Bielza C, Fernández del Pozo JA, Larrañaga P. Parameter control of genetic algorithms by learning and simulation of Bayesian networks — A case study for the optimal ordering of tables. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(4): 720–731 July 2013. DOI 10.1007/s11390-013-1370-0

Parameter Control of Genetic Algorithms by Learning and Simulation of Bayesian Networks — A Case Study for the Optimal Ordering of Tables

Concha Bielza, *Member*, *IEEE*, Juan A. Fernández del Pozo, and Pedro Larrañaga, *Member*, *IEEE*
Computational Intelligence Group, Department of Artificial Intelligence, Technical University of Madrid, 28660 Boadilla del Monte, Madrid, Spain

E-mail: {mcbielza, jafernandez, pedro.larranaga}@fi.upm.es

Received June 15, 2012; revised May 27, 2013.

Abstract Parameter setting for evolutionary algorithms is still an important issue in evolutionary computation. There are two main approaches to parameter setting: parameter tuning and parameter control. In this paper, we introduce self-adaptive parameter control of a genetic algorithm based on Bayesian network learning and simulation. The nodes of this Bayesian network are genetic algorithm parameters to be controlled. Its structure captures probabilistic conditional (in)dependence relationships between the parameters. They are learned from the best individuals, i.e., the best configurations of the genetic algorithm. Individuals are evaluated by running the genetic algorithm for the respective parameter configuration. Since all these runs are time-consuming tasks, each genetic algorithm uses a small-sized population and is stopped before convergence. In this way promising individuals should not be lost. Experiments with an optimal search problem for simultaneous row and column orderings yield the same optima as state-of-the-art methods but with a sharp reduction in computational time. Moreover, our approach can cope with as yet unsolved high-dimensional problems.

Keywords genetic algorithm, estimation of distribution algorithm, parameter control, parameter setting, Bayesian network

1 Introduction

In the field of evolutionary computation, parameterized evolutionary algorithms (EAs) modify their behavior by changing the values of their parameters. Finding suitable parameter values for EAs is still an important challenge these days. De Jong[1] and Grefenstette[2] tried to find the optimal and general set of parameters, i.e., applicable to a wide range of problems, for a traditional genetic algorithm (GA) and for a meta-genetic algorithm, respectively. However, when the parameter values are changed, the performance of an EA on a particular fitness landscape can change significantly. Conversely, appropriate parameter settings for one fitness landscape might be inappropriate for others. In fact, no free lunch theorems[3] state that a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration. This calls upon EA practitioners to be careful about generalizing their results and to use parameter setting as a mechanism for adapting algorithms to particular classes of problems.

Following Eiben *et al.*[4] there are two major ways of setting parameter values: parameter tuning and parameter control. In parameter tuning, good values are found before running the algorithm, and these assignments remain fixed during the run. To do this, parameter values are varied systematically, and then statistical techniques, such as factorial design[5] or ANOVA[6-7], are used to analyze the influence of each parameter on EA performance. Some works[8] focus tuning on a subset of problems to output a generalist EA rather than a specialist or problem-tailored EA.

In parameter control, the initial parameter values are changed during the run. The use of rigid parameters with fixed values contrasts with the observation that a run of an EA is an intrinsically dynamic, adaptive process. This is why we will follow the parameter control approach in this paper. Parameter control techniques can be divided into three categories depending on how the parameter change is made: deterministic,

Regular Paper

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness under Grant No. TIN2010-20900-C04-04 and Cajal Blue Brain.

adaptive and self-adaptive. In deterministic parameter control, the parameter values are modified using a deterministic rule that does not take any feedback from the search into account. In adaptive parameter control, feedback from the search is used to determine the parameter changes. Two examples of adaptive parameter control are the 1/5 rule for the evolutionary strategy[9] and the change of probabilistic models during the evolution of estimation of distribution algorithms (EDAs)[10]. In self-adaptive parameter control[11], the parameters are encoded in the chromosomes implementing the idea of evolution of evolution. Other aspects that could be used to classify parameter control techniques include: the target of the change (representation, evaluation function, operators, etc.); the level of change (population-level, individual-level, etc.); and the criterion used to decide the change (operator performance, population diversity, etc.). Other classification schemes for parameter control methods have also been proposed[12-14].

In this paper we propose a new self-adaptive parameter control technique that bears a resemblance to the meta-GA[2,15-17] and the nested evolution strategy[18]. Unlike these approaches, our top-level EA used to conduct the optimization procedure that searches the performance landscape is based on Bayesian network learning and simulation. Its nodes are the parameters for control. The bottom level is a GA. The Bayesian network is learned from the best combination of parameters. These are individuals in the bottom level GA. The Bayesian network is then simulated to generate a new population, thereby evolving the parameters of the bottom-level GA. Learning and simulation steps are repeated until some stopping criterion is met. Therefore, the top-level EA is an EDA[18]. At the top level, the Bayesian network is able to capture the interactions among the parameters, intelligently guiding the search in the performance landscape. Interestingly, at the bottom level, each GA only runs for a few generations. Parameter interaction capture and short-run GAs both reduce the intrinsic computational burden of parameter setting.

Note that research on combining/hybridizing several EA types (or EAs with other heuristics) has been regularly published in the literature[19-22], but their rationale is different with our approach. In [19], GAs and EDAs are combined: two groups of offspring are generated, one by the GA mechanism and the other by the EDA process. The next population is selected from these two groups and the current population. In [20], variable neighborhood search (VNS) and EDAs are hybridized in three different ways: incorporating VNS within EDAs, using probabilistic models within VNS and alternating VNS and EDAs. Continuous optimization is however tackled in [21–22], with hybrid approaches which combine EDAs with differential evolution[21] and with niching strategies[22]. The hybridized heuristics operate on the representation space of solutions to the optimization problem in all these proposals.

Our proposal, however, which might be considered as a hybrid of EDAs and GAs, operates on two representation spaces. EDAs search the GA parameter representation space, whereas GAs search in the optimization problem solution representation space. In addition, each GA only evolves for a few generations. To the best of our knowledge, this is the first paper to propose such an EDA that is able to learn the GA parameter interactions, thereby guiding the search of the GA parameter space.

The outline of the paper is as follows. Section 2 describes our approach, where an EDA guides the GA parameter control. Section 3 presents the experiments on five instances of a combinatorial optimization problem that simultaneously searches for the optimal permutation of table rows and columns. The simpler three instances have already been solved, serving as a benchmark for comparison, whereas the two more complex instances will be solved for the first time using our approach. Section 4 rounds off the paper with the conclusions of this research.

2 Self-Adaptive Genetic Algorithm Parameter Control Guided by an Estimation of Distribution Algorithm

Parameter setting in evolutionary computation has two technical drawbacks: 1) it is time consuming, and 2) it rarely considers the dependency among parameters leading to a simpler procedure where only one parameter is modified at a time. In fact, some researches, e.g., [23] and [24], refer to the impact of these parameter interactions on EA performance.

