal among the three problems, probably meaning that the portfolio behaves similarly on them. Deceptive is perhaps the most complex problem and its intrinsic structure "confuses" the learning scheme. Only RS-AP-RB is able to obtain better results than AP-NoLearn. When considering the Knapsack problem, perhaps the one closer to real life problems, the learning feature starts to be very relevant. Just in one case out of eight (NRS-AP-REB) the use of learning returned worst results than no learning. The other seven cases allowed obtaining better or equal results than AP-NoLearn. It has to be noted that RS-AP-RB is the only variant that consistently outperforms AP-NoLearn over all the cases considered.

5.1.2. Analysis of learning schemes

We will compare here the different learning schemes proposed to determine what it is the best one. We have analyzed the best scheme both globally (over all problems) and on each specific problem. Firstly, and using the Friedman test, we compared all learning strategies over all problems and on each problem separately. In all cases the test returned a $p - value = 0$, thus indicating that there are significant differences among the portfolios when using different learning schemes.

As we stated before, Friedman's test also outputs a mean ranking for each compared method. These results are shown in Figure 3 where each series represents the mean raking for a specific learning scheme on each of the cases mentioned before (Global, OneMax, Plateau, RoyalRoad, Deceptive and Knapsack). In order to assess whether the best learning strategy on each case has a performance significantly different to the others, we applied the Holm's and Finner's post-hoc tests at a significance level of $\alpha = 0.05$. Table 5 shows the results of these tests, where the symbol '*' indicates the learning scheme considered as control method (the method with the best mean ranking according to Friedman's test), whereas '$>$' and '-' indicates the existence or not, respectively, of significant differences between the control method and the corresponding learning scheme. In case both tests do not provide the same result (e.g., Holm's test does not rejects the hull hypothesis and Finner's does), the result of the Finner's post-hoc test is displayed within parenthesis.

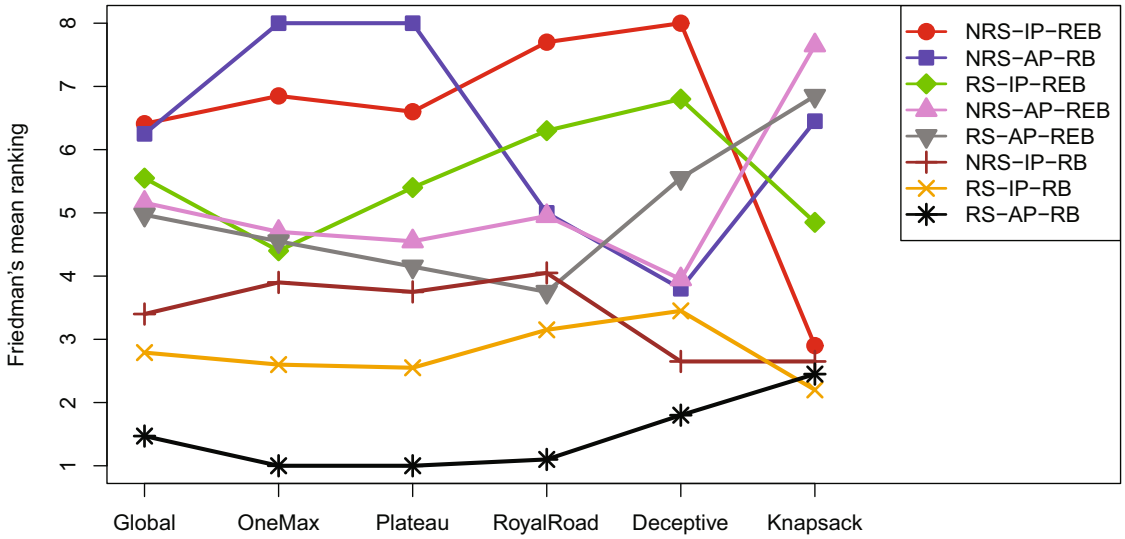

Fig. 3. Mean ranking provided by the Friedman's non-parametric test when all learning schemes are compared. Each series represents the mean raking (Y axis) for a specific learning scheme when all problems are considered (Global) and for each problem (OneMax, Plateau, RoyalRoad, Deceptive and Knapsack).

Figure 3 shows that the best ranked scheme globally is RS-AP-RB (credit is restarted after a change, penalization is active and credit is gained if the new solution is strictly better than the best one available). Furthermore, the difference in performance with respect to all the other schemes is significant (Table 5). Therefore, RS-AP-RB is statistically the best learning scheme over all problems.

If we consider each problem separately, RS-AP-RB is the best portfolio variant in four out of the five problems, that is, all but the Knapsack problem. Looking at Table 5, it improves significantly the rest of methods in three of these four problems (OneMax, Plateu and RoyalRoad), whereas in Deceptive the null hypothesis cannot be rejected for two learning schemes, RS-AP-RB and NRS-IP-REB (or only NRS-IP-REB if we consider the Finner's post-hoc test). In the Knapsack problem, despite not being the best learning scheme, RS-AP-RB is not significantly worse than RS-IP-RB, the control method in this case. In short, the performance of RS-AP-RB is better or similar to the other learning schemes, so we can conclude that it is the best portfolio variant.

5.1.3. *Analyzing the influence of the learning scheme in the selection of the algorithms*

So far, we have focused on the performance of the different learning schemes. In this section we aim at studying how the learning scheme influences the behavior of the algorithm portfolio. More specifically, we want to analyze how the selection of the algorithms varies from one learning scheme to another by studying the evolution of the selection probabilities $P_{selec}^{a_i}$ of the algorithms that compose the portfolio: Hill Climbing (HC), Random Search (RndS), Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA), Evolutionary Strategy (ES) and Estimation of Distribution Algorithm (EDA). For the sake of the simplicity and space, we have limited our analysis to two representative learning schemes and problem configurations. Concretely, we have chosen the best learning scheme, RS-AP-RB, and its non-restarting counterpart, NRS-AP-RB, in order to have a clear view of the effects of restarting the credit after each change. Regarding the problem configurations, we considered the two

Table 5. Results returned by the Holm's and Finner's post-hoc tests at a significance level of $\alpha = 0.05$ when the best learning scheme globally (over all problems) and on each specific problem is compared against the others. The symbol '*' indicates the learning scheme considered as control method (the method with the best mean ranking); '$>$' means that the control method improves significantly the corresponding learning scheme; and '-' indicates no significant differences between the two learning schemes. In case both tests do not provide the same result (e.g., Holm's test does not rejects the hull hypothesis and Finner's does), Finner's post-hoc test result is displayed within parenthesis.

