Expert Systems with Applications 37 (2010) 804–815

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Multidimensional statistical analysis of the parameterization of a genetic algorithm for the optimal ordering of tables

C. Bielza [a,*], J.A. Fernández del Pozo [a], P. Larrañaga [a], E. Bengoetxea [b]

[a] Universidad Politécnica de Madrid, Departamento de Inteligencia Artificial, 28660 Boadilla del Monte, Madrid, Spain
[b] Departamento de Arquitectura y Tecnología, Universidad del País Vasco, San Sebastián, Spain

A R T I C L E I N F O

Keywords:
Bertin matrices
Evolutionary computation
Parameterization
Multiple hypothesis testing
Regression tree
Conciseness

A B S T R A C T

The optimal table row and column ordering can reveal useful patterns to improve reading and interpretation. Recently, genetic algorithms using standard crossover and mutation operators have been proposed to tackle this problem. In this paper, we carry out an experimental study that adds to this genetic algorithm crossover and mutation operators specially designed to deal with permutations and includes other parameters (initialization, replacement policy, mutation and crossover rates and stopping criteria) not examined in previous works. A proper analysis of the results must take into account all the parameters simultaneously, since the wrong conclusions can be drawn by studying each separately from the others. This is why we propose a framework for a multidimensional analysis of the results. This includes multiple hypothesis testing and a regression tree that builds a parsimonious and predictive model of the suitable configurations of the parameters.

1. Introduction

In descriptive statistics, rearranging the rows and columns of a table where their ordering is irrelevant reveals interesting patterns that make the table easier to read and interpret. For example, Fig. 1a shows a graphical representation of a 144×128 table, constructed by repeating the original 9×16 table introduced by Bertin (1981) several times. The columns are townships, whereas the rows are characteristics that are present (red[1]) or absent (cream) in the townships. All rows and columns are considered in arbitrary ordering. Fig. 1b contains the same information but this is displayed more clearly after reordering the rows and columns to reveal some patterns.

The clarity of the tables is evaluated with a measure of their conciseness. One way to quantify this conciseness is by considering whether each table entry does not differ much from the values of its neighboring entries. This paper uses the Moore neighborhood. The Moore neighborhood considers eight neighboring entries and a dissimilarity measure to gauge the lack of conciseness that has to be minimized. For the table in Fig. 1a this measure yields the value 64,960, while the table in Fig. 1b reduces this score to 1986.

Some application fields of table rows and columns rearrangement include general elections crossing parties vs. states, olympic medals vs. countries, correlation matrices, microarray data, nutritional components in different foods, economic indicators vs. countries, customer satisfaction in the automobile industry (Niermann, 2005a) and molecular viewing (Liu, Feng, & Young, 2005).

Although different standard multivariate techniques such as principal component analysis (Friendly, 2002), cluster analysis (Banfield & Raftery, 1992) or minimum spanning tree-based algorithms (Friedman & Rafsky, 1979) could be used to tackle the problem, genetic algorithms are a good approach since the problem we face is NP-complete. Note that our problem is like the product of two TSPs (traveling salesman problem), which are known to be NP-complete problems (Johnson & Papadimitriou, 1985).

The methods mentioned above (principal component analysis, cluster analysis, minimum spanning tree) analyze row permutations and column permutations separately, i.e. considering $r! + c!$ configurations, where r is the number of rows and c is the number of columns. This is suboptimal since $r! \cdot c!$ configurations should be considered. This is what Niermann (2005b) does using a genetic algorithm, which is our starting point.

However, some remarks about the design of this genetic algorithm are in order. First, Niermann (2005b) uses a crossover operator that assures that the children inherit the whole part associated with the row ordering from one parent and with the column ordering from the other parent. None of the positions in either of the two orderings (rows and columns) are changed. Therefore, it is up to mutation to introduce diversity into the population. Second, Niermann (2005b) uses the standard 2-opt mutation operator with an unusually high mutation rate (0.5). Here we ask whether crossover

* Corresponding author. Tel.: +34 91 3366596; fax: +34 91 3524819.
E-mail address: mcbielza@fi.upm.es (C. Bielza).
[1] For interpretation of color in Fig. 1, the reader is referred to the web version of this article.

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.05.088

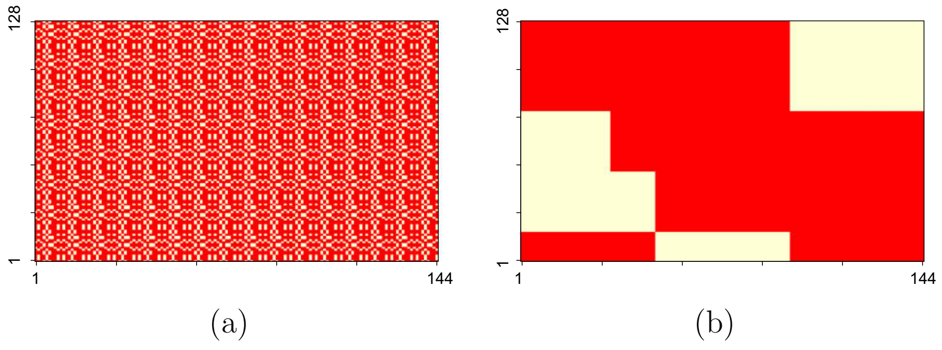


Fig. 1. (a) Original table and (b) table after reordering rows and columns of a Bertin example.

and mutation operators specially designed for orderings are able to outperform the results achieved by the operators used by Niermann (2005b).

Genetic algorithms also have other parameters that influence their performance. They include the initialization of the population starting the evolutionary process, the replacement policy, mutation and crossover rates and stopping criteria. This is explained in Section 2. Due to its stochasticity we run the genetic algorithm several times for each combination of the parameter values.

The experiments are carried out on a table including characteristics (rows) and townships (columns) with a binary response, used by Niermann (2005b). This is then augmented several times to produce more complex tables. Furthermore, we employ a non-binary table including hospitals (rows) and discrete and continuous characteristics (columns), also used by Niermann (2005b). These tables are described in Section 3.1.

We analyze the results at different levels. First, we carry out a unidimensional exploratory data analysis (Section 3.2). Second, we perform a multiple hypothesis testing procedure to simultaneously compare the results of all the parameterizations of the genetic algorithm (Section 3.3). Finally, we propose a predictive model of the objective function based on a regression tree to multivariately locate combinations of subsets of parameters that achieve different ranges of goodness (Section 4).

Whereas the first two levels of analysis have been routinely used in the literature (Gómez & Bielza, 2004; Larrañaga, Kuijpers, Murga, & Yurramendi, 1996; Shilane, Martikainen, Dudoit, & Ovaska, 2008), the regression tree approach is a novel contribution for analyzing the behavior of a genetic algorithm with varied parameters.

Section 5 concludes with a discussion and future research.

2. Parameterizations of a genetic algorithm

Genetic algorithms (Holland, 1975) are heuristics that have performed well at solving difficult optimization problems. Like evolutionary algorithms, they explore the search space evolving a population of individuals, each of them representing a potential solution to the optimization. The quality of an individual is measured with an objective or fitness function.

The main steps of a genetic algorithm are as follows. Firstly, the *initial population* of individuals is chosen, and each individual's fitness function is evaluated. Secondly, parents are selected from the population based on these evaluations. Thirdly, a *crossover operator* is applied for these parents to produce children. The crossover operator has a previously fixed and usually high *crossover probability*. Next, a *mutation operator* is responsible for small modifications to the children that tend to be performed with a near-zero *mutation probability*. Crossover and mutation are needed to explore the search space, whereas mutation is specially designed to avoid local optima. Finally, some individuals are removed from the set containing the population and children according to a *replacement policy*. This yields a final population of the same size as the original one. This whole process is called a generation. These steps are repeated until a *stopping criterion* is met.

In our optimal table ordering problem, an individual is likely to be represented by two linked permutations. The first permutation represents the reordering of the rows and the second, the reordering of the columns. Thus, a table with r rows and c columns is represented by an individual composed of $r + c$ genes, that is, a vector of length $r + c$ given as

$$(\pi^r, \pi^c) = (\pi^r(1), \ldots, \pi^r(r), \pi^c(1), \ldots, \pi^c(c)),$$

where $\pi^r(\pi^c)$ indicates a permutation of rows (columns). Therefore, $\pi^r(i)$ is the position of the row in the original given table that moves to the ith row of the table that the individual represents. The same idea applies to columns. Crossover and mutation operators will be applied to each permutation (rows and columns) separately. With this representation, the search space has $r! \cdot c!$ different points.