In this paper we use self-adaptive parameter control to overcome these two drawbacks. Let us assume that a GA is used to solve an optimization problem. Our approach is composed of two levels: the top level and the bottom level. At the top level, a Bayesian network guides an intelligent search of the parameter space. The Bayesian network nodes are the parameters of the GA for optimization. This explicitly captures the dependency among parameters. This Bayesian network is induced from a dataset via a structure learning algorithm. The dataset is composed of cases corresponding to the best configurations of the GA parameters. Cases initially generated at random are simulated from the Bayesian network in subsequent iterations. This process is an EDA, more specifically, an estimation of Bayesian network algorithm (EBNA)[25], although any other EDA could be used, for example, the Bayesian

optimization algorithm (BOA)[26]. At the bottom level, the GA is run on a fixed case (configuration of the parameters). However, this GA is stopped before it converges. This reduces its computational burden. Therefore, the top-level Bayesian network and the bottom-level GA procedure overcome drawbacks 2) and 1) mentioned above respectively.

Stated formally, assume we have an optimization problem defined in the (bottom-level) space Ω_b of solutions $\boldsymbol{x} = (x_1, \ldots, x_m)$ to find $\boldsymbol{x}^*$, the global minimum of a function $f_b : \Omega_b \to \mathbb{R}$, i.e.,

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega_b} f_b(\boldsymbol{x}). \tag{1}$$

Suppose we use a GA (bottom-level), determined by a vector of parameters $\boldsymbol{z} = (z_1, \ldots, z_n)$ in the space Ω_t (top-level), to address this problem. Running this GA under the $\boldsymbol{z}$ parameter configuration provides a solution $\boldsymbol{x}_{\boldsymbol{z}}$ with fitness $f_t(\boldsymbol{z})$, where

$$f_t : \Omega_t \to \mathbb{R}.$$

Then, problem (1) is transformed into a search of Ω_t, since $f_t(\boldsymbol{z}) = f_b(\boldsymbol{x}_{\boldsymbol{z}})$. At the top level, problem (1) is equivalent to solving

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}\in\Omega_t} f_t(\boldsymbol{z}).$$

Note that the different configurations $\boldsymbol{z}$ and $\boldsymbol{z}'$ may lead to the same solution $\boldsymbol{x}$ for the original problem, whereas the same configuration $\boldsymbol{z}$ can provide two different solutions $\boldsymbol{x}$ and $\boldsymbol{x}'$ due to the stochasticity of the GA.

The pseudo-code of our proposal is shown in Fig.1. Fig.2 illustrates the idea.

At the top level, populations of $\boldsymbol{z}$-individuals and generation j are denoted as $\mathcal{P}_t^{(j)}$. Their size is N individuals. Since each $\boldsymbol{z}$ is a fixed configuration of a GA, we run this GA with (other) populations of M individuals in the space of solutions $\boldsymbol{x}$ of the original problem. Each bottom-level population is associated with a fixed $\boldsymbol{z}$ and generation j (from the top level), where each population at generation l (from the bottom level) is denoted as $\mathcal{P}_{(b,\boldsymbol{z})}^{(j,l)}$. However, these populations are only evolved for g generations, where g can be a long way from convergence. Since these few generations are not computationally prohibitive, we expect to finally find the solution guided by the EDA that selects N' ($< N$) $\boldsymbol{z}$-configurations at the top level.

The underlying assumption is that there are a number of generations at the bottom level, g, with a population size, M, such that the following two pools of $N' < N$ individuals are the best ranked: 1) the set of selected $\boldsymbol{z}$-individuals with fitness values output by running the GA with a large population size (bigger than M) until convergence; and 2) the set of selected $\boldsymbol{z}$-individuals with fitness values approximated by a short

Input: N, N' (top level) and M (bottom level)
$j = 0$ [Generation counter for the top level]
1. [Initial sampling]
$\mathcal{P}_t^{(j)} \leftarrow$ Sample N $\boldsymbol{z}$-individuals at random
Find $g \geqslant 3$ with the approximate test (Subsection 3.2): three consecutive generations with the same top-ranked individuals
For each $\boldsymbol{z}$ in $\mathcal{P}_t^{(j)}$ {
[Bottom level]
Run a genetic algorithm for g generations under the $\boldsymbol{z}$ parameter configuration
Output: g populations of size M with $\boldsymbol{x}$-individuals: $\mathcal{P}_{(b,\boldsymbol{z})}^{(j,0)}, \ldots, \mathcal{P}_{(b,\boldsymbol{z})}^{(j,g)}$
Evaluate $\boldsymbol{z}$ as $f_t(\boldsymbol{z}) = f_b(\boldsymbol{x}_{\boldsymbol{z}})$
}
[Top level] **Repeat until** a stopping criterion is met
2. [Selection]
$\mathcal{P}_t^{(j),sel} \leftarrow$ Select $N' < N$ individuals from $\mathcal{P}_t^{(j)}$
3. [Learning]
$BN^{(j)} \leftarrow$ Learn a Bayesian network from $\mathcal{P}_t^{(j),sel}$
4. [Sampling]
$\mathcal{P}_t^{(j+1)} \leftarrow$ Sample N individuals from $BN^{(j)}$
Find $g \geqslant 3$ with the approximate test (Subsection3.2): three consecutive generations with the same top-ranked individuals
For each $\boldsymbol{z}$ in $\mathcal{P}_t^{(j+1)}$ {
[Bottom level]
Run a genetic algorithm for g generations under the $\boldsymbol{z}$ parameter configuration
Output: g populations of size M with $\boldsymbol{x}$-individuals: $\mathcal{P}_{(b,\boldsymbol{z})}^{(j+1,0)}, \ldots, \mathcal{P}_{(b,\boldsymbol{z})}^{(j+1,g)}$
Evaluate $\boldsymbol{z}$ as $f_t(\boldsymbol{z}) = f_b(\boldsymbol{x}_{\boldsymbol{z}})$
}
Output: $\mathcal{P}_t^{(j+1)}$ and $\hat{\boldsymbol{x}}^*$ obtained after running the GA until it converges for each $\boldsymbol{z} \in \mathcal{P}_t^{(j+1)}$

Fig.1. Pseudo-code of our algorithm.

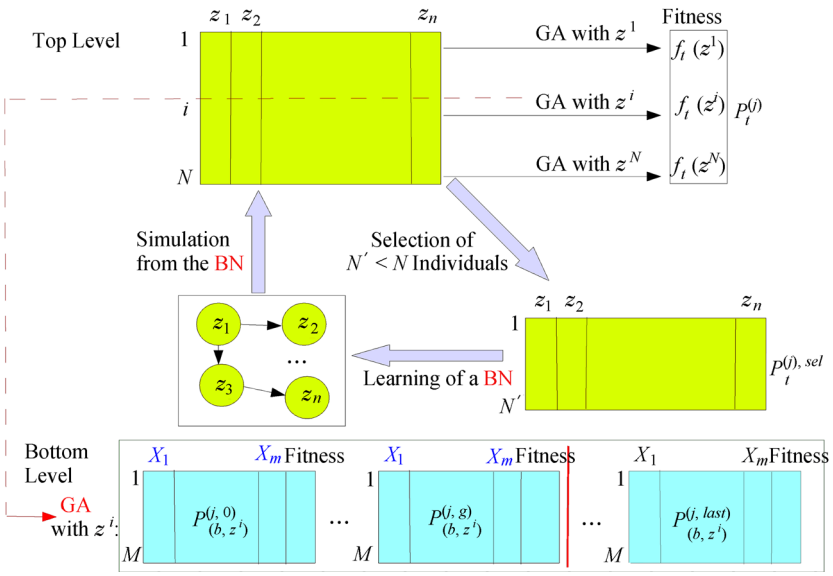

Fig.2. Our algorithm: EDA (top level) and GA (bottom level).

evolution (g generations) of the GA. This hypothesis cannot be tested in practice due to the high computational burden of computing the first set of individuals. Therefore, we propose an approximate test method (see Section 3).