Learning scheme	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
RS-AP-RB	$*$	$*$	$*$	$*$	$*$	$-$
RS-AP-REB	$>$	$>$	$>$	$>$	$>$	$>$
RS-IP-RB	$>$	$>$	$>$	$>$	$-(>)$	$*$
RS-IP-REB	$>$	$>$	$>$	$>$	$>$	$>$
NRS-AP-RB	$>$	$>$	$>$	$>$	$>$	$>$
NRS-AP-REB	$>$	$>$	$>$	$>$	$>$	$>$
NRS-IP-RB	$>$	$>$	$>$	$>$	$-$	$-$
NRS-IP-REB	$>$	$>$	$>$	$>$	$>$	$-$

most representative problems from a practical point of view, RoyalRoad and Knapsack, with intermediate values for severity and frequency of change, concretely, 0.5 and 6000, respectively.

Figure 4 displays the evolution of the mean selection probability for each algorithm, measured every 600 evaluations and aggregated over 30 runs, in the first five changes of the objective function. Rows and columns corresponds to learning schemes (RS-AP-RB and NRS-AP-RB) and problem configurations (RoyalRoad and Knapsack), respectively. In each plot, the horizontal line marks the probability value for a uniform distribution where all individual algorithms have the same selection probability, whereas the vertical lines show when the changes take place.

First, we focus our analysis on the Royal Road configuration. For a better understanding of the analysis, it is important to highlight that in this problem configuration, RS-AP-RB is significantly better than NRS-AP-RB (Mann-Whitney's U non-parametric test $\alpha < 0.05$) and the performance of the isolated algorithms in descent order (better $\rightarrow$ worse) according to their offline performance is HC, ES, GA, SA, RS, TS and EDA. Looking at the plot we can clearly observe the differences between restarting or not the credit after changes. In the NRS-AP-RB scheme, the probabilities converge after the first change to a virtually uniform distribution, where all the algorithms have the same chances of being selected. However, in the RS-AP-RB scheme, the probabilities vary right after each change and converge approximately after 3000 evaluations, that is, at half of the period between changes. Focusing on the individual methods, in the period before the first change, HC presents the highest probability for both credit assignment schemes. In the next stages, we observe very interesting behaviors in RS-AP-RB. The selection probability for HC becomes closer to the uniform distribution value but experiences abrupt changes for ES, SA and EDA, with high values at similar moments of the stationary periods of the function. Taking into account that SA and EDA are not among the best performing methods for this problem configuration when run individually, this behavior shows that although a solver may not have a good isolate performance, it can be very useful in some moments of the search when it is combined with other methods. This also explains why "forgetting" the learning gained after each change (by restarting the credit) is beneficial in this case. It allows taking advantage of those algorithms that only have a good performance in specific parts of the search but bad in the others.

The evolution of the selection probabilities for the Knapsack configuration is different but it keeps some similarities. In this case, RS-AP-RB is also significantly better than NRS-AP-RB (Mann-Whitney's U non-parametric test $\alpha < 0.05$), and the performance of the algorithms in descent order according to their offline performance is SA, HC, RS, TS, GA, EDA y ES. The first issue to highlight here

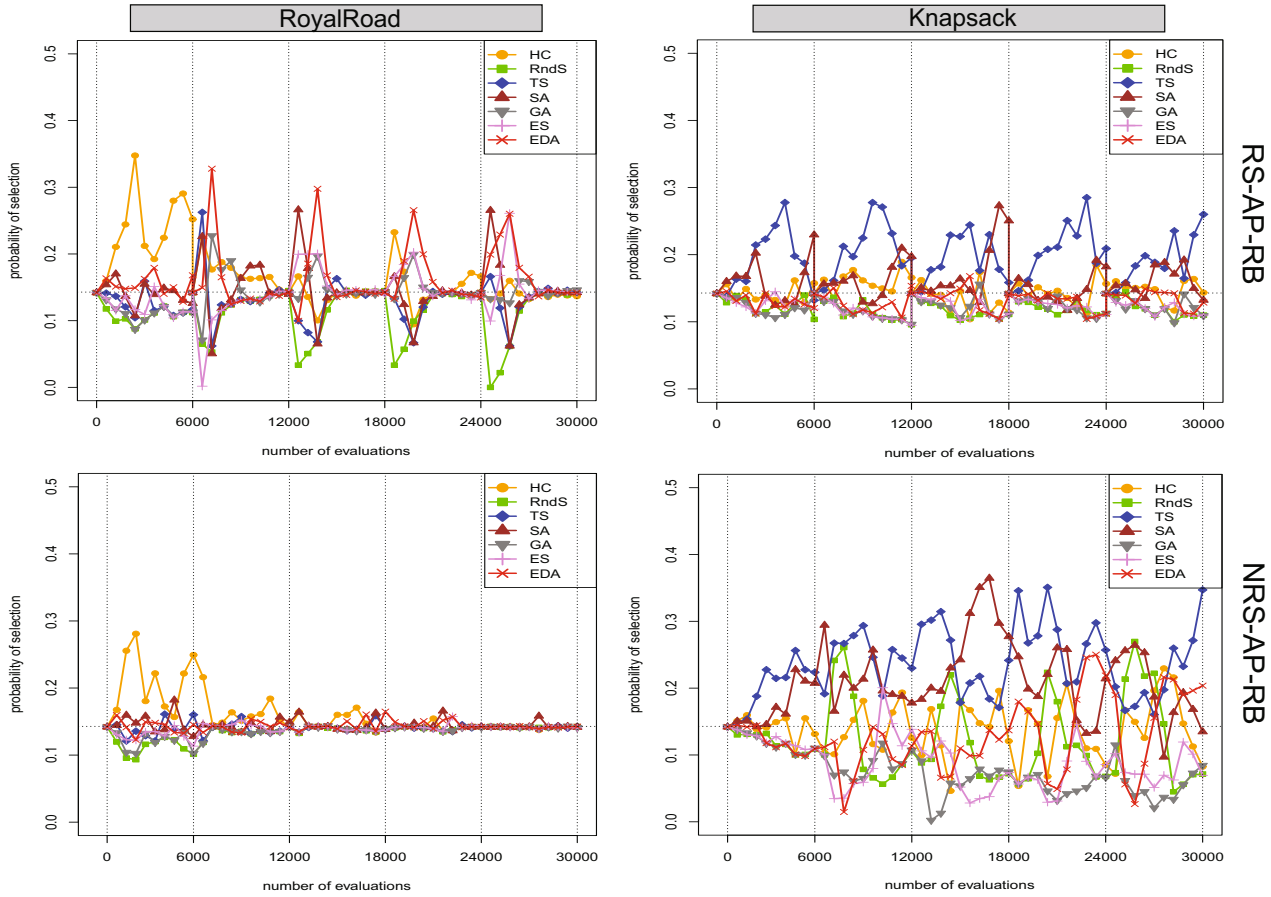

Fig. 4. Evolution of the mean selection probability for each individual algorithm, measured every 600 evaluations and aggregated over 30 runs, in the first five changes of the objective function. Rows and columns corresponds to learning schemes (RS-AP-RB and NRS-AP-RB) and problem configurations ([RoyalRoad (severity 0.5, frequency of change 6000) and Knapsack (severity 0.5, frequency of change 6000)), respectively. In each plot, the horizontal line marks the probability value for a uniform distribution where all individual algorithms have the same selection probability, whereas the vertical lines show when the changes take place.