The fitness function for evaluating each individual is based on the Moore neighborhood. The Moore neighborhood considers, for each table entry $x_{\pi^r(i),\pi^c(j)}$ 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 $x_{\pi^r(l),\pi^c(m)}$, given by the squared differences:

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

If applied to all table entries we get a global stress measure of the individual (π^r, π^c) :

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

that has to be minimized.

The seven factors that define the parameterization of our genetic algorithm are varied as follows.

2.1. Initial population

The *initial population* is chosen at `random` or using a `heuristic`. `random` means that two random permutations, one for rows and one for columns, are generated whenever the resulting individual has a better fitness function value than the original given table, where this value is used as a threshold to be improved. `heuristic` rearranges the rows first. To do this, it moves rows with a higher sum of their values (more ones in the binary 0/1 case) to the first positions. Then the columns are rearranged, moving those with higher sums (more ones in the binary case) to the left. The fitness function value of the resulting individual is a threshold for applying a `random` initialization as above.

2.2. Crossover operators

As far as the *crossover operators* are concerned, Niermann (2005b) simply chooses to produce two offspring from two parents who have exchanged their respective permutation of columns. This results in 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). However, with the aim of providing crossover-induced population diversity, the other operators used here will include rearrangements of both parts, rows and columns. We borrow the following six operators from the literature on permutation problems like the TSP (see e.g. Larrañaga, Kuijpers, Murga, Inza, & Dizdarevic, 1999): 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 `pmx` operator (Goldberg, 1985) builds an offspring by choosing a substring from one parent and copying the order and position of as many genes as possible from the other parent. If a gene is already present in the offspring, it is replaced according to the mappings created between the genes from that substring and its counterpart in the other parent, both defined by choosing two random cut points on the parent strings. For example, for parents $p_1 = (ABCDEF), p_2 = (EDABFC)$, an offspring is $o_1 = (EBCDFA)$ if the substring consists of the third and fourth genes. The mappings are $C \leftrightarrow A$, $D \leftrightarrow B$. We first copy the substring and $o_1 = (--CD--)$. Then, its first gene would be an E, and the second gene should be a D. But o_1 already has that gene, and the second mapping leads to B being allocated as the second gene. Finally, F is copied into the fifth position, and A is the last gene due to the first mapping. By exchanging the parent roles, a second offspring can be built. In the example, it would be $o_2 = (CDABEF)$.

The `cx` operator (Oliver, Smith, & Holland, 1987) builds offspring by trying to take each gene and its position from one of their parents. For p_1 and p_2 above, we start taking genes from p_1 and we have $o_1 = (A-----)$. Now we look for A in p_2 and it is found in the third position. The third gene from p_1 is C and then we have $o_1 = (A-C---)$. This, in turn, implies $o_1 = (A-C-EF)$. The following movement would lead to A being selected again, completing a cycle. Thus, the remaining genes are taken from p_2 in the same way, to give $o_1 = (ADCBEF)$. Similarly, $o_2 = (EBADFC)$.

Like `pmx`, the `ox1` operator (Davis, 1991) copies a substring from one parent. Then, it tries to preserve the relative order of genes from the other parent. For p_1 and p_2 above and the same substring as in `pmx`, we start with $o_1 = (--CD--)$. Now, as of the second cut point (between the fourth and the fifth gene), genes from p_2 are copied in the same order, provided they are not already present. After reaching the last gene, we continue with the first position. Thus, the sequence to be copied from p_2 is $FEAB$, yielding $o_1 = (ABCDFE)$. Similarly, $o_2 = (CDABEF)$.

The `ox2` operator (Syswerda, 1991) selects several positions in a parent, say in p_2, at random. Next, genes in the selected positions are deleted from p_1. Finally, o_1 is p_1 but its deleted genes are filled in from p_2, following the p_2 order. In our example, suppose the first, second and fourth positions are selected. The corresponding genes in p_2 are E, D and B, in this order. These genes are located at the second, fourth and fifth positions in p_1. Hence, $o_1 = (A-C--F)$. Finally, substring EDB completes the offspring: $o_1 = (AECDBF)$. Similarly and using the same selected positions, $o_2 = (ADBEFC)$.

The `ap` operator (Larrañaga, Kuijpers, Murga, & Yurramendi, 1997) simply builds an offspring by alternately selecting a gene from each parent, whenever it is not already present in the offspring. In the example, $o_1 = (AEBDCF)$ and $o_2 = (EADBCF)$.

The voting recombination crossover operator (`vr`) (Mühlenbein, 1989) is a p-sexual crossover operator ($p \geqslant 2$), such that a gene is copied into the offspring whenever it occupies the same position in at least t ($t \leqslant p$) parents. t is the threshold. The remaining positions of the offspring are filled randomly with the as yet unallocated genes. In our example, also with $p_3 = (AEBDCF), p = 3, t = 2$, the offspring might be $o_1 = (EACBDF)$.

2.3. Crossover probability

The *crossover probability* is usually high. We use probabilities of: 0.85, 0.90 and 0.95. (Niermann, 2005b) sets it as 0.5.

2.4. Mutation operators

As regards *mutation operators*, (Niermann, 2005b) chooses the `2opt` operator (Croes, 1958) applied separately to both the row and the column permutation vectors. This operator reverses the order of the genes between two randomly chosen breakpoints. In this paper, we also borrow 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`).

The `dm` operator (Michalewicz, 1992) selects a substring at random, removes it from the parent and inserts it in a random place. For example, if the chosen substring in p_1 above is BCD, and the random place is after gene E, then $o = (AEBCDF)$. The `ism` operator (Fogel, 1988) is like `dm` but with a substring of length 1. Thus, if gene B is randomly chosen in p_1, and the operator randomly inserts it after gene E, then $o = (ACDEBF)$.

The `em` operator (Banzhaf, 1990) exchanges two randomly selected genes. If these are the second and fourth genes in our example, it results in $o = (ADCBEF)$. The `tim` operator (Lin, 1965) reverses the substrings that fall outside two randomly selected cut points. For example, if the cut points are selected between gene 2 and 3 and between gene 4 and 5, the result is $o = (BACDFE)$.

The `ivm` operator (Fogel, 1993) is like `dm` but the substring is inserted after being reversed. The same example used to illustrate `dm` would result in $o = (AEDCBF)$. The `sm` operator (Syswerda, 1991) selects a substring at random and scrambles its genes. Thus, if substring BCD is chosen in p_1 above, a possible result would be $o = (ACBDEF)$.

2.5. Mutation probability

Although the *mutation probability* is usually near to zero, (Niermann, 2005b) sets it, like the crossover probability, at 0.5. As mentioned above, this high mutation probability is the main element that introduces diversity into the population, since the crossover operator chosen in Niermann (2005b) alters neither the row permutation nor the column permutation. Since we use other less simple crossover operators, we can employ lower mutation probabilities of: 0.10, 0.05 and 0.01, as usual.

2.6. Replacement policies

Three *replacement policies* are proposed to produce the same sized population in the evolved generations as the original one. 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 for the generation with the best individual stored so far. In the third policy, file segment block (`fsb`), the 33% worst ranked individuals are replaced by the 33% best ranked individuals.

Table 1
Parameterization of the genetic algorithm.

Parameter	Possible values	Choice in Niermann (2005b)
Initial population	`random`, `heuris`	`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`

2.7. Stopping criteria

Finally, we consider three *stopping criteria*: stop after a fixed number of generations (256 in our experiments) (`fix`), as in Niermann (2005b); stop after 128 generations where there has been no improvement of the fitness function (`lock`) and stop whenever the coefficient of variation of the *S* stress measure is less than 3 (`var`).

Table 1 shows a summary of the seven parameters that will define the different genetic algorithms to be used in the experiments and the combination of parameters chosen by Niermann (2005b).

Illustratively, Fig. 2 shows how the genetic algorithm typically evolves through the generations. The top graph illustrates the changes in the *S* fitness function values depending on the number of generations run until it reaches stability. The bottom charts are the images of the tables produced at (a) generation 50; (b) generation 150; and (c) generation 250. Note that according to the applied fitness function, a fitter solution implies a row and column ordering that results in a more intuitive visual pattern for interpreting the table data.