Stopping GAs before convergence resembles statistical racing techniques applied to EAs, as in [27]. Such techniques propose hybridizing racing and a $(1+\lambda)$ evolutionary strategy. The aim is to find the best individual, which is a parameter setting of an EA. Evolving each EA (in our case a GA) until convergence would be very time consuming. Instead, these individuals are raced, and the individuals that evolve significantly worse than others are removed from the race until only the best one is left. Setting the population size is a problem in its own right[28] and could be considered as another parameter for control. However, our method cannot account for the population size, M, and the number of GA generations, g, since they must be kept low for the racing to be efficient.

Unlike [27], our aim is not to find a single individual but a set of individuals. Yuan and Gallagher[27] used a different number of generations for each individual depending on the result of a statistical test used to compare fitness values. Instead, we stop the GA after a number of generations g. This is dynamically computed at each EDA generation from the approximate tests of the underlying hypothesis.

The steps for selecting N' individuals from $\mathcal{P}_t^{(j)}$, learning the Bayesian network and simulating N new individuals are as follows. First, we fix the value of N' based on a result given in [29]. The authors gave an upper bound for the number of instances needed to learn a Bayesian network from data, where the minimum description length (MDL) metric is used as the score. The bound depends on two parameters ε and δ, guaranteeing that the Kullback-Leibler divergence between the target distribution (from which training samples are drawn) and the learned distribution is at least ε — error threshold — with a probability of at most δ — confidence threshold. Then, the sample size N' should be greater than $(\frac{1}{\varepsilon})^{\frac{4}{3}} \log \frac{1}{\varepsilon} \log \frac{1}{\delta} \log \log \frac{1}{\delta}$. Second, according to this result, the Bayesian network structure learning algorithm must be based on the MDL principle. Thus, we use the algorithm described in [30], which runs a greedy search in the space of directed acyclic graphs. The conditional probabilities that define the Bayesian network are estimated by the maximum likelihood method. Third, individuals can be simulated from the Bayesian network using a standard procedure, like probabilistic logic sampling (PLS)[31]. Finally, the stopping criterion is based on the absence of arcs in the Bayesian network, i.e., the GA parameters are mutually independent. This occurs when most of the individuals are equal. Once the EDA has stopped, the fitness of the best $\boldsymbol{z}$-individual is refined by running the GA in the usual manner (until convergence and with an appropriate population size). This results in the final proposed solution $\hat{\boldsymbol{x}}^*$, an estimation of $\boldsymbol{x}^*$, the solution of problem (1).

In short, the challenge is to find adequate values for the population size, M, and the number of GA generations, g, in order to correctly explore the search space without wasting computational resources. The current approach helps to overcome the two technical hitches mentioned above: 1) the time-consuming task of parameter setting in GA by stopping GAs before convergence, and 2) the often overlooked dependency among

parameters using the Bayesian network defined in the parameter space. Thus, we are looking for a trade-off between available computational resources and our short-run GA strategy that will identify the best-ranked solutions. The aim of the Bayesian network of the EDA is to represent a probability model of the GA parameters; the underlying problem structure is then learned on the fly. The EDA, in charge of exploring the parameter space, and ultimately the solution space, will correct bad parameterizations and will find better options.

3 Experiments

This section reports experiments on a problem that consists of finding the optimal ordering of table rows and columns to improve table readability. Subsection 3.1 describes the problem, introducing the Bertin-type tables that we will use. Subsection 3.2 specifies how the above algorithm is run. Subsection 3.3 shows the results.

3.1 Optimal Ordering of Tables

Illustratively, we choose a descriptive statistics problem, where the rows and columns of a table should be rearranged to show up interesting patterns and ease interpretation. Table ordering is irrelevant, and any permutation of table rows and columns is allowed.

Bertin[32] introduced the following 9×16 Bertin table, see Table 1, where columns are townships, whereas rows are characteristics that are present (1) or absent (0) in the townships. We consider an arbitrary ordering of all rows and columns.

Fig.3(a) shows a graphical representation of a 144×128 table, called `Bertin128` in this paper, constructed by repeating Table 1 several times. A present (absent) characteristic is shown in red (cream). Fig.3(b) contains the same information which is, however, displayed more clearly after reordering the rows and columns to reveal patterns. This clarity is evaluated using a measure of conciseness, considering the similarity between each table entry and its neighbors (see below).

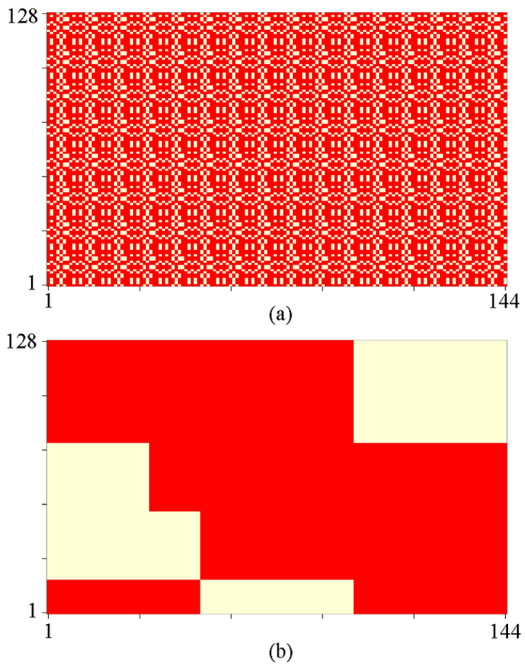

Fig.3. (a) `Bertin128` table and (b) the same table after reordering rows and columns.

For a table with r rows and c columns, $r! \times c!$ permutations should be considered. The search space cardinality of this problem is the product of the search space cardinalities of two traveling salesman problems, which are known to be NP-complete problems[33]. This justifies using heuristics to solve the problem. Niermann[34] used a simple GA.

An individual is composed of $r + c$ genes, that is, a vector of length $r + c$ given as

$$\boldsymbol{x} = (\pi^r, \pi^c) = (\pi^r(1), \ldots, \pi^r(r), \pi^c(1), \ldots, \pi^c(c)), \tag{2}$$

where π^r (π^c) indicates a permutation of rows (columns). Therefore, $\pi^r(i)$ is the position of the row in the original table that moves to the i-th row of the table that the individual represents. The same idea applies to columns. Crossover and mutation operators are applied to each permutation (rows and columns) separately.