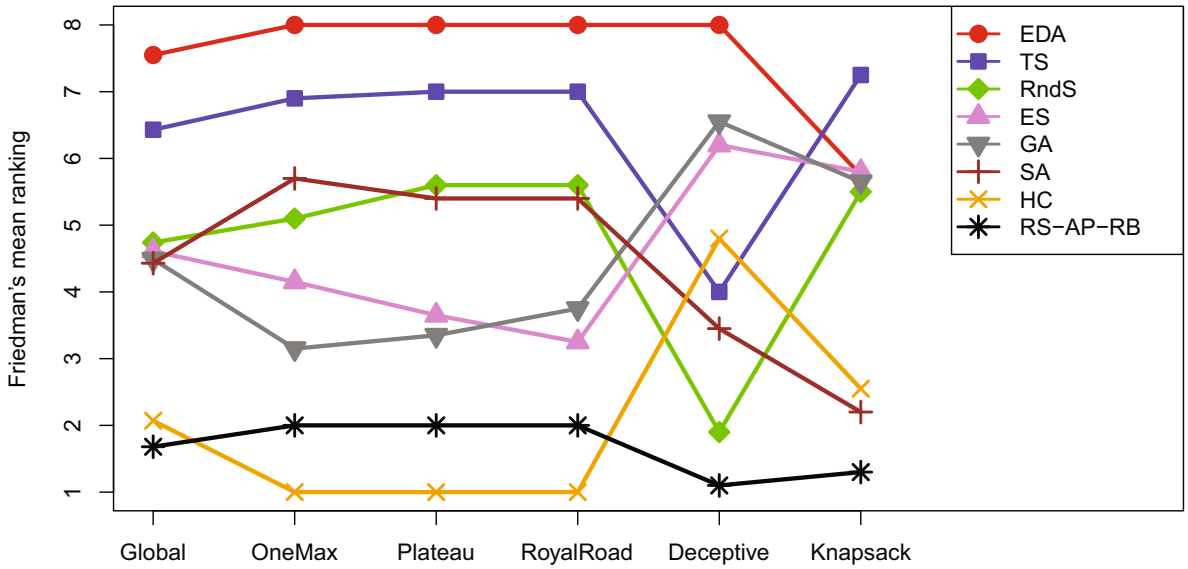

Fig. 5. Mean ranking provided by the Friedman's non-parametric test when the best portfolio variant (RS-AP-RB) is compared against the individual metaheuristics. Each series represent the mean raking (Y axis) for a method when all problems are considered (Global) and for each problem (OneMax, Plateau, RoyalRoad, Deceptive and Knapsack).

is the higher variation of the selection probabilities along the whole stationary periods, unlike in Royal Road where probabilities tended to converged. This is due to the greater difficulty of the Knapsack problem, which slows the convergence of the methods and spread the improvements, and thus the credit rewards, along the whole search process. Analyzing the individual methods, we observe that TS and SA are the algorithms that present a higher selection probability for both learning schemes. Again, it is interesting to see that TS is one of the two algorithms with the highest selection probability despite being the fourth best performing method when it is run individually. This suggests again that bad individual methods could be useful in some part of the search when combined with other algorithms.

5.2. Comparisons with isolated methods

In last section we determined the best learning scheme (RS-AP-RB). Here, we will check if the portfolio is able to obtain better results than their individual metaheuristics separately. To this end, we compare RS-AP-RB against the methods that compose the portfolio.

We will follow the same methodology employed in the former section. Figure 5 displays the mean ranking returned by the Friedman's non-parametric test for all methods both globally and on each problem. The $p-value$ obtained in all cases was equal to 0 so we can reject the null hypothesis. We also applied Holm's and Finner's post-hoc tests, at a significance level of $\alpha = 0.05$, in order to check if the difference in performance between the best method against the remaining algorithms is significant or not. These results are shown in Table 6 (the notation is the same explained above).

Globally over all problems, RS-AP-RB improves significantly all the isolated methods except HC (see Figure 5 and Table 6). Analyzing the problems separately, the first issue to highlight is the performance variability of the isolated metaheuristics, especially when the difficulty of the problems increases. Three good examples are HC, RndS and SA, the only

Table 6. Results returned by the Holm's and Finner's post-hoc tests at a significance level of $\alpha = 0.05$ when the best learning scheme (RS-AP-RB) and the individual metaheuristics are compared globally and on each specific problem. The symbol '*' indicates the learning scheme considered as control method (the method with the best mean ranking); '>' means that the control method improves significantly the corresponding learning scheme; and '-' indicates no significant differences between the two learning schemes. In case both tests do not provide the same result (e.g., Holm's test does not rejects the hull hypothesis and Finner's does), Finner's post-hoc test result is displayed within parenthesis.

Method	Global	OneMax	Plateau	RoyalRoad	Deceptive	Knapsack
HC	−	*	*	*	>	−
RndS	>	>	>	>	−	>
SA	>	>	>	>	>	−
TS	>	>	>	>	>	>
GA	>	>	>	>	>	>
ES	>	>	>	>	>	>
EDA	>	>	>	>	>	>
RS-AP-RB	*	−	−	−	*	*

methods that are not significantly worse than RS-AP-RB in all problems. HC and RndS have an opposite performance. While HC offers good results for OneMax, Plateau, RoyalRoad and Knapsack, and poor for Deceptive, RndS performs very well in Deceptive and poorly in the other four problems. As for SA, it shows a high performance for Knapsack and Deceptive but low for OneMax, Plateu and RoyalRoad. On the contrary, the portfolio presents very robust results along all problems. Particularly for the two hardest ones (Knapsack and Deceptive) where it improves all the individual metaheuristics in terms of mean ranking, although the null hypothesis cannot be rejected for RndS in Deceptive, and for HC and SA in Knapsack. In the end, although HC has a better performance than RS-AP-RB in OneMax, Plateau and RoyalRoad, we can affirm that the best variant of the portfolio obtains similar or significantly better results (in at least one problem) than the isolated versions of the metaheuristics that integrate it.

5.3. Comparisons against SORIGA and AHMA

To finish the analysis of the results, we will compare RS-AP-RB against SORIGA and AHMA (both described in Section 4.2), to check whether its results are competitive with high-performance algorithms for these problems. We used the same benchmarks as before and the parameter setting of both methods was done according to the guidelines given in their original works ([42] and [46], respectively).

We will follow the same comparison scheme used in the two former sections, that is, we will use the mean ranking provided by the Friedman's non-parametric test, which rejects the null hypothesis in all cases ($p - value = 0$), and the Holm's and Finner's post-hoc tests at a confidence level of $\alpha = 0.95$ to check the significance of the difference in performance among the three methods. These results are displayed in Figure 6 and Table 7, respectively.