3. Analysis of the experiments

3.1. Description of the set of tables

A typical and simple example of table is given in Bertin (1981). The table consists of characteristics (rows) and townships

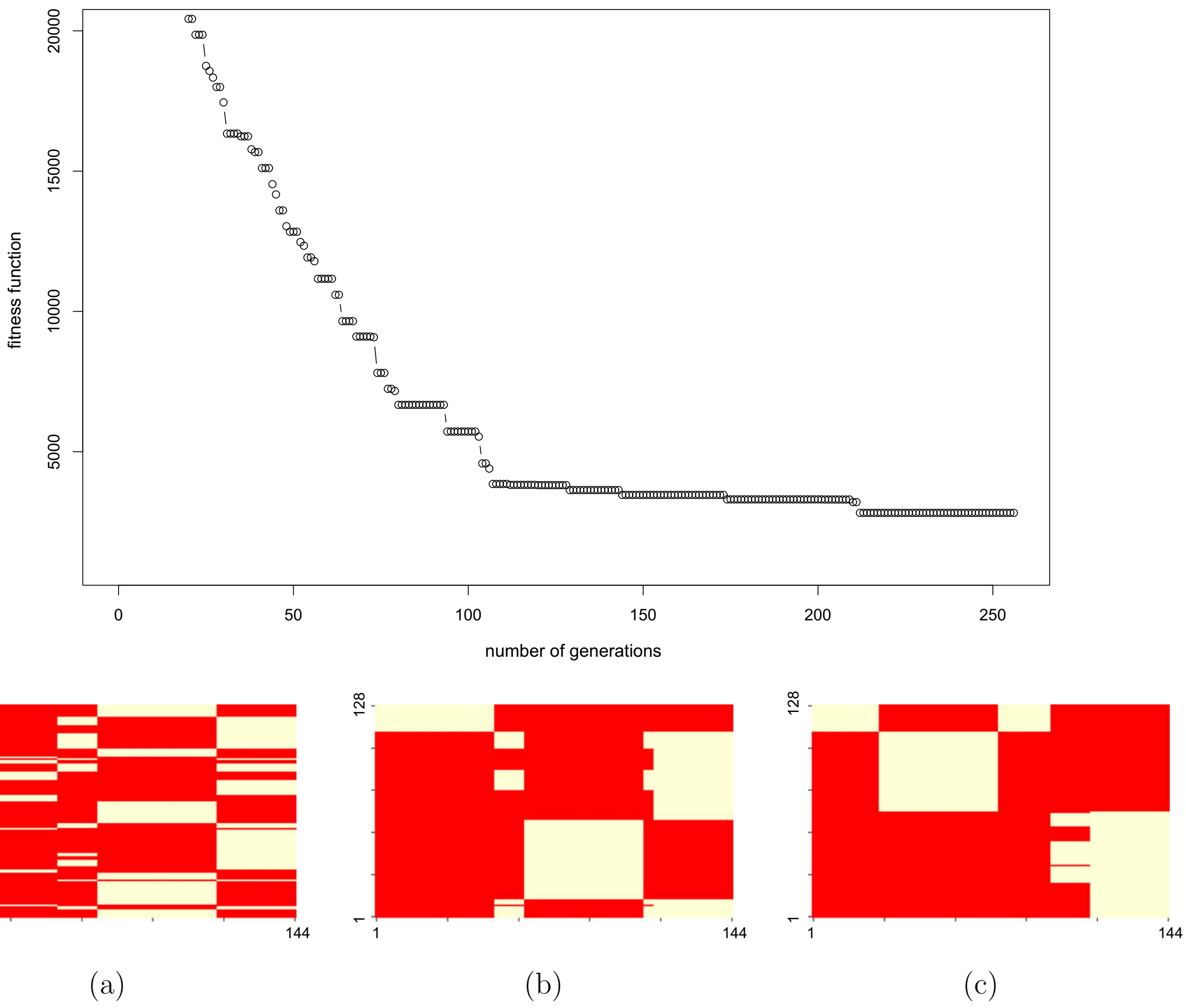


Fig. 2. Evolution of the fitness function vs. the number of generations (top) and tables at (a) generation 50; (b) generation 150; and (c) generation 250 (bottom).

Table 2
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

(columns) with a binary response (present or absent) in arbitrary ordering, see Table 2. This is the table used by Niermann (2005b).

We also use this table (Bertin). With the aim of gaining insights into the scalability of the genetic algorithms, however we augment the table borrowed from Bertin several times. These larger tables are denoted as Bertin2, Bertin4, Bertin8, Bertin32 and Bertin128, see below for their specific dimensions.

As an example of a non-binary table, we take a table about hospitals (rows) and characteristics (columns) that are discrete (number of beds, visits, operations, etc.), continuous (administrative cost) and binary (trauma unit), see Table 3. This table, called here Hospitals, was used by Niermann (2005b) as a simplification of the original table introduced in Cabrera and McDougall (2002). Due to the different scales measuring these variables, they are all rescaled to the unit interval (see Niermann, 2005b for further details). The new rescaled entries of the table are used to compute the stress of any individual derived during the evolution of the genetic algorithm.

Table 4 shows the dimension of our examples and their corresponding stress values S. The last two columns include results that will be analyzed later.

All the genetic algorithm parameterizations considered (see Table 1) give rise to $2 \cdot 7 \cdot 4 \cdot 7 \cdot 4 \cdot 3 \cdot 3 = 14,112$ possible combinations. Due to the stochasticity of genetic algorithms, an experiment will consist of 10 executions of each of the 14,112 algorithms given by a fixed combination of its parameters. This is, for each of the seven examples, 141,120 executions altogether.

Table 3
Characteristics of 24 hospitals.

ID	BEDS	RBEDS	OUTV	ADM	SIR	H95	K95	TR	H96	K96	FEM
1	220	26	0	5954	9500	793	617	0	768	590	146
2	493	0	44,542	12,345	8910	80	47	0	65	49	52
3	962	28	721,546	37,662	23,860	69	155	0	107	219	109
4	788	32	209,145	22,316	7772	108	49	1	97	44	138
5	726	0	58,263	26,029	20,128	132	76	0	150	77	104
6	1048	0	181,245	37,126	22,203	80	0	1	75	0	179
7	565	0	0	18,653	13,468	62	19	0	67	24	37
8	192	0	0	6333	6314	1421	840	0	1373	727	149
9	634	0	133,578	28,164	15,022	178	68	0	173	77	154
10	788	0	0	26,215	2363	126	45	0	133	48	237
11	1071	36	284,564	39,630	28,659	245	68	0	233	70	191
12	1314	41	0	41,493	22,314	239	155	0	232	119	212
13	212	0	9275	7803	10,523	102	284	0	0	0	0
14	829	0	13,250	29,832	25,346	136	62	0	168	53	125
15	103	0	130,767	5441	17,736	0	0	1	0	0	0
16	543	35	307,372	7538	3300	14	13	0	22	20	10
17	1232	52	357,425	24,875	6295	28	12	1	27	10	85
18	152	152	0	1050	0	0	0	0	0	0	0
19	236	0	0	4938	2998	7	10	0	10	3	33
20	575	20	289,584	14,645	4338	16	8	0	21	3	18
21	678	30	544,529	16,071	3722	3	0	1	12	0	52
22	213	0	14,855	8526	7215	29	3	0	22	3	56
23	250	60	0	549	0	0	0	0	0	0	0
24	417	120	0	728	0	0	0	0	0	0	0

Table 4
Characteristics of the set of tables.

Table	Number of rows	Number of columns	Initial S	Best S using configuration in Niermann (2005b)	Our best S
Bertin	9	16	456	154	150
Bertin2	18	16	952	222	222
Bertin4	18	32	1936	346	330
Bertin8	36	32	3952	550	474
Bertin32	72	64	16,096	2010	978
Bertin128	144	128	64,960	15,704	1986
Hospitals	24	11	529	342	299

3.2. Unidimensional analysis of the results

Table 5 includes the mean fitness ($\overline{S}$) and standard deviation (*sd*) of the best fitness found in each of the 10 executions when fixing the parameter value shown, whereas the remaining parameters range over all their possible values. For example, when the value is `random` for the initial population parameter, the mean and standard deviation are computed over $7 \cdot 4 \cdot 7 \cdot 4 \cdot 3 \cdot 3 \cdot 10 = 70,560$ best fitnesses.

