Controlling behavioral and structural parameters in evolutionary algorithms

Jorge Maturana, Frédéric Lardeux, Frédéric Saubion

Controlling Behavioral and Structural Parameters in Evolutionary Algorithms

Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion

Université d'Angers, France
{maturana,lardeux,saubion}@info.univ-angers.fr

Abstract. Evolutionary algorithms have been efficiently used for solving combinatorial problems. However a successful application rely on a good definition of the algorithm structure and a correct search guidance. Similarly to the majority of metaheuristic methods, the performance of an evolutionary algorithm is intrinsically linked to its ability to properly manage the balance between the exploitation and the exploration of the search space. Recently, new approaches have emerged to make these algorithms more independent, especially by automating the setting of parameters. We propose a new approach whose objective is twofold: (1) to manage an important set of potential operators, whose performances are *a priori* unknown, and (2) to dynamically control the behavior of operators in a evolutionary algorithm, thanks to their probabilities of application.

1 Introduction

Two main phases can be distinguished when implementing Evolutionary Algorithms (EA). The first one is related with the algorithm design, where the components of the algorithm (mainly the operators to be used) are defined. The second phase has to do with the correct execution of the EA, by adjusting its behaviour during the run. Both tasks –design and behaviour control– may be done by means of parameterization. From this perspective, we can intuitively recognize two general classes of parameters: *behavioral* ones, that guides the search without changing the structure of the EA itself (mainly operator application rates or population size), and *structural* parameters, that could eventually turn an algorithm into a different one (e.g., encoding and choice of operators).

Behavioral parameters have been widely studied [8]. Typically, the adjustment of parameters relies on a series of time consuming experiences. This approach leads to *ad-hoc* settings, hardly applicable to other problems. Another approach is to adjust parameter values during the execution of the EA, using a mechanism based on a previously acquired knowledge.

Besides behavioral parameters, the choice of structural components of the EA also requires expertise from the user. Even though standard variation operators (such as uniform crossover or swap mutation) have been conscientiously studied, acceptable performances for applied problems often require the specialization of both the algorithmic scheme and the operators.

In order to asses the control of the algorithm in a general way, its operational process must be abstracted by considering general measures –common to all EAs– that reflect the current state of search. We have previously considered two measures, the average quality of the population and its genetic diversity (entropy) [10] to evaluate the algorithm and to allow the controller to adjust parameter values.

Many previous studies have addressed the problem of parameter setting for evolutionary algorithms. We refer the reader to [8] for a survey. Parameter setting can be addressed, using the taxonomy initially proposed by Eiben et al. [2].

In this paper, we present a method that, in addition to controlling behavioral parameters (i.e., operators application rates), may automate the EA design, by deciding dynamically which operators, from a set of available ones, will be included in the algorithm.

Therefore, our goal is twofold: (1) Show that the controller can autonomously adapt the EA structure, adapting it to the current state of the search and, (2) Show that, using this scheme and without any prior knowledge, we can obtain comparable results as those obtained with an specific algorithm, obtained through an exhaustive analysis of the best possible operators. Experiments will be conducted on the well-known Boolean satisfiability problem (SAT).

2 Controller description

This section presents the general control mechanism, whose architecture is depicted by figure 1. Two main components can be distinguished. The first one is the *Adaptive Operator Selection (AOS)* [9]. AOS communicates with the EA in order to decide which operator would be applied. AOS also receives feedback from the algorithm, in order to update the credit registry, that stores the rewards assigned to each operator. This component performs the control of the behavioral parameters, which correspond to application rates of the operators that are currently available to the algorithm.

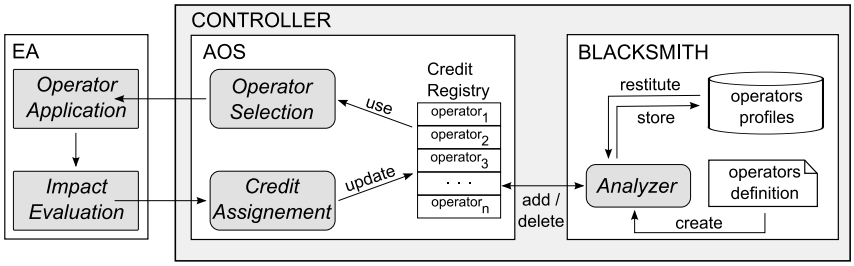

Fig. 1. General scheme of the controller, depicting its two main components, AOS and Blacksmith

The second component, presented in this work, is called *Blacksmith*, and deals with the structural parameters of the EA. Blacksmith is the "operator manager"

that decides which operators will be available to the AOS (and therefore included in the EA) at each moment of the search. The operators are built according to a specification that may result from a combination of different basic parts of an operator (as performed in this work) or simply taken from a list of operator names.

We want to highlight the conceptual difference between the two components of the controller. Blacksmith actually "creates" the EA that the AOS manages. In this approach, the decisions of the AOS depend on those of the Blacksmith.

2.1 Adaptive Operator Selection (AOS)

Given a set of operators, we want to select the best possible ones. However, this "optimal" choice is complex, given the dynamic nature of EAs.

Two main considerations must be taken into account when choosing the operator to apply. On one hand, it is desirable to apply "good" operators, that have evidenced their quality in the recent past. On the other hand, less successful operators must be tried occasionally, in order to discover if they have become useful for the current state of the search. This situation is indeed a typical Exploration vs Exploitation (EvE) dilemma, not at the EA level, but at AOS level. AOS, and more specifically the Operator Selection module, must deal with this problem.