When considering all the problems and configurations (Global), the portfolio coupled with the RS-AP-RB learning scheme achieves significantly better performance than AHMA and SORIGA. Separating the results by problem, we observe that RS-AP-RB is the best alternative in OneMax, Plateau and RoyalRoad, as shown in Figure 6. Furthermore, the null hypothesis of similar performance can be rejected for both AHMA and SORIGA in the three cases (Table 7). For Deceptive, AHMA is significantly the best method, whereas for Knapsack, AHMA and RS-AP-RB achieved similar performance. Overall, (and setting apart Deceptive, maybe the one most distant to real problems), it becomes clear that RS-AP-RB is better or equal than the reference algorithms considered.

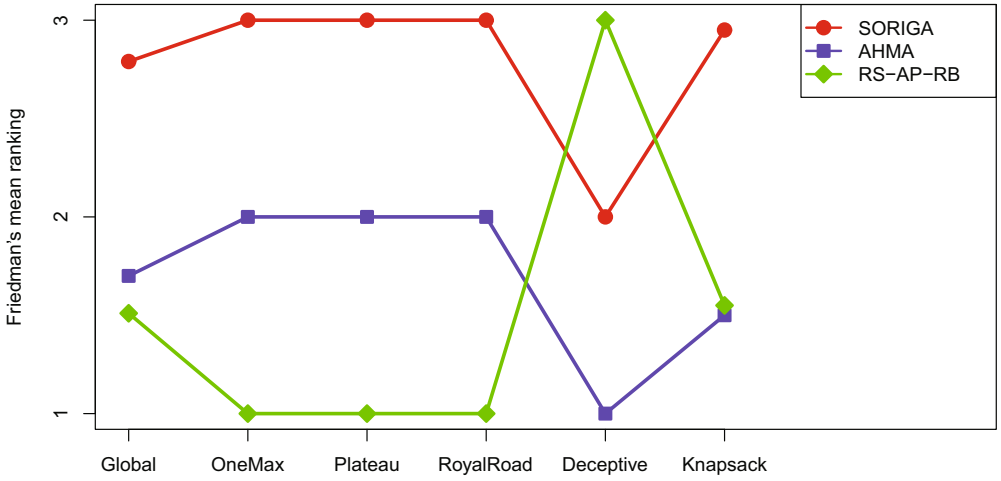

Fig. 6. Mean ranking provided by the Friedman's non-parametric test when the best portfolio variant (RS-AP-RB) is compared against SORIGA and AHMA. Each series represent the mean raking (Y axis) for a specific method when all problems are considered (Global) and for each problem (OneMax, Plateau, RoyalRoad, Deceptive and Knapsack).

Table 7. Results returned by the Holm's and Finner's post-hoc tests at a significance level of $\alpha = 0.05$ when the best learning scheme (RS-AP-RB), SORIGA and AHMA are compared globally and on each specific problem. The symbol '*' indicates the learning scheme considered as control method (the method with the best mean ranking); '$>$' means that the control method improves significantly the corresponding learning scheme; and '-' indicates no significant differences between the two learning schemes. In case both tests do not provide the same result (e.g., Holm's test does not rejects the hull hypothesis and Finner's does), Finner's post-hoc test result is displayed within parenthesis.

Method	**Global**	**OneMax**	**Plateau**	**RoyalRoad**	**Deceptive**	**Knapsack**
RS-AP-RB	*	*	*	*	$>$	$-$
AHMA	$>$	$>$	$>$	$>$	*	*
SORIGA	$>$	$>$	$>$	$>$	$>$	$>$

6. Conclusions

In this work we have presented an Algorithm Portfolio to solve combinatorial Dynamic Optimization Problems. This method consists of a set of metaheuristics that are run iteratively. At each stage of the search, the portfolio selects which metaheuristic to apply using a credit based approach that acts as a learning scheme.

The algorithm portfolio was tested over five test problems (OneMax, Plateau, RoyalRoad, Deceptive and Knapsack) to which we induced dynamism by means of XOR-DOP generator. For each problem we considered 4 and 5 different severities and frequencies of change, respectively. To compare the methods, we employed the offline performance, as performance measure, and paired non-parametric tests to check the significance of the differences among algorithms.

The experimentation was oriented to check: whether the algorithm portfolio obtained better results when using a learning scheme than when not; what learning scheme provided better results; how the learning scheme influenced the selection of algorithms; if the portfolio with the best learning scheme improved the performance of the individual metaheuristics that compose it; and how the performance of the portfolio was with respect to two reference methods in the literature, AHMA and SORIGA.

After analyzing the results of the experimentation we can draw the next conclusions:

- The right design of the learning scheme for the algorithm portfolio is a crucial task in DOPs, since only 4 out of 8 learning schemes provided better result than the non-learning version of the algorithm portfolio.
- Different learning schemes lead to different patterns of algorithm selection.
- Algorithms with bad performance when run individually may be useful in specific moments of the search when combined with other methods.
- RS-AP-RB was the only learning scheme that consistently outperformed the non-learning version of the algorithm portfolio in the five problems considered.
- RS-AP-RB was significantly the best credit assignment scheme both globally and in the majority of the problems, except for Knapsack. We understand the lower performance of RS-AP-RB in knapsack problem, as an indication that the structure of the problem may influence the performance of the learning scheme, making some features more appropriate than others (e.g., penalization performs better for OneMax, Plateau, RoyalRoad and Deceptive, while worse for the Knapsack problem).
- When considering the results over all problems, RS-AP-RB improved all the individual metaheuristics that integrate it, with all the difference in performance significant but one, Hill Climbing. Hill Climbing is the best standalone method for OneMax, Plateau and RoyalRoad. As a consequence, RS-AP-RB selects this method with a higher frequency, making it, at the end, to reach a Hill Climbing-like behavior. The situation is different in Deceptive and Knapsack problems, where the differences in performance are lower among the individual methods. Thus RS-AP-RB shows a behavior that can be understood as a "hybrid" among its constituent methods, taking the "best" of each one.
- RS-AP-RB offered very robust results in all problems, in contrast to the performance variability of the individual metaheuristics.
- Over all problems, RS-AP-RB obtained significantly better results than AHMA and SORIGA.

In general terms, the results showed that the old idea of the portfolio of algorithms can also provide good results in DOPs. An important aspect to highlight it is the extreme simplicity of the learning scheme of the portfolio and the metaheuristics that integrate it. We have seen how very general and basic versions of common metaheuristics working together under a simple learning scheme can provide competitive results with respect to ad-hoc high-performance methods for DOPs. In our opinion, these results show that algorithm portfolios could be a good paradigm for designing solvers not only for static problems but also for dynamic ones and therefore, they deserve a greater attention in this field.