Some general observations derived from Table 5 follow, see figures in boldface. First, for all `Bertinn` and `Hospitals` examples, `heuris` provides better results than `random` for the initial population parameter. Second, the best crossover operator is generally `rxc`, then `ox2` and then `pmx`, while surprisingly 0.50 is the crossover probability that behaves best. Third, `tim`, `2opt` and `dm` stand out as mutation operators for small/medium-sized `Bertinn` examples. However, for `Bertin128` and `Hospitals`, `dm`, `2opt` and `em` are better. Fourth, for small/medium-sized examples, mutation probabilities of 0.10 and 0.05 yield better results, whereas for `Bertin32` and `Bertin128` the best value is 0.50 followed by 0.10. Fifth, the best replacement policy is `exs`, then `ets` and, finally, `fsb` in all cases.

Table 5
Mean fitness ($\overline{S}$) ± standard deviation (*sd*) of the best fitness found when fixing the value of one parameter (shown in the rows).

Init. pop.	$\overline{S} \pm sd$	Cross. op.	$\overline{S} \pm sd$	Cross. pr.	$\overline{S} \pm sd$	Mut. op.	$\overline{S} \pm sd$	Mut. pr.	$\overline{S} \pm sd$	Repl.	$\overline{S} \pm sd$	Stop	$\overline{S} \pm sd$
`Bertin`													
`random`	268 ± 78	`rxc`	**231 ± 58**	0.50	**257 ± 71**	`2opt`	**264 ± 78**	0.50	267 ± 82	`ets`	271 ± 72	`fix`	**263 ± 76**
`heuris`	**266 ± 74**	`pmx`	**231 ± 57**	0.85	269 ± 77	`dm`	**266 ± 77**	0.10	**254 ± 77**	`exs`	**226 ± 63**	`lock`	268 ± 75
		`cx`	306 ± 46	0.90	270 ± 77	`em`	273 ± 72	0.05	**263 ± 72**	`fsb`	303 ± 72	`var`	269 ± 77
		`ox1`	259 ± 79	0.95	272 ± 77	`ism`	267 ± 73	0.01	284 ± 69				
		`ox2`	**227 ± 69**			`tim`	**261 ± 79**						
		`ap`	278 ± 69			`ivm`	267 ± 78						
		`vr`	335 ± 74			`sm`	270 ± 72						
`Bertin2`													
`random`	548 ± 207	`rxc`	**415 ± 146**	0.50	**506 ± 193**	`2opt`	**526 ± 209**	0.50	534 ± 222	`ets`	555 ± 193	`fix`	**524 ± 203**
`heuris`	**516 ± 198**	`pmx`	449 ± 167	0.85	537 ± 205	`dm`	**530 ± 206**	0.10	**510 ± 209**	`exs`	**399 ± 160**	`lock`	534 ± 201
		`cx`	685 ± 154	0.90	541 ± 205	`em`	541 ± 191	0.05	**526 ± 194**	`fsb`	642 ± 175	`var`	539 ± 205
		`ox1`	517 ± 206	0.95	545 ± 206	`ism`	529 ± 197	0.01	558 ± 182				
		`ox2`	**431 ± 188**			`tim`	**520 ± 216**						
		`ap`	565 ± 180			`ivm`	532 ± 209						
		`vr`	662 ± 181			`sm`	546 ± 189						
`Bertin4`													
`random`	1065 ± 450	`rxc`	**832 ± 322**	0.50	**986 ± 419**	`2opt`	**1014 ± 454**	0.50	1049 ± 499	`ets`	1085 ± 405	`fix`	**1009 ± 436**
`heuris`	**1000 ± 425**	`pmx`	**879 ± 374**	0.85	1040 ± 445	`dm`	**1020 ± 455**	0.10	**979 ± 450**	`exs`	**707 ± 346**	`lock`	1022 ± 438
		`cx`	1232 ± 337	0.90	1047 ± 444	`em`	1058 ± 403	0.05	**1013 ± 413**	`fsb`	1305 ± 334	`var`	1066 ± 440
		`ox1`	1030 ± 491	0.95	1055 ± 444	`ism`	1033 ± 415	0.01	1087 ± 376				
		`ox2`	**833 ± 428**			`tim`	**1002 ± 478**						
		`ap`	1120 ± 420			`ivm`	1026 ± 461						
		`vr`	1299 ± 409			`sm`	1072 ± 392						
`Bertin8`													
`random`	2274 ± 1023	`rxc`	**1623 ± 764**	0.50	**2048 ± 996**	`2opt`	**2116 ± 1075**	0.50	2144 ± 1147	`ets`	2292 ± 961	`fix`	**2095 ± 1030**
`heuris`	**2031 ± 1028**	`pmx`	1912 ± 998	0.85	2174 ± 1042	`dm`	**2125 ± 1071**	0.10	**2060 ± 1064**	`exs`	**1308 ± 786**	`lock`	2110 ± 1031
		`cx`	2719 ± 831	0.90	2187 ± 1043	`em`	2195 ± 948	0.05	**2115 ± 1001**	`fsb`	2856 ± 658	`var`	2251 ± 1030
		`ox1`	2101 ± 1109	0.95	2199 ± 1042	`ism`	2154 ± 973	0.01	2289 ± 887				
		`ox2`	**1739 ± 992**			`tim`	**2091 ± 1127**						
		`ap`	2357 ± 984			`ivm`	2137 ± 1085						
		`vr`	2613 ± 962			`sm`	2247 ± 925						
`Bertin32`													
`random`	10,340 ± 4082	`rxc`	**7481 ± 3519**	0.50	**9095 ± 4313**	`2opt`	**9366 ± 4532**	0.50	**9184 ± 4965**	`ets`	10,347 ± 3942	`fix`	**9139 ± 4441**
`heuris`	**8604 ± 4483**	`pmx`	**8791 ± 4266**	0.85	9547 ± 4403	`dm`	**9376 ± 4551**	0.10	**9254 ± 4443**	`exs`	**5738 ± 3672**	`lock`	9207 ± 4441
		`cx`	11,439 ± 3408	0.90	9601 ± 4387	`em`	9535 ± 4001	0.05	9474 ± 4137	`fsb`	12,330 ± 2345	`var`	10,070 ± 4173
		`ox1`	9029 ± 4858	0.95	9645 ± 4370	`ism`	9439 ± 4107	0.01	9977 ± 3825				
		`ox2`	**8079 ± 4475**			`tim`	**9242 ± 4863**						
		`ap`	10,453 ± 4217			`ivm`	9425 ± 4576						
		`vr`	11,030 ± 4063			`sm`	9920 ± 3863						
`Bertin128`													
`random`	46,547 ± 12,004	`rxc`	**35,738 ± 13,939**	0.50	**40,522 ± 16,348**	`2opt`	**41,635 ± 17,103**	0.50	**41,800 ± 16,326**	`ets`	45,491 ± 12,642	`fix`	**40,414 ± 14,453**
`heuris`	**38,093 ± 16,034**	`pmx`	40,767 ± 15,029	0.85	42,544 ± 14,627	`dm`	**41,588 ± 17,283**	0.10	**41,935 ± 15,009**	`exs`	**31,382 ± 15,171**	`lock`	41,086 ± 15,312
		`cx`	47,506 ± 11,965	0.90	43,029 ± 14,101	`em`	**41,663 ± 15,480**	0.05	42,249 ± 14,040	`fsb`	50,086 ± 8683	`var`	45,460 ± 14,036
		`ox1`	40,595 ± 15,921	0.95	43,184 ± 13,757	`ism`	41,818 ± 14,795	0.01	43,295 ± 13,547				
		`ox2`	**40,334 ± 14,943**			`tim`	41,810 ± 15,319						
		`ap`	45,165 ± 14,679			`ivm`	43,397 ± 12,324						
		`vr`	46,133 ± 13,094			`sm`	44,329 ± 9266						
`Hospitals`													
`random`	498 ± 58	`rxc`	**390 ± 48**	0.50	**468 ± 62**	`2opt`	**470 ± 71**	0.50	481 ± 63	`ets`	479 ± 63	`fix`	**474 ± 63**
`heuris`	**461 ± 62**	`pmx`	**449 ± 65**	0.85	482 ± 62	`dm`	**473 ± 68**	0.10	**475 ± 68**	`exs`	**467 ± 65**	`lock`	477 ± 63
		`cx`	500 ± 26	0.90	483 ± 62	`em`	**480 ± 61**	0.05	**479 ± 63**	`fsb`	493 ± 58	`var`	488 ± 61
		`ox1`	496 ± 36	0.95	485 ± 63	`ism`	482 ± 60	0.01	484 ± 57				
		`ox2`	492 ± 43			`tim`	483 ± 59						
		`ap`	512 ± 41			`ivm`	485 ± 59						
		`vr`	519 ± 55			`sm`	485 ± 57						

Table 6
Configurations obtained by simply taking the best value for each parameter from Table 5, as separate from the rest. Parameters are ordered as (Init. pop., Cross. op., Cross. pr., Mut. op., Mut. pr., Repl., Stop).