Table 1. Original `Bertin` Table with Characteristics of 16 Townships

Characteristics	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
High School	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
Agricult. Coop.	0	1	1	1	0	0	1	0	0	0	0	1	0	0	1	0
Railway Station	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
One-Room-School	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	1
Veterinary	0	1	1	1	0	0	1	0	0	0	0	1	0	0	1	0
No Doctor	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	1
No Water Supply	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
Police Station	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
Land Reallocation	0	1	1	1	0	0	1	0	0	0	0	1	0	0	1	0

One way to quantify the conciseness of a table is by considering how different each table entry is from the values of its neighboring entries. The Moore neighborhood considers eight neighboring entries and a dissimilarity measure to gauge the diffuseness that has to be minimized. The fitness function considers, for each table entry $e_{\pi^r(i),\pi^c(j)}$ in the $\pi^r(i)$-th row and $\pi^c(j)$-th column, a local stress measure $s(\pi^r(i),\pi^c(j))$ of dissimilarity between this entry and its eight neighboring entries $e_{\pi^r(l),\pi^c(m)}$, given by the squared differences:

$$s(\pi^r(i),\pi^c(j)) = \sum_{l=\max(1,\pi^r(i-1))}^{\min(r,\pi^r(i+1))} \sum_{m=\max(1,\pi^c(j-1))}^{\min(c,\pi^c(j+1))} \cdot \left(e_{\pi^r(i),\pi^c(j)} - e_{\pi^r(l),\pi^c(m)}\right)^2$$

If applied to all table entries we get a global stress measure of the individual $\boldsymbol{x} = (\pi^r, \pi^c)$:

$$f_b(\boldsymbol{x}) = f_b(\pi^r,\pi^c) = \sum_{i=1}^{r}\sum_{j=1}^{c} s(\pi^r(i),\pi^c(j)),$$

which has to be minimized.

For the table in Fig.3(a), this measure yields the value 64 960, which drops to 1 986 in the table in Fig.3(b).

Bielza *et al.*[24] improved Niermann's GA by considering up to seven parameters: initial population, crossover operator, crossover probability, mutation operator, mutation probability, replacement policy and stopping criterion, see columns 1 and 2 in Table 2. These are also used in this paper.

Table 2. Parameterization of the GA

Parameter	Possible Values	Choice in [34]
Initial population	`random, heuristic`	`random`
Crossover operator	`rxc, pmx, cx, ox1, ox2, ap, vr`	`rxc`
Crossover probability	0.50, 0.85, 0.90, 0.95	0.50
Mutation operator	`2opt, dm, em, ism, tim, ivm, sm`	`2opt`
Mutation probability	0.50, 0.10, 0.05, 0.01	0.50
Replacement policy	`ets, exs, fsb`	`ets`
Stopping criterion	`fix, lock, var`	`fix`

The initial population was chosen at `random` or using a `heuristic`. `Random` means that two random permutations, one for rows and one for columns, are generated in a best-first manner. `Heuristic` initializes the previous approach by first moving rows whose values have a greater sum (a higher number of ones in the binary 0/1 case) to the top. It then rearranges the columns, moving columns with higher sums (a higher number of ones in the binary case) to the left.

As far as the crossover operators are concerned, Niermann[34] simply chose to produce two offsprings from two parents that exchanged their respective permutation of columns. As a result there are no changes in any position of the two orderings (rows and columns), preventing the expected diversity that these operators usually introduce. We will call this type of crossover operator `rxc` (row times columns). With the aim of providing crossover-induced population diversity, however, the other operators used in [24] include rearrangements of both parts, rows and columns. We borrowed the following six operators from the literature on permutation problems like the traveling salesman problem (see, e.g., [35]): partially-mapped crossover operator (`pmx`), cycle crossover operator (`cx`), order crossover operator (`ox1`), order-based crossover operator (`ox2`), alternating-position crossover operator (`ap`), and voting recombination operator (`vr`).

The crossover probability is usually high. We use probabilities of 0.50, 0.85, 0.90 and 0.95. Niermann[34] set probability at 0.50.

As regards mutation operators, Niermann[34] chose the `2opt` operator[36] applied separately to both the row and column permutation vectors. This operator reverses the order of the genes between two randomly chosen breakpoints. In [24], we also borrowed six operators from the literature: displacement mutation operator (`dm`), exchange mutation operator (`em`), insertion mutation operator (`ism`), tail simple-inversion mutation operator (`tim`), inversion mutation operator (`ivm`), and scramble mutation operator (`sm`).

Although the mutation probability is usually near zero, Niermann[34] set it, like the crossover probability, at 0.50. In [24] we employed lower mutation probabilities of 0.10, 0.05 and 0.01, as it is in usual practice.

We proposed three replacement policies in [24]. The first policy, elite tournament selection (`ets`), chooses the best individual of each pairwise tournament. The second policy, elite times selection (`exs`), replaces individuals with worse-than-average fitness by individuals from the generation with the best fitness stored so far. In the third policy, file segment block (`fsb`), the 33% worst-ranked individuals are replaced by the 33% best-ranked individuals.

Finally, we considered three stopping criteria: stop after a fixed number of generations (256 in our experiments) (`fix`), as in [34]; stop after 128 generations in which there has been no improvement of the fitness function (`lock`); and stop whenever the coefficient of variation of the f_b stress measure is less than 3 (`var`).

Whereas Niermann[34] always used a single parameterization of the GA (see column 3 in Table 2), each experiment carried out in [24] covers all possible combinations of the seven parameters, that is, 2×7×4×7×4×

$3 \times 3 = 14\,112$ (see Table 2). Moreover, each experiment was run 10 times using each combination of parameters due to GA stochasticity. We chose a set of 7 Bertin tables. In this paper, we will take the biggest three tables, `Bertin8, Bertin32` and `Bertin128`. Moreover, we will also include results for two bigger tables, `Bertin256` and `Bertin512`, which Bielza *et al.*[24] were unable to solve for computational reasons. Table 3 shows their dimensions.

Table 3. Dimensions of the Set of Tables

Table	Number of Rows r	Number of Columns c
`Bertin8`	32	36
`Bertin32`	64	72
`Bertin128`	128	144
`Bertin256`	256	288
`Bertin512`	512	576

Table 4 shows the stress values f_b for each Bertin table and computation time reported in [34] and in [24]. Time for `Bertin256` and `Bertin512`, which were not solved in [24], is an approximation based on the time taken to compute each fitness value (the only operation with a non-negligible computation time) multiplied by the number of evaluated individuals. This number is calculated as the product of the population size, the number of generations and 14 112.