References

1. P. Angeline. Tracking extrema in dynamic environments. In P. Angeline, R. Reynolds, J. McDonnell, and R. Eberhart, editors, *Evolutionary Programming VI*, volume 1213 of *Lecture Notes in Computer Science*, pages 335–345. Springer Berlin Heidelberg, 1997.
2. T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.
3. J. Branke, M. Orbayi, and S. Uyar. The role of representations in dynamic knapsack problems. In F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. Moore, J. Romero, G. Smith, G. Squillero, and H. Takagi, editors, *Applications of Evolutionary Computing*, volume 3907 of *Lecture Notes in Computer Science*, pages 764–775. Springer Berlin Heidelberg, 2006.
4. J. Branke and H. Schmeck. Designing evolutionary algorithms for dynamic optimization problems. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing*, Natural Computing Series, pages 239–262. Springer Berlin Heidelberg, 2003.
5. E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.
6. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin Heidelberg, 2001.
7. C. Cruz, J. R. González, and D. A. Pelta. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448, 2011.
8. F. O. de França, F. J. Von Zuben, and L. N. de Castro. An artificial immune network for multimodal function optimization on dynamic environments. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 289–296, New York, NY, USA, 2005. ACM.
9. I. G. del Amo, D. A. Pelta, J. R. González, and A. D. Masegosa. An algorithm comparison for dynamic optimization problems. *Applied Soft Computing*, 12(10):3176 – 3192, 2012.
10. S. Droste. Analysis of the (1+1) EA for a dynamically bitwise changing OneMax. In E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation (GECCO 2003)*, volume 2723 of *Lecture Notes in Computer Science*, pages 909–921. Springer Berlin Heidelberg, 2003.
11. W. Du and B. Li. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, 178(15):3096 – 3109, 2008.
12. J. Fajardo and A. Rosete. Algoritmo multigenerador de soluciones, para la competencia y colaboración de generadores metaheurísticos. *Revista Internacional de Investigación de Operaciones*, 1:57–63, 2011.
13. C. Fernandes, J. Laredo, A. Rosa, and J. Merelo. The sandpile mutation genetic algorithm: an investigation on the working mechanisms of a diversity-oriented and self-organized mutation operator for non-stationary functions. *Applied Intelligence*, 39(2):279–306, 2013.
14. M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2006.
15. S. García, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009.
16. C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.
17. J. González, C. Cruz, I. Amo, and D. Pelta. An adaptive multiagent strategy for solving combinatorial dynamic optimization problems. In D. Pelta, N. Krasnogor, D. Dumitrescu, C. Chira, and R. Lung, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2011)*, volume 387 of *Studies in Computational Intelligence*, pages 41–55. Springer Berlin Heidelberg, 2011.
18. J. R. González, A. D. Masegosa, and I. G. del Amo. A cooperative strategy for solving dynamic optimization problems. *Memetic Computing*, 3:3–14, 2011.
19. M. Guntsch, M. Middendorf, and H. Schmeck. An ant colony optimization approach to dynamic TSP. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference, GECCO'01*, pages 860–867, San Francisco, CA, USA, 2001. Morgan Kaufmann.
20. B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.
21. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
22. B. Kiraz, A. S. Uyar, and E. Özcan. Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769, 2013.
23. C. Li, S. Yang, T. Nguyen, E. Yu, X. Yao, Y. Jin, H. Beyer, and P. Suganthan. Benchmark generator for CEC 2009 competition on dynamic optimization.

Technical report, 2008.
24. L. Liu, D. Wang, and W. Ip. A permutation-based dual genetic algorithm for dynamic optimization problems. *Soft Computing*, 13(7):725–738, 2009.
25. A. D. Masegosa, D. Pelta, and I. G. del Amo. The role of cardinality and neighborhood sampling strategy in agent-based cooperative strategies for dynamic optimization problems. *Applied Soft Computing*, 14, Part C(0):577 – 593, 2014.
26. D. C. Mattfeld and C. Bierwirth. An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research*, 155(3):616 – 630, 2004.
27. M. Mavrovouniotis and S. Yang. A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing*, 15(7):1405–1425, 2011.
28. R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Natural Computing Series. Springer Berlin Heidelberg, 2004.
29. R. Morrison and K. De Jong. A test problem generator for non-stationary environments. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, volume 3, pages 2047–2053, Washington D.C., USA, 1999. IEEE Press.
30. T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1 – 24, 2012.
31. M. Nikolić, F. Marić, and P. Janičić. Simple algorithm portfolio for SAT. *Artificial Intelligence Review*, 40(4):457–465, 2013.
32. F. Peng, K. Tang, G. Chen, and X. Yao. Population-based algorithm portfolios for numerical optimization. *IEEE Transactions on Evolutionary Computation*, 14(5):782–800, 2010.
33. M. Petrik and S. Zilberstein. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence*, 48(1-2):85–106, 2006.
34. D. Pisinger. Where are the hard knapsack problems? *Computers and Operations Research*, 32(9):2271–2284, 2005.
35. P. Rohlfshagen and X. Yao. The dynamic knapsack problem revisited: A new benchmark problem for dynamic combinatorial optimisation. In M. Giacobini, A. Brabazon, S. Cagnoni, G. Caro, A. Ekárt, A. Esparcia-Alcázar, M. Farooq, A. Fink, and P. Machado, editors, *Applications of Evolutionary Computing*, volume 5484 of *Lecture Notes in Computer Science*, pages 745–754. Springer Berlin Heidelberg, 2009.
36. R. Santana, P. Larrañaga, and J. A. Lozano. Adaptive estimation of distribution algorithms. In C. Cotta, M. Sevaux, and K. Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 177–197. Springer Berlin Heidelberg, 2008.
37. L. Schönemann. The impact of population sizes and diversity on the adaptability of evolution strategies in dynamic environments. In *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, volume 2, pages 1270–1277, Portland, OR, US, 2004. IEEE Press.
38. L. Schönemann. Evolution strategies in dynamic environments. In S. Yang, Y.-S. Ong, and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, pages 51–77. Springer Berlin Heidelberg, 2007.
39. N. Shukla, A. Choudhary, P. Prakash, K. Fernandes, and M. Tiwari. Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance. *International Journal of Production Economics*, 141(1):146 – 166, 2013.
40. N. Shukla, M. K. Tiwari, and D. Ceglarek. Genetic algorithms based algorithm portfolio for inventory routing problem with stochastic demand. *International Journal of Production Research*, 51(1):118–137, 2013.
41. S. A. Stanhope and J. M. Daida. (1+1) genetic algorithm fitness dynamics in a changing environment. In *Proceedings of the 1999 Congress of Evolutionary Computation, CEC 2009*, volume 3, pages 1851–1858, Washington D.C., USA, 1999.
42. R. Tinós and S. Yang. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. *Genetic Programming and Evolvable Machines*, 8(3):255–286, 2007.
43. H. R. Topcuoglu, A. Ucar, and L. Altin. A hyper-heuristic based framework for dynamic optimization problems. *Applied Soft Computing*, 19(0):236 – 251, 2014.
44. A. M. Turky and S. Abdullah. A multi-population harmony search algorithm with external archive for dynamic optimization problems. *Information Sciences*, 272(0):84 – 95, 2014.
45. G. Uludağ, B. Kiraz, A. Etaner-Uyar, and E. Özcan. A framework to hybridize PBIL and a hyper-heuristic for dynamic environments. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 358–367. Springer Berlin Heidelberg, 2012.
46. H. Wang, D. Wang, and S. Yang. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing*, 13(8-9):763–780, 2009.
47. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*,