Table	Configuration
Bertin	(heuris, ox2, 0.50, tim, 0.10, exs, fix)
Bertin2	(heuris, rxc, 0.50, tim, 0.10, exs, fix)
Bertin4	(heuris, rxc, 0.50, tim, 0.10, exs, fix)
Bertin8	(heuris, rxc, 0.50, tim, 0.10, exs, fix)
Bertin32	(heuris, rxc, 0.50, tim, 0.50, exs, fix)
Bertin128	(heuris, rxc, 0.50, dm, 0.50, exs, fix)
Hospitals	(heuris, rxc, 0.50, 2opt, 0.10, exs, fix)

Sixth, the stopping criterion values in best to worst order are fix, lock and var.

If we were looking for a favorite configuration from this information, we would simply take the best value for each parameter, as separate from the rest. The final configuration for each table would be as shown in Table 6.

Table 7 follows the same structure as Table 5 except that it includes other performance measures: the best fitness value (S^*) and range (R) rather than the mean fitness and standard deviation.

The results in this new table are more homogeneous, see the repeated S^* values. Only the Bertin128 example shows different

Table 7
Best fitness value (S^*) and range (R).

Init. pop.	$S^*\|R$	Cross. op.	$S^*\|R$	Cross. pr.	$S^*\|R$	Mut. op.	$S^*\|R$	Mut. pr.	$S^*\|R$	Repl.	$S^*\|R$	Stop	$S^*\|R$
Bertin													
random	150|334	rxc	150|228	0.50	150|304	2opt	150|318	0.50	150|318	ets	150|334	fix	150|328
heuris	150|328	pmx	150|252	0.85	150|328	dm	150|328	0.10	150|328	exs	150|254	lock	150|334
		cx	150|264	0.90	150|326	em	150|334	0.05	150|326	fsb	150|326	var	150|326
		oxl	150|256	0.95	150|334	ism	150|322	0.01	150|334				
		ox2	150|256			tim	150|326						
		ap	150|294			ivm	150|320						
		vr	150|334			sm	150|326						
Bertin2													
random	222|758	rxc	222|558	0.50	222|734	2opt	222|742	0.50	222|710	ets	222|742	fix	222|758
heuris	222|722	pmx	222|598	0.85	222|742	dm	222|730	0.10	222|742	exs	222|614	lock	222|738
		cx	222|642	0.90	222|738	em	222|738	0.05	222|734	fsb	222|758	var	222|742
		oxl	222|610	0.95	222|758	ism	222|714	0.01	222|758				
		ox2	222|606			tim	222|758						
		ap	222|650			ivm	222|738						
		vr	222|758			sm	222|742						
Bertin4													
random	330|1636	rxc	330|1222	0.50	330|1570	2opt	330|1634	0.50	330|1524	ets	330|1634	fix	330|1636
heuris	330|1538	pmx	330|1320	0.85	330|1636	dm	330|1566	0.10	330|1636	exs	330|1346	lock	330|1596
		cx	330|1380	0.90	330|1602	em	330|1636	0.05	330|1582	fsb	330|1636	var	330|1634
		oxl	330|1336	0.95	330|1634	ism	330|1582	0.01	330|1634				
		ox2	330|1348			tim	330|1570						
		ap	330|1442			ivm	330|1596						
		vr	330|1636			sm	330|1602						
Bertin8													
random	474|3422	rxc	474|2722	0.50	474|3298	2opt	474|3322	0.50	474|3354	ets	474|3326	fix	474|3386
heuris	474|3214	pmx	474|2874	0.85	474|3370	dm	474|3396	0.10	474|3364	exs	474|2938	lock	474|3364
		cx	474|3054	0.90	474|3358	em	474|3358	0.05	474|3422	fsb	474|3422	var	474|3422
		oxl	474|2860	0.95	474|3422	ism	474|3422	0.01	474|3396				
		ox2	474|2934			tim	474|3354						
		ap	474|3098			ivm	474|3280						
		vr	474|3422			sm	474|3326						
Bertin32													
random	978|14,774	rxc	978|12,824	0.50	978|14,240	2opt	978|14,466	0.50	978|14,358	ets	978|14,774	fix	978|14,774
heuris	978|13,946	pmx	978|13,060	0.85	978|14,602	dm	978|14,346	0.10	978|14,602	exs	978|13,286	lock	978|14,476
		cx	978|13,514	0.90	978|14,774	em	978|14,452	0.05	978|14,474	fsb	1748|13,832	var	978|14,602
		oxl	978|12,938	0.95	978|14,476	ism	978|14,476	0.01	978|14,774				
		ox2	978|13,172			tim	978|14,602						
		ap	978|13,816			ivm	978|14,410						
		vr	978|14,774			sm	980|14,772						
Bertin128													
random	1986|60,918	rxc	1986|57,638	0.50	1986|60,166	2opt	1986|59,818	0.50	1986|60,042	ets	2826|60,078	fix	1986|60,918
heuris	1986|57,420	pmx	1986|55,490	0.85	1986|60,918	dm	2178|59,444	0.10	1986|60,918	exs	1986|55,902	lock	1986|60,166
		cx	1986|57,822	0.90	1986|60,910	em	2826|59,178	0.05	3534|58,380	fsb	13,564|48,588	var	1986|59,928
		oxl	2170|56,374	0.95	1986|59,902	ism	3204|59,582	0.01	2826|59,326				
		ox2	1988|57,384			tim	1986|60,910						
		ap	1986|57,822			ivm	2714|60,190						
		vr	1986|60,918			sm	3280|58,524						
Hospitals													
random	299|369	rxc	301|225	0.50	301|351	2opt	301|359	0.50	301|351	ets	304|371	fix	299|376
heuris	302|373	pmx	299|303	0.85	299|376	dm	299|360	0.10	299|369	exs	299|369	lock	301|367
		cx	308|244	0.90	302|350	em	304|349	0.05	304|356	fsb	311|349	var	301|352
		oxl	305|296	0.95	302|366	ism	304|371	0.01	307|368				
		ox2	304|301			tim	308|343						
		ap	303|314			ivm	302|351						
		vr	305|370			sm	306|362						