Table 4. Stress Values f_b and Computation Time of the Set of Tables

Table	Initial f_b	Best f_b in [34]	Best f_b in [24]	Time in [34]	Time in [24]
`Bertin8`	3 952	550	474	02 min	104 h 24 min
`Bertin32`	16 096	2 010	978	06 min	739 h 32 min
`Bertin128`	64 960	15 704	1 986	55 min	28 259 h 17 min
`Bertin256`	260 992	124 232	N/A	09 h 51 min	$\approx$ 596 368 h
`Bertin512`	1 046 272	855 428	N/A	24 h 06 min	$\approx$ 12 571 066 h

Note that our results are much better than Niermann's[34], especially for large-sized tables. However, they took much longer to compute. For example, for `Bertin128`, Niermann[34] reached a solution with a stress value of 15 704 in 55 minutes, whereas Bielza *et al.*[24] obtained a stress of 1 986 in 28 259 hours and 17 minutes (on a 180 eServer BladeCenter JS20 cluster, with 2 744 CPUs and main memory of 5 488 GB). Computational time is excessive and is even worse for bigger tables, as shown in Table 4. This is our motivation for researching the new approach developed here. Obviously, there are many other examples where this would be applicable.

3.2 Setting up the Algorithm

Remember that we have two nested levels, the top level governed by an EDA and a bottom level governed by a GA. With Bertin tables, top-level individuals are fixed configurations $\boldsymbol{z}$ of the GA, given in this case by a vector of seven components, each corresponding to a parameter (listed in Table 2) to be controlled. Bottom-level individuals are solutions $\boldsymbol{x}$ of the original problem, given in this case by a vector of $r+c$ components, where r (c) is the number of rows (columns) in the Bertin tables (see Table 3). Vector $\boldsymbol{x}$ includes a permutation of the r rows followed by a permutation of the c columns, as shown in (2). Thus, for `Bertin128`, $r = 144$ and $c = 128$, and the bottom-level individuals have a dimension of 272.

The population size at the top level, N, is fixed at 92 individuals. If we set the error threshold $\varepsilon = 0.08$ and the confidence threshold $\delta = 0.001$ to guarantee a similarity between the target and the learned distributions in the sense of Kullback-Leibler divergence, N' is 46 individuals. Thus, we will select the best 46 individuals in each EDA generation from the whole population of size 92.

Two key parameters are left: the population size on which the GA will be run at the bottom level, M, and the number g of (few) generations of the GA. Both should be chosen to comply with our underlying assumption (see Section 2), that is, the sets of N' individuals that the EDA selects at each generation should match irrespective of whether the GA 1) is run until convergence (with a usual population size) and 2) uses only g generations and sparse populations of M individuals. M is fixed as a function of the table dimensions: $M = \lfloor \ln(r \times c) \rfloor + 1$, i.e., 8, 9, 10, 12 and 13 for `Bertin8, Bertin32, Bertin128, Bertin256` and `Bertin512`, respectively.

It is unfeasible to check that assumption for most of the problems addressed here since it is computationally prohibitive to run the GA until convergence. In fact, this is the motivation for the approach introduced in this paper. Therefore, we check this point approximately as follows. To set g, we progressively increase the number of generations g of the GA at the bottom level and observe the resulting N' selected individuals at the top level. We choose g as the minimum number of generations when this pool of individuals is unchanged for three consecutive $g-2, g-1$ and g generations.

Table 5 shows each g found for each Bertin table at each EDA generation.

For `Bertin128`, for instance, our algorithm found that the GA run for 23, 24 and 25 generations returned the same pool of best $N' = 46$ individuals in the third generation of the EDA (column labelled "3"), which differed from the pool output by the GA run for 22 generations, and therefore $g = 25$.

This is illustrated in Fig 4. In Fig.4(a), the 92 individuals are shown on the X axis against their GA fi-

Table 5. Number of Generations $g$of the GA Found as EDA Evolves

Table	EDA Generations											
	1	2	3	4	5	6	7	8	9	10	11	12
`Bertin8`	15	39	44	41	21	35	12	19	5	3	–	–
`Bertin32`	10	20	48	23	16	28	8	7	7	7	7	–
`Bertin128`	20	11	25	48	29	16	44	15	26	13	6	6
`Bertin256`	18	22	35	29	23	24	7	44	13	6	–	–
`Bertin512`	16	23	33	29	38	7	48	39	22	14	–	–

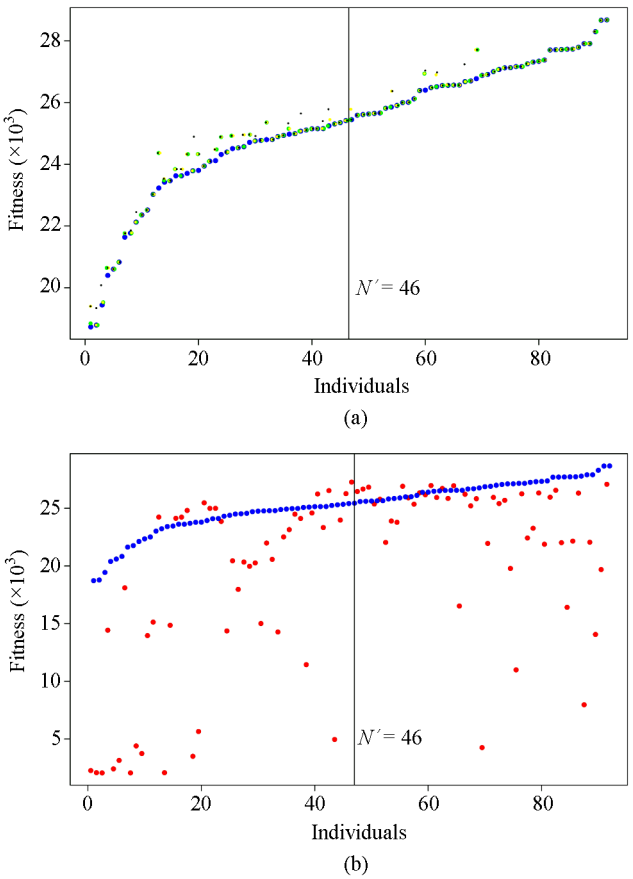

Fig.4. (a) Fitness values of the 92 individuals when running the GA for 22 generations (black), 23 generations (yellow), 24 generations (green) and 25 generations (blue) for `Bertin128` table at the third generation of the EDA. (b) Fitness values of the 92 individuals when running the GA until convergence (red) for `Bertin128` table. Blue points are as in (a).

fitness (Y axis), approximately output by 25 generations (blue), 24 generations (green), 23 generations (yellow) and 22 generations (black). The numbers on the X axis are ordered according to fitness values after 25 generations. Note that for some individuals that are not among the best 46, their fitness values (black points) are better than the worst value of this group of 46. In other words, the partition provided by the 25 generations into good (the best 46) and bad (the other) individuals is valid for the individuals output after 23 and 24 generations but not for the individuals output after 22 generations. In Fig.4(b), red points show the fitness running the (long) GA until convergence (256 generations and a population of 544 individuals), computed in [24]. For comparison purposes, the blue points plotted in Fig.4(a) are reproduced. Most of the better (real) points are chosen by our group of 46 individuals, output with small-sized populations and just a few (25) generations of our short GA.