32:565–606, 2008.
48. S. R. Yadav, R. R. M. Muddada, M. Tiwari, and R. Shankar. An algorithm portfolio based solution methodology to solve a supply chain optimization problem. *Expert Systems with Applications*, 36(4):8407–8420, 2009.
49. S. Yang, Y. Jiang, and T. T. Nguyen. Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, 24(4):451–480, 2013.
50. S. Yang, Y. Ong, and Y. Jin. *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*. Springer, 2007.
51. S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11):815–834, 2005.

Appendix A

Table A.1. Offline performance obtained in 30 runs by each learning schemes considered in the experimentation for each problem, severity and frequency of change.

		OneMax				Plateau				RoyalRoad				Deceptive				Knapsack			
		severity				severity				severity				severity				severity			
	change	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
RS-IP-REB	1200	96.697	93.627	88.183	88.009	92.951	85.863	73.776	73.967	75.169	59.801	44.084	40.929	18.201	17.282	16.710	16.893	1674.750	1668.979	1665.183	1674.889
	3000	98.650	97.330	94.934	94.932	96.861	93.046	86.141	86.149	83.193	71.834	56.033	51.294	20.123	19.249	18.599	18.599	1679.836	1674.244	1670.331	1679.611
	6000	99.326	98.674	97.472	97.461	98.274	95.740	91.089	91.197	87.647	77.650	63.426	58.637	21.726	20.413	19.931	19.707	1684.083	1677.994	1674.314	1683.923
	9000	99.540	99.117	98.311	98.300	98.789	96.696	92.894	92.975	89.458	80.805	67.657	63.931	22.679	21.388	20.829	20.702	1687.852	1680.461	1676.542	1687.469
	12000	99.662	99.339	98.731	98.724	99.023	97.227	93.892	94.069	90.199	81.972	70.500	67.099	23.729	22.160	21.690	21.353	1689.696	1682.967	1678.778	1689.932
RS-IP-RB	1200	97.363	94.564	87.864	87.637	94.402	87.784	71.548	71.467	77.185	60.925	43.184	40.244	19.714	18.985	18.339	18.346	1675.406	1669.828	1664.828	1675.449
	3000	98.942	97.811	95.086	95.024	97.736	94.774	86.920	86.983	87.960	75.308	56.465	52.667	22.491	21.438	20.995	20.737	1681.025	1675.253	1671.222	1681.000
	6000	99.473	98.907	97.564	97.510	98.841	97.207	92.955	93.053	92.340	82.820	66.147	63.261	24.206	23.468	22.815	22.562	1686.174	1680.274	1676.124	1687.073
	9000	99.649	99.271	98.370	98.328	99.203	98.074	95.131	95.151	94.125	86.425	71.761	70.069	25.510	24.476	23.928	23.675	1690.165	1682.660	1678.850	1689.953
	12000	99.737	99.455	98.778	98.752	99.395	98.541	96.299	96.277	94.902	88.690	75.223	74.054	26.851	25.399	24.765	24.664	1692.488	1685.421	1681.317	1692.558
RS-AP-REB	1200	97.321	94.353	87.360	87.149	94.250	87.546	71.518	71.416	77.925	61.276	43.467	40.495	19.762	18.644	17.992	17.596	1672.132	1667.715	1663.481	1672.086
	3000	98.924	97.731	94.904	94.810	97.624	94.602	86.541	86.573	87.756	75.211	56.438	51.641	22.002	21.181	20.400	20.214	1674.764	1670.086	1666.705	1675.079
	6000	99.462	98.865	97.441	97.404	98.792	97.109	92.653	92.698	92.218	82.934	66.019	62.806	23.953	22.899	22.351	22.221	1675.907	1671.159	1668.374	1675.161
	9000	99.640	99.238	98.296	98.273	99.177	97.968	94.724	94.865	94.030	86.067	70.856	68.555	25.454	24.058	23.503	23.210	1676.288	1671.747	1668.235	1676.354
	12000	99.728	99.429	98.721	98.706	99.353	98.417	95.928	95.922	94.946	88.001	74.934	72.949	26.417	24.717	24.434	24.067	1676.373	1671.604	1668.943	1676.417
RS-AP-RB	1200	99.532	99.266	99.617	99.544	98.739	95.148	86.702	85.823	92.048	92.452	84.869	82.448	20.195	18.998	18.431	18.367	1675.046	1669.866	1665.136	1675.534
	3000	99.007	98.096	96.215	98.167	98.839	95.425	89.887	89.925	89.058	77.949	61.739	79.613	22.407	21.855	21.060	20.933	1682.253	1675.598	1671.495	1681.525
	6000	99.507	99.047	98.105	98.091	98.878	97.591	94.667	94.770	92.914	85.540	73.682	73.129	24.589	23.488	23.006	22.973	1686.111	1680.050	1676.114	1685.967
	9000	99.670	99.363	98.740	98.725	99.228	98.314	96.345	96.384	94.455	88.680	79.680	79.797	25.917	25.024	24.265	24.001	1689.472	1682.749	1678.951	1689.532
	12000	99.750	99.523	99.056	99.042	99.429	98.720	97.231	97.194	95.329	90.588	93.004	93.013	27.272	25.686	25.304	24.927	1691.908	1684.759	1681.273	1691.020
NRS-IP-REB	1200	95.195	90.859	86.096	87.285	89.885	82.284	71.610	73.837	74.705	60.427	40.266	37.505	15.349	14.593	13.986	14.462	1675.527	1669.410	1664.945	1675.560
	3000	98.042	95.894	92.528	93.357	95.506	90.181	83.431	84.105	82.838	71.478	52.451	43.020	17.284	16.034	15.224	15.753	1681.742	1675.308	1671.148	1681.524
	6000	99.008	97.885	95.590	95.696	97.134	93.402	87.985	90.382	86.181	77.351	60.528	47.917	19.355	17.585	16.564	17.183	1686.731	1679.761	1675.553	1686.680
	9000	99.337	98.576	96.656	97.160	98.128	94.737	89.811	91.811	87.860	80.065	65.131	49.795	21.020	18.957	17.636	18.274	1689.568	1682.132	1678.363	1689.815
	12000	99.511	98.932	97.172	97.301	98.518	95.368	90.601	92.256	88.372	81.515	68.120	51.566	22.070	19.566	17.852	18.747	1692.428	1684.379	1680.528	1692.290
NRS-IP-RB	1200	97.496	94.488	86.434	86.146	94.653	87.671	69.833	69.350	78.607	60.814	43.116	40.024	20.012	19.111	18.463	18.476	1675.520	1669.685	1664.952	1675.860
	3000	98.972	97.791	94.466	94.332	97.785	94.686	85.306	85.377	88.181	75.075	55.933	51.808	22.428	21.499	21.073	20.852	1681.993	1675.298	1671.091	1681.263
	6000	99.486	98.891	97.210	97.182	98.860	97.179	92.118	92.074	92.402	83.072	65.364	61.482	24.632	23.363	22.801	22.625	1686.723	1679.341	1675.554	1686.996
	9000	99.659	99.259	98.147	98.120	99.228	98.062	94.443	94.403	94.110	86.124	70.421	68.465	25.682	24.400	24.133	23.792	1689.640	1682.575	1678.596	1689.953
	12000	99.745	99.449	98.607	98.590	99.379	98.481	95.600	95.649	94.944	88.181	74.229	72.275	26.574	25.345	24.723	24.686	1692.360	1684.905	1680.654	1692.137
NRS-AP-REB	1200	97.441	94.457	86.258	86.063	94.616	87.441	69.535	69.256	78.241	60.636	42.639	39.959	19.879	19.164	18.190	18.326	1659.158	1647.679	1630.299	1660.947
	3000	98.973	97.743	94.377	94.268	97.784	94.672	85.390	85.327	88.111	74.814	55.847	51.629	22.233	21.337	20.791	20.626	1671.098	1654.507	1641.605	1673.837
	6000	99.486	98.879	97.211	97.146	98.833	97.122	92.024	91.999	92.415	82.778	65.419	61.875	24.344	23.263	22.818	22.360	1676.858	1660.464	1648.235	1671.442
	9000	99.657	99.257	98.133	98.103	99.191	98.000	94.389	94.394	93.932	86.067	70.303	68.329	25.527	24.556	23.758	23.814	1678.867	1670.334	1657.083	1678.964
	12000	99.743	99.446	98.603	98.584	99.408	98.479	95.608	95.591	94.793	88.177	74.102	72.589	26.556	25.123	24.606	24.433	1678.505	1669.627	1637.374	1678.866
NRS-AP-RB	1200	89.827	86.918	73.941	76.055	81.079	73.119	51.658	53.481	69.108	60.407	54.910	51.738	17.710	16.308	16.101	15.808	1662.259	1656.879	1650.774	1661.837
	3000	85.416	81.313	81.423	79.265	75.669	64.629	62.475	61.077	76.467	64.109	50.036	50.735	21.173	19.783	18.875	19.068	1670.644	1666.472	1661.524	1671.387
	6000	97.360	89.784	89.206	90.113	94.805	87.656	80.108	79.627	88.007	78.186	64.692	63.529	24.386	23.318	22.556	22.506	1677.665	1674.624	1670.708	1679.441
	9000	98.261	96.288	93.037	93.369	96.596	92.771	86.053	86.608	91.951	83.735	71.930	72.178	26.382	25.237	24.296	24.495	1683.665	1678.657	1674.753	1684.315
	12000	98.700	97.442	95.130	95.303	97.428	94.571	88.966	90.208	93.423	87.068	77.265	77.509	27.652	26.108	25.800	25.670	1686.898	1682.265	1679.556	1686.938