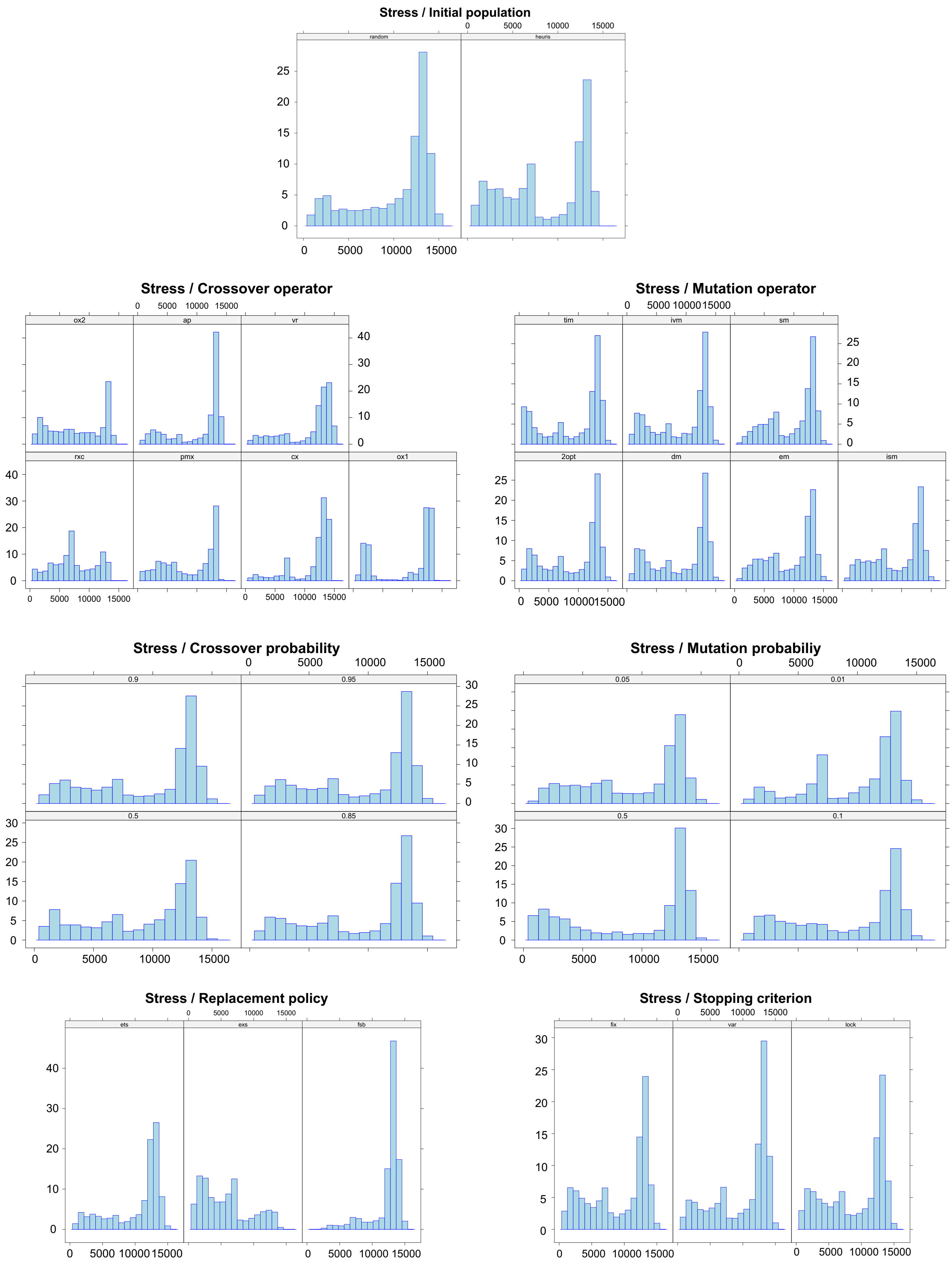


Fig. 3. Histograms (relative frequencies) of the stress measure for `Bertin32` for each of the seven parameters of the genetic algorithms.

best fitness values S^* for the crossover operator (oxl is the worst), mutation operator (2opt and tim are the best while ism and sm are the worst), mutation probability (0.50 and 0.10 are the best) and replacement policy (exs is the best). In Bertin32, the only noteworthy differences are with respect to the replacement policy, where fsb behaves badly and the sm mutation operator almost reaches the minimum value 978. Despite having the same best fitness value when varying the parameter values, the variability is different in some cases, as R indicates. Thus, crossover operator vr has a greater range than its competitors.

Further details on the Bertin32 example are found in Fig. 3, showing the histograms (relative frequencies) of the stress measure for each of the seven parameters of the genetic algorithms. The other examples have similar shapes.

Our initial conclusions based on a simple exploratory analysis are confirmed when we test the null hypothesis on equal distributions of the best fitness across the different values for each parameter. The Kruskal–Wallis test is used because the fitness is not normally distributed as shown e.g. in the histograms. For all the examples and all the parameters, the null hypothesis is rejected with a p-value $p < 0.01$. This means that the fitness behavior is different when each parameter varies regardless of the rest.

3.3. Multiple hypothesis testing

We have so far analyzed the behavior of the genetic algorithm with respect to each parameter univariately, regardless of the values taken by the other parameters. However, if our goal is to find a good combination of all the parameters, it is necessary to carry out a multivariate analysis. In this section, the application of a multiple hypothesis testing, similar to what Shilane et al. (2008) recently proposed in the more general field of evolutionary computation, will compare the configurations with a mean stress (in the 10 executions) better than Niermann's configuration in Niermann (2005b) against Niermann's configuration. Niermann's configuration was shown in Table 1 and is given by (random, rxc, 0.50, 2opt, 0.50, ets, fix). Some genetic algorithm configurations are able to outperform Niermann's, as shown in the last two columns of Table 4, where the solution in Niermann (2005b) is remarkably below par compared with Bertin32 and Bertin128.

For each example, we order all 14,112 possible configurations of parameters with respect to the mean stress $\bar{S}$ in ascending order and count how many of them (m) improve Niermann's. For instance, m is 1359 in the Bertin example. This means that we have m hypotheses to be tested, where each hypothesis test compares the results of a given configuration of parameters against the outcomes of Niermann's configuration. The null hypothesis states that they will behave equally while the alternative hypotesis favors the given configuration.

Thus, we have a set, or family, of hypothesis tests to be carried out simultaneously, i.e. a multiple hypothesis testing. Considering the family as a whole, hypothesis tests that incorrectly reject the null hypothesis are more likely. Many multiple testing procedures have been developed to control the family-wise Type I error rate (FWER) associated with making multiple statistical comparisons. The FWER is defined as the probability of making at least one Type I error. We have chosen the step-down Holm procedure (Holm, 1979) that operates as follows.

Let $p_1, \ldots, p_m$ be the p-values ordered, from the lowest to the highest corresponding to tests with the null hypotheses denoted by $H_1^0, \ldots, H_m^0$, respectively, tested using the Mann–Whitney U statistic. The Holm procedure rejects H_1^0 to H_{i-1}^0 at the α level if i is the lowest integer such that $p_i > \alpha/(m - i + 1)$.

Table 8 shows the results when $\alpha = 0.05$. In this table, m (explained above) is the number of configurations of parameters that are better than Niermann's, $i - 1$ is the number of configurations that are significantly better than Niermann's and the last column shows the configuration of parameters associated with the first (the lowest $\bar{S}$) of the m configurations. Note that the second column indicates that, for the Bertinn examples, approximately 10% of configurations beat Niermann's. However, they are not all statistically significant and there is no regular pattern (see the third column). For Hospitals, only 3% of configurations are better than Niermann's, although 15% of these are statistically significant.

Note also that the top-ranking configurations shown in the last column are not the same for any example as yielded by the unidimensional analysis on the mean fitness value in the previous section (see Table 6). For example, for Bertin, only two of the parameters match: mutation probability (0.10) and replacement policy (exs). Therefore, multivariate parameter analysis is necessary.

Clustering the configurations that passed the Holm test could provide representative prototypes of them all. Opting for three clusters, the three medoids (Kaufman & Rousseeuw, 1990) for each example are given in Table 9.

Table 8
Results of Holm test, where m is the number of configurations of parameters that are better than Niermann's, $i - 1$ is the number of configurations that are significantly better than Niermann's. The last column shows the configuration of parameters associated with the first of the m configurations. The parameters of these configurations are ordered as (Init. pop., Cross. op., Cross. pr., Mut. op., Mut. pr., Repl., Stop).

Table	m	$i-1$	Configuration
Bertin	1359	559	(random, oxl, 0.85, ism, 0.10, exs, var)
Bertin2	1323	0	–
Bertin4	1555	192	(heuris, oxl, 0.50, tim, 0.10, exs, var)
Bertin8	1663	144	(heuris, pmx, 0.85, tim, 0.50, exs, var)
Bertin32	1486	3	(heuris, rxc, 0.85, 2opt, 0.50, exs, fix)
Bertin128	1311	0	–
Hospitals	576	84	(random, pmx, 0.85, 2opt, 0.50, exs, var)

4. Regression tree for finding good parameterizations of the genetic algorithm

The multivariate analysis carried out in the previous section tried to find configurations that are significantly better than Niermann's. Note that these configurations always include values for all the parameters of the genetic algorithm. However, not all the parameters will necessarily affect the performance of the genetic algorithm with certain configurations of the other parameters.

Table 9
Prototypes representing the set of configurations that passed the Holm test, built by three-medoids clustering. The parameters for each configuration are ordered as (Init. pop., Cross. op., Cross. pr., Mut. op., Mut. pr., Repl., Stop).