To further strengthen the implicit assumption, we can compute the magnitude of difference between neighboring configurations. For example, this is computed for `Bertin128` at the 10th generation of the EDA (column labelled "10") as the sum of the fitness differences between individuals that are equally ranked in two consecutive generations. This magnitude is 404 768 (from generation 4 to 5), 349 346 (from 5 to 6), 182 362 (from 6 to 7), 107 474 (from 7 to 8), 55 594 (from 8 to 9), 71 982 (from 9 to 10), 6 042 (from 10 to 11), and 0 for the other comparisons. Therefore, differences decay to zero until we find the value of g. The same behavior is observed for other cases.

The number of generations at which the EDA needs to be stopped varies considerably as shown in Table 5, where we see an initially upward trend, which then drops to low values. Some Bertin tables stopped before others, and this is what the "−" symbol indicates in Table 5. Note that, fortunately, the g value is dynamically inferred from data at each EDA generation.

3.3 Results

Table 6 shows the best results and computational time output by our algorithm. Table 6 also includes the same results obtained with the more exhaustive strategy following [24].

Table 6. Best Results and Computational Time of the Set of Tables with Our Algorithm

Table	Best f_b in [24]	Time in [24]	Our Best f_b	Our Time
`Bertin8`	474	104 h 24 min	474	2 h 00 min
`Bertin32`	978	739 h 32 min	978	6 h 48 min
`Bertin128`	1 986	28 259 h 17 min	1 986	15 h 50 min
`Bertin256`	N/A	≈596 368 h	4 002	34 h 52 min
`Bertin512`	N/A	≈12 571 066 h	15 240	143 h 11 min

For tables with reported results, like `Bertin8`, `Bertin32` and `Bertin128`, the best fitness values are the same, although there is a sharp reduction in the computational time taken. Using the strategy proposed in [24], `Bertin8`, `Bertin32` and `Bertin128` took approximately 52 109 and 1 788 times longer than with our new approach. For (as yet unsolved) bigger tables, like `Bertin256` and `Bertin512`, we provide good enough so-

lutions in a reasonable time. Observe in Table 4 that Niermann[34] required a quarter of the computational time to output a much worse solution: 124 232 and 855 428 compared with our solutions of 4 002 and 15 240 for these two big tables, `Bertin256` and `Bertin512`, respectively. A solution for each of these two tables is shown in Fig.5, both of which have a good rearrangement of rows and columns.

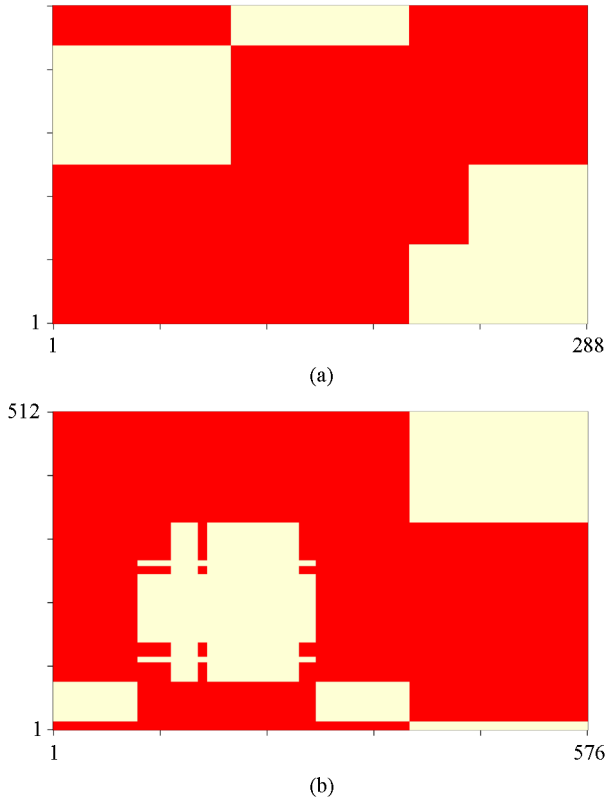

Fig.5. Solution of our algorithm for (a) `Bertin256` and (b) `Bertin512`.

In general, the computational cost of the EDA is very low compared with the classical tuning strategy, where many configurations are run many times to find the best option. In our experiments, we run fewer than 1% of GA configurations, using a tiny population for a few generations (given by M and g, respectively). It takes around one second to find g, since the fitness values of only three populations have to be ordered. The learning and simulation processes of the Bayesian network models are also low cost because the data sets they use are small ($n = 7$ columns and $N = 92$ or $N' = 46$ rows): the structure search takes about ten minutes and the simulation process two minutes for all problems (they could be added to the result tables).

Let us now compare our algorithm with a procedure that does not include the learning and simulation steps that are essential to our approach. This random adaptive method works as follows. First, random populations (of GA parameterizations) are independently generated from univariate uniform distributions. Second, the N' best individuals are selected. Third, the next population is formed by combining these N' individuals with $N - N'$ randomly generated new individuals. From this pool of N individuals, we then select the N' best individuals. This random generation-and-selection procedure is repeated until a stopping criterion is met (convergence or maximum number of iterations). Fig.6 shows how the fitness values of the last population are distributed for `Bertin128`. The top chart illustrates our algorithm where the values are the same for the whole population and have converged on the best solution. However, the bottom chart illustrates this random adaptive method, displaying a population which has not converged and still has far-from-optimum fitness values.

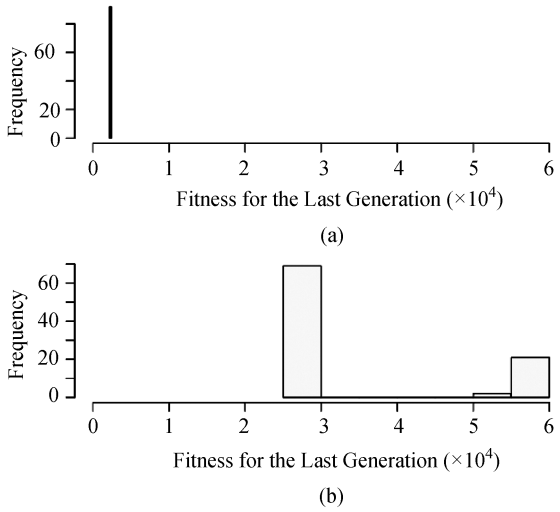

Fig.6. Distribution of `Bertin128` fitness values for the last generation of (a) our algorithm and (b) a random adaptive method.