Table A.2. Offline performance obtained in 30 runs by the isolated versions of algorithms that compose the portfolio for each problem, severity and frequency of change.

		OneMax				Plateau				RoyalRoad				Deceptive				Knapsack			
		severity				severity				severity				severity				severity			
	change	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
HC	1200	99.197	98.384	95.920	92.639	98.258	96.182	89.746	83.748	89.268	80.940	66.838	60.687	11.972	11.915	11.980	13.516	1676.305	1671.407	1665.839	1675.655
	3000	99.682	99.348	98.369	97.043	99.298	98.459	95.909	93.519	95.719	92.163	85.759	82.872	11.988	11.972	16.766	17.594	1677.620	1672.574	1668.479	1677.862
	6000	99.840	99.673	99.183	98.523	99.656	99.234	97.953	96.769	97.838	96.089	92.901	91.411	11.996	11.990	18.077	18.922	1678.150	1673.188	1668.759	1679.739
	9000	99.893	99.783	99.453	99.015	99.768	99.491	98.636	97.844	98.566	97.411	95.307	94.281	11.997	11.992	18.665	19.271	1678.829	1673.836	1669.775	1679.033
	12000	99.920	99.837	99.592	99.259	99.829	99.619	98.981	98.372	98.926	98.041	96.461	95.708	11.999	11.995	19.137	19.456	1679.537	1673.282	1669.751	1679.821
RS	1200	65.550	64.894	64.653	64.644	39.346	38.548	38.245	38.138	24.437	24.000	23.727	23.767	18.818	18.652	18.622	18.610	1657.082	1656.955	1656.826	1656.963
	3000	66.721	66.219	65.969	65.915	41.268	40.469	40.118	40.183	26.914	26.475	26.415	26.413	19.598	19.559	19.519	19.497	1663.341	1663.146	1663.080	1663.413
	6000	67.521	67.052	66.919	66.874	42.285	41.784	41.551	41.618	27.889	27.630	27.553	27.496	19.976	19.933	19.917	19.910	1667.171	1666.993	1667.124	1667.158
	9000	67.998	67.614	67.467	67.482	43.038	42.582	42.492	42.430	28.397	28.066	27.939	27.978	21.872	21.490	21.574	21.573	1669.207	1669.229	1669.017	1669.082
	12000	68.340	67.948	67.844	67.790	43.645	43.104	43.062	43.055	29.526	29.162	29.075	29.063	23.233	23.210	23.229	23.021	1670.454	1670.362	1670.198	1670.487
TS	1200	60.265	60.183	60.115	60.193	32.217	32.151	32.168	32.149	18.503	18.537	18.528	18.537	13.490	13.578	13.217	13.469	1636.800	1636.055	1636.392	1636.426
	3000	62.245	62.365	62.213	62.273	35.168	35.219	35.153	35.154	21.287	21.459	21.283	21.305	17.035	16.904	16.990	16.921	1652.397	1652.123	1652.610	1652.826
	6000	63.770	63.690	63.727	63.708	37.205	37.125	37.131	37.252	23.180	23.288	23.174	23.209	18.344	18.347	18.371	18.348	1660.527	1660.413	1660.311	1660.419
	9000	64.481	64.473	64.564	64.546	38.274	38.352	38.126	38.254	24.129	24.000	24.275	24.110	18.830	18.852	18.868	18.871	1663.741	1664.157	1663.850	1663.749
	12000	65.027	65.013	65.054	65.026	39.143	39.101	39.105	39.043	25.207	25.186	25.370	25.161	19.257	19.194	19.223	19.232	1666.039	1666.021	1666.035	1666.191
SA	1200	62.523	62.610	62.322	62.088	36.643	36.493	36.154	35.933	22.105	21.875	21.890	21.650	13.051	12.865	12.857	12.782	1669.539	1668.425	1666.993	1669.549
	3000	64.930	64.937	64.810	64.634	40.244	40.158	40.034	39.757	25.947	25.917	25.698	25.610	17.069	17.074	17.122	16.818	1674.405	1673.263	1673.662	1674.654
	6000	66.615	66.581	66.627	66.554	42.911	42.938	42.775	42.768	28.652	28.683	28.343	28.352	18.615	18.584	18.684	18.539	1677.910	1676.437	1677.128	1678.012
	9000	67.532	67.480	67.480	67.423	44.438	44.394	44.227	44.257	30.340	30.245	30.233	30.081	19.247	19.240	19.273	19.219	1680.058	1678.683	1678.785	1679.600
	12000	68.189	68.087	68.016	67.969	45.469	45.418	45.232	45.269	31.251	31.259	31.109	31.153	19.641	19.587	19.591	19.617	1681.037	1679.657	1679.558	1680.794
GA	1200	96.216	91.330	74.958	58.341	91.788	80.084	50.348	44.777	64.647	43.782	24.990	33.893	11.803	11.000	10.701	11.000	1662.625	1649.844	1614.986	1662.139
	3000	98.518	96.503	88.401	74.220	96.815	90.894	67.155	49.101	75.416	54.964	31.542	38.823	11.998	11.760	11.091	11.801	1671.867	1661.739	1642.618	1672.269