Table	Configuration
Bertin	(random, oxl, 0.85, ism, 0.10, exs, var)
	(heuris, pmx, 0.90, ivm, 0.10, ets, fix)
	(heuris, rxc, 0.85, tim, 0.10, fsb, fix)
Bertin4	(heuris, oxl, 0.50, tim, 0.10, exs, var)
	(heuris, rxc, 0.85, dm, 0.50, exs, var)
	(heuris, pmx, 0.50, tim, 0.10, ets, lock)
Bertin8	(heuris, pmx, 0.85, tim, 0.50, exs, var)
	(heuris, oxl, 0.50, tim, 0.10, exs, var)
	(heuris, rxc, 0.90, tim, 0.10, ets, lock)
Bertin32	(heuris, rxc, 0.85, 2opt, 0.50, exs, fix)
	(heuris, rxc, 0.95, 2opt, 0.50, exs, lock)
	(heuris, rxc, 0.95, tim, 0.50, exs, fix)
Hospitals	(random, pmx, 0.85, 2opt, 0.50, exs, var)
	(heuris, rxc, 0.95, 2opt, 0.10, fsb, lock)
	(heuris, rxc, 0.85, ivm, 0.50, exs, lock)

Moreover, some values for the same parameters may lead to similar performance. Identifying them can be helpful for saving computational resources. On the other hand, given a significance level, multiple hypothesis testing in some cases fails to show up any configurations, as applies to `Bertin2` and `Bertin128` (see column 3 in Table 8).

Taking into account these limitations –lack of configurations that are significantly better than Niermann's and irrelevance of some parameters (or their values) regarding the performance– an alternative method is necessary. We propose using decision trees adapted for regression problems.

Decision trees are popular and well-known multivariate models for supervised classification. A decision tree is a hierarchical and non-parametric method that divides the input space into local regions identified by a sequence of recursive splits. From the root node downwards, a test is applied at each internal decision node, and one of the branches is taken depending on the outcome. The process is applied recursively until a leaf node is reached. This node contains an output label, the predicted class, which is the same for all instances falling in the region defined by this leaf node.

This idea can be adapted for regression problems where the predicted output is numeric (in our case, the fitness function of the genetic algorithm). The resulting tree is called a *regression tree* (Breiman, Friedman, Olshen, & Stone, 1984). In a regression tree, each leaf stores a value that represents the average output value of instances that reach the leaf. In our case, an instance is given by a configuration of parameters and its corresponding stress value outputted by the genetic algorithm. The splitting criterion used to determine which variable (in our case, the parameter of the genetic algorithm) is the best to split the portion of instances that reaches a particular node is based on maximizing the expected error reduction. The error is measured by the standard deviation. The splitting process terminates when the standard deviation of the instances that reach a node is only a small fraction of the standard deviation of the original instance set. Another stopping criterion is to terminate the growth of the regression tree when a few instances remain.

We have used a recent version of regression trees, called *M5′* (Wang & Witten, 1997), implemented in Weka free software (Witten & Frank, 2005). We set the minimum number of instances for a node to be further split at 1000. This number was chosen to trade-off the size of the tree against the error.

Fig. 4 shows part of the regression tree for `Bertin128`. Specifically, it shows the best eight rules out of a total of 201 rules generated by M5′ in this case. The boxes at the leaf nodes include the average fitness of instances that reach the leaf, whereas the number in parentheses is the number of these instances. Note that this is just 5160 executions (i.e. the sum of all the numbers in parentheses) out of a possible 141,120 executions of the genetic algorithm.

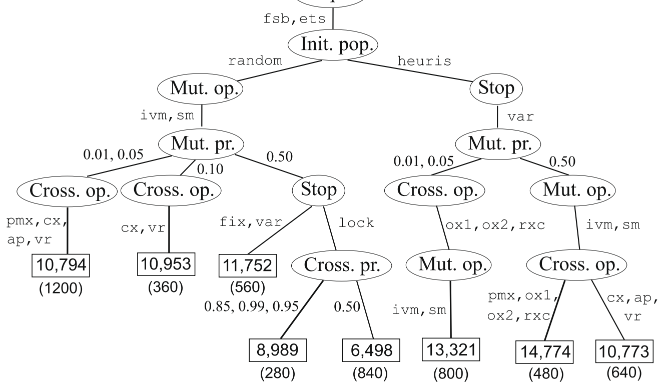


Fig. 4. Part of the regression tree obtained for `Bertin128` showing the best eight rules out of a total of 201 rules generated by M5′.

An analysis of the eight rules indicates that two are composed of only five parameters and the other rules have six parameters, instead of using the whole set with seven. However, the seven parameters are involved in at least one rule. For example, the best rule, with fitness 6498 (840 instances) is "IF Repl.=`fsb,ets` AND Init. pop.=`random` AND Mut. op.=`ivm,sm` AND Mut. pr.=0.50 AND Stop=`lock` AND Cross. pr.=0.50 THEN average fitness=6498." Note that crossover probability is the least frequent parameter appearing in the rules.

Also, note that not all possible values for the seven parameters are chosen in the eight rules. For the mutation operator, the values `2opt`, `dm`, `em`, `ism`, `tim` are never selected and for the replacement policy, the tree never contains the `exs` value.

Let us compare this information with what we have gathered so far from other approaches. A unidimensional analysis of the results for `Bertin128` yielded the following configuration as the best (see Table 6): (`heuris`, `rxc`, 0.50, `dm`, 0.50, `exs`, `fix`). However, the multidimensional analysis provided by M5′ does not find any configuration like that. The closest rule to that configuration has the same values in only three parameters (`heuris`, `rxc` and 0.50 for the mutation probability). A comparison with Niermann's result reveals that his choice (see Table 1) is not included in any of the eight rules. This is another point in favour of a multidimensional analysis of the results. The conclusions are different, and more importantly, the configurations chosen by the multidimensional algorithm M5′ are shorter, thereby identifying irrelevant parameters, which is really useful especially when there are many parameters.

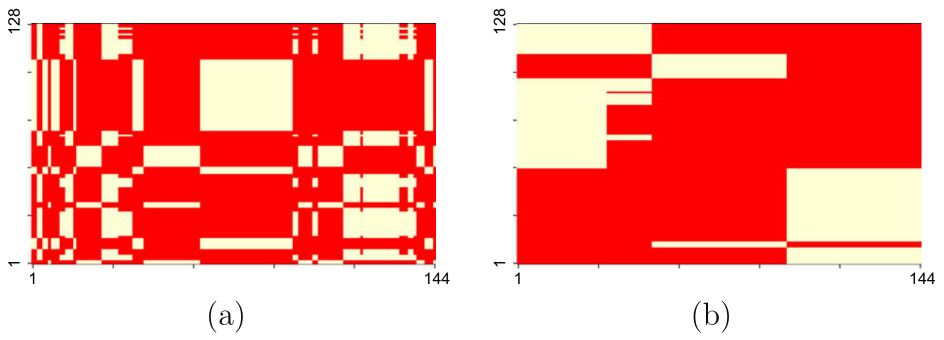
From Fig. 5 it is clear how different some of the solutions provided by the different approaches for `Bertin128` are. Fig. 5a is the best table reported in Niermann (2005b), where $S = 15704$. Fig. 5b is a table built from a simple unidimensional analysis (see Table 6), where $S = 4048$. Fig. 5c shows the table built when the genetic algorithm is run with a configuration *recommended* by the M5′ algorithm tree, given by Repl.=`ets`, Init. pop.=`random`, Mut. op.=`sm`, Mut. pr.=0.50, Stop=`lock` and Cross. pr.=0.50. In this case, $S = 2726$. Finally, Fig. 5d is our best table found after running all the experiments, where $S = 1986$.

When we used the Holm test to try to find configurations significantly better than Niermann's, we did not find any (see Table 8). Therefore, this multidimensional analysis was not very helpful for studying `Bertin128`, and M5′ was a useful alternative here, as we have shown.

Also, the regression tree provides a useful model for determining good configurations of the genetic algorithm parameters. This avoids having to run the algorithm for all the possible configurations of parameters, saving a lot of computational time. Thus, for future tables of similar characteristics to `Bertin128`, we could use that model to decide how to parameterize a genetic algorithm that finds good solutions. For example, the image of a table of the same size as `Bertin128` is shown in Fig. 6a. The fitness function is $S = 40842$. Fig. 6b illustrates the image resulting from using our genetic algorithm parameterized with one of the configurations recommended by the M5′ algorithm tree applied to `Bertin128`, also used in Fig. 5c. Now the fitness function is $S = 2342$. Note that this is a satisfactory solution that is readily obtained without having to run all the possible configurations of parameters in the genetic algorithm exhaustively, as we did for `Bertin128`.