The Bayesian networks learned at each EDA generation encode information about the parameters to be controlled in the GA, i.e., the nodes of this network. Illustratively, Fig.7 shows the evolution of the Bayesian networks learned for `Bertin128`. Note that the initial networks are denser, capturing probabilistic dependencies between some of the parameters. Thus, in the initial Bayesian network, the crossover operator parameter depends on crossover probability and the mutation operator, which in turn depends on mutation probability and crossover probability, which in turn depends on the stopping criterion, replacement policy and mutation probability. The initial population is independent of the other parameters. Moreover, we can derive probabilistic conditional independencies from the local Markov property. This property states that in any Bayesian

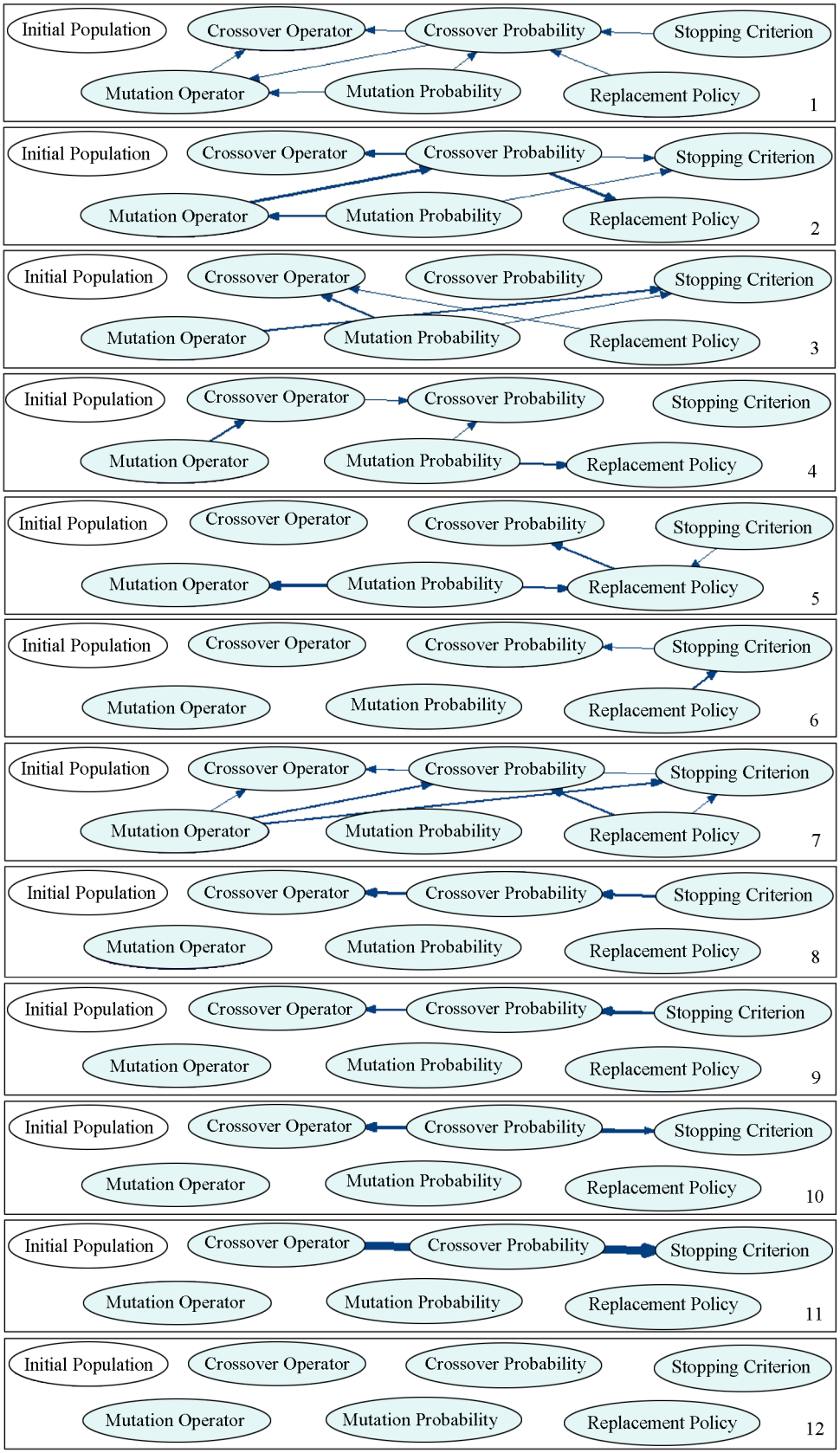

Fig.7. Bayesian networks learned on the seven GA parameters (nodes) at the different EDA generations for `Bertin128`.

network any node is conditionally independent of its non-descendants given its parents[37]. In the initial Bayesian network, we could infer, that given the crossover and mutation probabilities, for example, then the mutation operator is conditionally independent of the stopping criterion, replacement policy and initial population. In other words, once the probabilities of mutation and crossover have been set, the mutation operator has no need of the stopping criterion, replacement policy and initial population values. As long as the EDA evolves and finds better configurations, the networks are sparser. By the last Bayesian network the parameters are independent of each other (there are no arcs), since all the individuals are equal. Note that the strength of the relationship, represented by the arc thickness, is stronger in the final networks.

4 Conclusions

This paper has proposed a new approach for self-adaptive GA parameter control. The idea is to evolve, at a top level, a set of individuals representing different parameter configurations of the GA. To do this, we use an EDA, where a Bayesian network is learned from the pool of the best parameter configurations and then new individuals are sampled according to the joint distribution function encoded by this network. At the bottom level, the GA runs use small population sizes and are stopped within a few generations. This provides a quick estimation of the fitness function that the EDA needs to evolve further. The required number of generations is identified automatically using a test: we order the individuals and check whether the best-ranked individuals are unchanged over a number of generations. This test approximates the underlying hypothesis defended here that a standard (long) GA and a shorter EDA-driven version run on small populations output the same best individuals. The number of individuals in the EDA populations (for learning the Bayesian network and simulating new individuals) is determined by a theoretical argument that guarantees that the true Bayesian network is learned within fixed error and confidence thresholds.

The approach has been tested on a combinatorial optimization problem aimed at finding the best organization of rows and columns of Bertin-type tables. The optimal values found are equal to other state-of-the-art GAs for tables of moderate sizes. However, we considerably reduced the computational burden. Furthermore, we are able to successfully deal with bigger tables, that state-of-the-art GAs are unable to solve satisfactorily. Using an EDA to guide a short-run GA to process big problems like these is the leading contribution of this approach.

Obviously, the approach is general enough to be applied to any other population-based evolutionary algorithm apart from GAs. Moreover, since there are EDAs designed to handle continuous or even mixed variables, our framework could be used for evolutionary algorithms that have to control continuous and discrete parameters. This is an interesting research topic for the near future.

References

[1] De Jong K A. An analysis of behavior of a class of genetic adaptive systems [Ph.D. Thesis]. University of Michigan, USA, 1975.

[2] Grefenstette J J. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 1986, 16(1): 122-128.

[3] Wolpert D, Macready W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1997, 1(1): 67-82.

[4] Eiben A E, Michalewicz Z, Schoenauer M, Smith J E. Parameter control in evolutionary algorithms. In *Studies in Computational Intelligence 54*, Lobo F G, Lina C F, Michalewicz Z *et al.* (eds.), Springer, 2007, pp.19-46.

[5] de Lima E B, Pappa G L, de Almeida J M *et al.* Tuning genetic programming parameters with factorial designs. In *Proc. 2010 IEEE Congress on Evolutionary Computation*, July 2010.