	6000	99.262	98.264	94.202	86.971	98.416	94.917	75.969	51.863	79.464	60.332	35.031	41.308	12.000	11.936	11.664	11.946	1676.324	1666.720	1652.077	1676.947
	9000	99.511	98.844	96.142	91.297	98.960	96.482	79.285	53.469	82.451	62.888	37.404	42.410	12.000	11.947	11.790	11.973	1677.934	1669.617	1656.146	1677.686
	12000	99.631	99.132	97.098	93.479	99.216	97.615	81.024	54.213	82.912	64.791	38.573	42.848	12.000	11.970	11.851	11.982	1678.553	1669.102	1657.887	1678.778
EE	1200	95.793	88.685	70.493	56.554	91.008	75.659	46.331	42.389	66.570	45.620	26.410	33.078	11.887	11.274	11.000	11.649	1662.833	1649.219	1630.740	1662.719
	3000	98.399	95.677	84.508	67.933	96.647	89.959	65.792	48.535	76.919	57.889	34.027	38.150	11.976	11.859	11.566	11.860	1671.422	1660.911	1642.799	1672.319
	6000	99.201	97.833	92.211	82.942	98.342	94.761	78.203	54.035	81.406	64.034	38.173	41.213	11.992	11.943	11.774	11.958	1676.044	1666.298	1652.384	1675.258
	9000	99.468	98.549	94.830	88.621	98.891	96.589	83.001	58.020	83.211	66.209	40.996	42.147	11.990	11.959	11.878	11.966	1678.461	1668.898	1656.286	1677.548
	12000	99.603	98.917	96.116	91.477	99.166	97.449	85.251	61.961	83.910	69.133	42.627	42.432	12.000	11.972	11.912	11.984	1678.946	1669.402	1658.208	1678.212
EDA	1200	51.879	50.645	50.247	50.874	20.987	18.256	18.011	18.038	15.654	18.995	16.654	17.002	9.556	9.111	8.776	9.554	1662.933	1650.063	1381.000	1661.928
	3000	51.654	50.213	50.749	55.845	18.587	18.558	18.841	18.446	15.655	15.699	15.325	15.967	8.997	9.223	9.568	9.668	1672.846	1661.352	1641.698	1672.438
	6000	51.652	50.878	50.625	50.451	20.211	18.659	19.654	18.965	11.558	15.694	16.875	15.874	9.351	9.662	8.231	9.884	1675.443	1666.444	1652.521	1676.018
	9000	51.635	51.524	50.894	52.451	20.111	18.654	18.963	18.549	16.854	13.325	16.895	16.552	9.622	9.304	9.002	9.684	1677.082	1668.926	1656.387	1677.307
	12000	51.623	51.841	50.254	50.647	20.249	20.654	18.122	18.998	17.654	16.215	16.888	17.995	9.335	9.664	8.399	9.645	1678.552	1669.649	1658.087	1678.341

Table A.3. Offline performance obtained by AHMA and SORIGA for each problem, severity and frequency of change.

		OneMax				Plateau				RoyalRoad				Deceptive				Knapsack			
		severity				severity				severity				severity				severity			
	change	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
AHMA	1200	97.648	95.355	90.662	90.607	95.073	89.423	76.900	77.154	79.723	62.762	45.140	42.663	65.123	57.088	53.930	65.736	1679.172	1671.414	1663.916	1669.617
	3000	99.060	98.126	96.236	96.179	98.037	95.763	90.229	90.253	89.845	78.694	62.560	61.201	73.112	64.655	56.092	75.094	1693.010	1681.298	1667.938	1677.169
	6000	99.531	99.069	98.119	98.097	99.023	97.897	95.118	95.180	93.771	86.590	76.134	75.379	78.593	70.534	62.542	80.269	1701.750	1689.347	1671.447	1682.675
	9000	97.688	99.379	98.750	98.727	99.350	98.595	96.739	96.780	95.019	89.807	82.400	82.514	81.574	74.068	65.763	82.302	1705.639	1694.148	1674.089	1688.286
	12000	99.763	99.532	99.060	99.049	99.512	98.943	97.580	97.581	95.731	91.983	85.730	85.788	83.433	76.954	67.740	83.265	1707.703	1697.228	1676.013	1692.966
SORIGA	1200	90.016	81.473	67.956	78.296	81.042	65.262	42.366	59.845	61.767	42.152	26.158	38.683	37.055	35.562	34.111	36.985	1664.800	1664.800	1664.800	1664.600
	3000	94.362	88.155	73.357	88.395	89.072	76.935	51.593	79.263	77.991	58.863	34.353	58.015	37.493	35.244	34.454	35.989	665.920	665.920	665.760	665.920
	6000	96.259	91.274	78.305	90.896	92.677	82.238	57.556	82.715	85.276	69.339	40.614	73.028	39.665	35.925	34.322	36.554	332.940	332.980	332.920	332.960
	9000	97.283	93.250	82.096	92.090	94.657	86.391	61.528	83.634	88.848	74.916	44.886	75.672	40.364	36.345	34.672	36.000	221.960	221.973	221.973	221.947
	12000	97.914	94.420	84.204	93.044	95.945	88.850	64.558	84.777	90.973	77.586	47.633	75.622	41.358	37.051	34.959	36.957	166.450	166.490	166.460	166.480