5. Conclusions and future research

Genetic algorithms have proved to be an appropriate technique for rearranging table rows and columns. This is an important



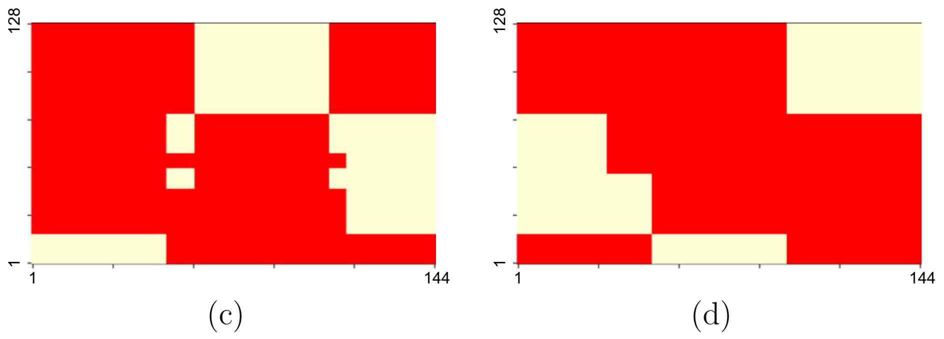


Fig. 5. Best table provided by (a) Niermann (2005b); (b) a unidimensional analysis; (c) M5′ algorithm; and (d) all our experiments.

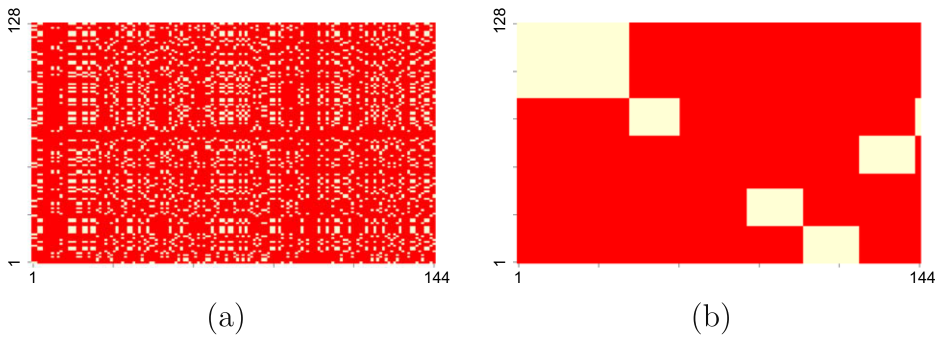


Fig. 6. (a) An image of a table of similar size to `Bertin128`; and (b) output provided by the genetic algorithm run with parameters chosen externally from the M5′ model for `Bertin128`.

problem in descriptive statistics as rearrangement reveals patterns necessary for better table reading and interpretation.

Many parameters beyond just the standard crossover and mutation operators have to be taken into account in a careful design of a genetic algorithm. They include the initialization of the population, the replacement policy, mutation and crossover rates and stopping criteria. The volume of results generated by a genetic algorithm that depends on a considerable number of parameters calls for a thorough analysis.

In this paper, we have extended the below par unidimensional analysis of results to multiple hypothesis testing. Also, applying a regression tree-based approach, used for the first time within this context, we were able to obtain parsimonious and predictive models. These not necessarily binary models are trees that predict the mean fitness of the genetic algorithm when it is run under each configuration of its parameters, shown along the different paths in the tree. Not all the paths are of the same length, leading to parsimony. Also, some values for the same parameters may lead to similar fitness, sharing the same path in the tree. All these features enrich the analysis and the conclusions.

Our experiments with Bertin's example and its augmented versions provided insights into how the genetic algorithm behaves in large dimensions, showing its scalability in this type of problems.

Obviously, the genetic algorithm admits other individual representations and associated parameterizations. For example, we have used a path representation but binary, ordinal or matrix representations are possible alternatives found in the literature related to the traveling salesman problem. Also, rather than using the Moore neighborhood that considers eight neighboring entries making it very complex to compute, other simpler fitness functions, perhaps with fewer neighbors, could be employed.

Finally, other metaheuristics (tabu search, scatter search, ant colony, estimation of distribution algorithms...) rather than a genetic algorithm might be tried.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Education and Science, Grants TIN2007-62626, TIN2008-06815-C02-02 and Consolider Ingenio 2010-CSD2007-00018.

References

Banzhaf, W. (1990). The "molecular" traveling salesman. *Biological Cybernetics, 64*, 7–14.

Banfield, R. D., & Raftery, A. E. (1992). Model-based Gaussian and non-Gaussian clustering. *Biometrics, 49*, 803–822.

Bertin, J. (1981). *Graphics and graphic information processing*. Berlin: Walter de Gruyter.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.

Cabrera, J., & McDougall, A. (2002). *Statistical consulting*. New York: Springer.

Croes, G. A. (1958). A method for solving traveling salesman problems. *Operations Research, 6*, 791–812.

Davis, L. (Ed.). (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

Fogel, D. B. (1988). An evolutionary approach to the traveling salesman problem. *Biological Cybernetics, 60*, 139–144.

Fogel, D. B. (1993). Applying evolutionary programming to selected traveling salesman problem. *Cybernetics and Systems, 24*, 27–36.

Friedman, J. H., & Rafsky, L. C. (1979). Multivariate generalizations of the Wald-Wolfowitz and Smirnov two-sample tests. *The Annals of Statistics, 7*, 697–717.

Friendly, M. (2002). Corrgramms: Exploratory displays for correlation matrices. *The American Statistician, 56*, 316–324.

Goldberg, D. E. (1985). Alleles, loci and the TSP. In *Proceedings of the first international conference on genetic algorithms and their applications* (pp. 154–159). New Jersey: Lawrence Erlbaum, Hillsdale.

Gómez, M., & Bielza, C. (2004). Node deletion sequences in influence diagrams using genetic algorithms. *Statistics and Computing, 14*, 181–198.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics, 6*, 65–70.

Johnson, D. S., & Papadimitriou, C. H. (1985). Computational complexity. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy, & D. B. Shmoys (Eds.), *The traveling salesman problem* (pp. 37–85). John Wiley and Sons.

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data*. New York: John Wiley and Sons.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., & Yurramendi, Y. (1996). Learning Bayesian networks structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics, Part A, 26*, 487–493.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., & Yurramendi, Y. (1997). Decomposing Bayesian networks: Triangulation of the moral graph with genetic algorithms. *Statistics and Computing, 7*, 19–34.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review, 13*, 129–170.

Lin, S. (1965). Computer solutions on the traveling salesman problem. *Bell Systems Technology Journal, 44*, 2245–2269.

Liu, J., Feng, J., & Young, S. S. (2005). PowerMV: A software environment for molecular viewing, descriptor generator, data analysis and hit evaluation. *Journal of Chemical Information and Modeling, 45*, 515–522.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. Berlin: Springer.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings on the third international conference on genetic algorithms* (pp. 416–421). Los Altos, CA: Morgan Kaufmann.

Niermann, S. (2005a). Letters to the editor. *The American Statistician, 59*, 354.

Niermann, S. (2005b). Optimizing the ordering of tables with evolutionary computation. *The American Statistician, 59*, 41–46.

Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators on the TSP. In *Genetic algorithms and their applications: Proceedings of the second international conference* (pp. 224–230). New Jersey: Lawrence Erlbaum, Hillsdale.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In L. Davis (Ed.), *Handbook of genetic algorithms* (pp. 332–349). New York: Van Nostrand Reinhold.

Shilane, D., Martikainen, J., Dudoit, S., & Ovaska, S. J. (2008). A general framework for statistical performance comparison of evolutionary computation algorithms. *Information Sciences, 178*, 2870–2879.

Wang, Y., & Witten, I. (1997). Induction of model trees for predicting continuous classes. In *Proceedings of the poster papers of the ECML* (pp. 128–137). University of Economics, Faculty of Informatics and Statistics, Prague.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.