[6] Rojas I, González J, Pomares H *et al.* Statistical analysis of the main parameters involved in the design of a genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics — Part C, Applications and Reviews*, 2002, 32(1): 31-37.

[7] Czarn A, MacNish C, Vijayan K *et al.* Statistical exploratory analysis of genetic algorithms: The importance of interaction. *IEEE Trans. Evolutionary Computation*, 2004, 8(4): 405-421.

[8] Smit S K, Eiben A E. Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In *Lecture Notes in Computer Science 6024*, Di Chio C, Cagnoni S, Cotta C *et al.* (eds.), Springer, 2010, pp.542-551.

[9] Rechenberg I. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Stuttgart, Germany: Frommann-Holzboog, 1973. (In German)

[10] Santana R, Larrañaga P, Lozano J A. Adaptive estimation of distribution algorithms. In *Studies in Computational Intelligence 136*, Cotta C, Sevaux M, Sörensen K *et al.* (eds.), Springer, 2008, pp.177-197.

[11] Kramer O. Self-Adaptive Heuristics for Evolutionary Computation. Berlin, Germany: Springer-Verlag, 2008.

[12] Angeline P J. Adaptive and self-adaptive evolutionary computation. In *Computational Intelligence: A Dynamic System Perspective*, Palaniswami Y, Attikiouzel R, Marks R *et al.* (eds.), IEEE, 1995, pp.152-161.

[13] Hinterding R, Michalewicz Z, Eiben A E. Adaptation in evolutionary computation: A survey. In *Proc. the 4th IEEE Conf. Evolutionary Computation*, Apr. 1997, pp.65-69.

[14] Smith J. Self adaptation in evolutionary algorithms [Ph.D. Thesis]. University of the West of England, UK, 1997.

[15] Friesleben B, Hartfelder M. Optimisation of genetic algorithms by genetic algorithms. In *Proc. Artificial Neural Networks and Genetic Algorithms*, Apr. 1993, pp.392-399.

[16] Bäck T. Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Al-

gorithms. New York, USA: Oxford University Press, 1996.
[17] Rechenberg I. Evolutionsstrategie'94. Stuttgart, Germany: Frommann-Holzboog, 1994.
[18] Larrañaga P, Lozano J A. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. New York, USA: Kluwer Academic Publishers, 2002.
[19] Peña J M, Robles V, Larrañaga P *et al.* GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In *Proc. the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, May 2004, pp.361-371.
[20] Santana R, Larrañaga P, Lozano J A. Combining variable neighborhood search and estimation of distribution algorithms in the protein side chain placement problem. *Journal of Heuristics*, 2008, 14(5): 519-547.
[21] Sun J, Zhang Q, Tsang E. DE/EDA: A new evolutionary algorithm for global optimisation. *Information Sciences*, 2005, 169(3/4): 249-262.
[22] Dong W, Yao X. NichingEDA: Utilizing the diversity inside a population of EDAs for continuous optimization. In *Proc. the 2008 IEEE Congress on Evolutionary Computation*, June 2008, pp.1260-1267.
[23] Nannen V, Smit S K, Eiben A E. Costs and benefits of tuning parameters of evolutionary algorithms. In *Proc. the 10th Int. Conf. Parallel Problem Solving from Nature*, Sept. 2008, Vol.5199, pp.528-538.
[24] Bielza C, Fernández del Pozo J A, Larrañaga P *et al.* Multidimensional statistical analysis of the parameterization of a genetic algorithm for the optimal ordering of tables. *Expert Systems with Applications*, 2010, 37(1): 804-815.
[25] Etxeberria R, Larrañaga P. Global optimization using Bayesian networks. In *Proc. the 2nd Symposium on Artificial Intelligence*, March 1999, pp.332-339.
[26] Pelikan M, Goldberg D E, Cantú-Paz E. BOA: The Bayesian optimization algorithm. In *Proc. the Genetic and Evolutionary Computation Conference*, July 1999, pp.525-532.
[27] Yuan B, Gallagher M. Combining meta-EAs and racing for difficult EA parameter tuning tasks. In *Studies in Computational Intelligence 54*, Lobo F J, Lima C F, Michalewicz Z (eds.), Springer, 2007, pp.121-142.
[28] Lobo F G, Lima C F. Adaptive population sizing schemes in genetic algorithms. In *Studies in Computational Intelligence 54*, Lobo F, Lima C F, Michalewicz Z (eds.), Springer, 2007, pp.185-204.
[29] Friedman N, Yakhini Z. On the sample complexity of learning Bayesian networks. In *Proc. the 12th Conference on Uncertainty in Artificial Intelligence*, Aug. 1996, pp.274-282.
[30] Lam W, Bacchus F. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 1994, 10(3): 269-293.
[31] Henrion M. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Proc. the 2nd Annual Conf. Uncertainty in Artificial Intelligence*, Aug. 1986, pp.149-164.
[32] Bertin J. Graphics and Graphic Information Processing, UK: Walter de Gruyter & Co, 1981.
[33] Johnson D S, Papadimitriou C H. Computational complexity. In *The Traveling Salesman Problem*, Lawler E L, Lenstra J K, Rinnooy Kan A *et al.* (eds.), John Wiley & Sons, 1985, pp.37-85.
[34] Niermann S. Optimizing the ordering of tables with evolutionary computation. *The American Statistician*, 2005, 59(1): 41-46.
[35] Larrañaga P, Kuijpers C M H, Murga R H *et al.* Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 1999, 13(2): 129-170.
[36] Croes G A. A method for solving traveling-salesman problems. *Operations Research*, 1958, 6(6): 791-812.
[37] Pearl J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Francisco, USA: Morgan Kaufmann, 1988.

Concha Bielza received the M.S. degree in mathematics from Complutense University of Madrid, Spain, in 1989 and the Ph.D. degree in computer science from the Technical University of Madrid (Universidad Politécnica de Madrid, UPM), in 1996. Since 2010 she is a full professor of statistics and operations research with the Department of Artificial Intelligence, UPM, Madrid. Her research interests are primarily in the areas of probabilistic graphical models, decision analysis, metaheuristics for optimization, data mining, classification models, and real applications, like biomedicine, bioinformatics and neuroscience.

Juan A. Fernández del Pozo received his M.S. degree in computer science in 1999 and Ph.D. degree in computer science in 2006 from Technical University of Madrid (Universidad Politécnica de Madrid, UPM), Spain. He is currently an associate professor of statistics and operations research at School of Computer Science and a member of the Computational Intelligence Group at the UPM. His research interests are decision analysis and intelligent decision support systems based on probabilistic graphical models, knowledge discovery and data mining on models output for explanation synthesis and sensitivity analysis. He is also interested in optimization based on evolutionary algorithms and classifications models.

Pedro Larrañaga is a full professor in computer science and artificial intelligence at UPM since 2007. He received the MSc degree in mathematics (statistics) from the University of Valladolid and the Ph.D. degree in computer science from the University of the Basque Country ("excellence award"). His research interests are in the areas of Bayesian networks and estimation of distribution algorithms with applications in biomedicine, bioinformatics and neuroscience.