The **Credit Assignment** (**CA**) module measures the performance of the operator after their application. The method *Compass (**C**)* [9, 11] (Figure 2.a), considers two criteria: variation of diversity (ΔD) and variation of quality (ΔQ). The distance between a point ($\Delta D, \Delta Q$), corresponding to the operator o, and a line tilted with an angle of $\Theta = \pi/4$ is measured. The higher and righter a point is located in the plane, the better the corresponding operator will be rewarded.

In this article, two other schemes of evaluation were compared, both based on the concept of Pareto dominance [12]. In an n-dimensional space, we say that a point $a = (a_1, a_2, \ldots, a_n)$ dominates another point $b = (b_1, b_2, \ldots, b_n)$ if $\forall i = 1 \ldots n, a_i$ *is better than* b_i. Here "*better*" means greater than or less than, depending on the nature of the problem (maximization or minimization, respectively). When none of the two points dominate each other, they are said incomparable. In our case, we have a 2-dimensional space ($\Delta D, \Delta Q$) with two criteria that we want to maximize.

The first scheme is called *Pareto Dominance (**PD**)*, and evaluates the number of operators that a given operator dominates (see figure 2.b). The purpose here is to obtain a high value. The second scheme, *Pareto Rank (**PR**)*, measures the number of operators that dominate a given operator (figure 2.b). Here the objective is to obtain low values. Operators with a PR value of 0 are said to belong to the Pareto frontier. There exists an important difference between these two evaluations: whereas PR will prefer only operators which are not dominated, PD rewards also those which are in strong competition with others.

After an operator has been applied, measures of ΔD and ΔQ are sent to the controller. The CA module computes the evaluation (C, PD or PR, depending

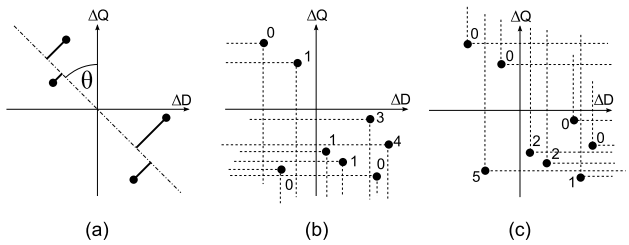

Fig. 2. Credit Assignment schemes. Compass (a), Pareto Dominance (b), Pareto Rank (c)

on the selected scheme), and normalizes the resulting values considering all operators. The normalized values are then stored into the Credit Registry as the assigned rewards. A list of the last m rewards of each operator (corresponding to their last m applications) is recorded in the registry, in order to provide updated historical information about operators performance to the OS module.

The **Operator Selection** (**OS**) module chooses the operator to be applied next, based on information stored in Credit Registry, without neglecting exploration at AOS level. The idea of the OS presented in [9], called Ex-DMAB, is inspired by the methods of multi armed bandits used in game theory. The strategy always chooses the most efficient operator, given by the expression:

$$MAB_{o,t} = r_{o,t} + C\sqrt{\frac{\log \sum_k n_{k,t}}{n_{o,t}}} \tag{1}$$

where $r_{o,t}$ is the reward obtained by the operator o, at current time t, and $n_{o,t}$ is the number of times that the operator o has been applied so far. The left term favors the use of the best operators, whereas the right term favors the operators that have been applied less often. Additionally, the values of $r_{o,t}$ and $n_{o,t}$ are restarted when a change in the behavior of operators is detected, in order to speed up the identification of better operators.

Note that expression 1 relies on the assumption that all operators are present in the EA since beginning. If an operator k is included during the execution, its value of $n_{o,t}$ would be so low that the AOS would be forced to apply it many times to level up the value of $MAB_{k,t}$ with the remaining operators.

Since we are interested in operators that enter and exit the EA along the search process, we have reformulated the expression 1, replacing $n_{o,t}$ by the number of elapsed generations since the last application of the operator (i.e., its idle time). This allows a new operator to level up immediately by applying it once. The new evaluation of performance is then defined as follows:

$$MAB2_{o,t} = r_{o,t} + 2 \times exp(p \times i_{o,t} - p \times x \times NO_t) \tag{2}$$

where $i_{o,t}$ is the idle time of operator o at time t. NO_t is the number of operators considered by the AOS at time t, x expresses how many times NO_t the controller must wait before applying o compulsory.The value of the exploration

component stays close to zero except when $i_{o,t}$ is close to $x \times NO_t$. Since the values of $r_{o,t}$ are normalized in $[0, 1]$, when an operator has not been applied for a long time, its application becomes mandatory. p is a parameter that adjusts the slope of the exponential.

This work compares four different OS modules:

- **Random (R)**, that simply chooses randomly among the operators currently available in the EA.
- **Probability Matching (PM)**, that chooses the operator with a probability proportional to the reward values stored by the CA module.
- **MAB2 (M2)** (already described), that always chooses the operator that maximizes the expression 2
- **MAB2 + Stuck detection (M2D)**, that adds to M2 a method to detect when the population is trapped into a local optima. This checking is achieved by considering the mean quality of the population. The detection is performed thanks to the linear regression of the values of mean quality during the last generations. If the value of the slope is close to zero and the difference between the maximum and minimum values of mean quality is small enough, a period of diversification is started, using only operators that have an exploration profile. This diversification period is maintained until the diversity reaches a range over the original value, while there exist exploration operators, or when a number of generations have passed without being able to reach the desired diversity.

2.2 Blacksmith

As mentioned above, the controller deals with operators that eventually enter and exit the Credit Registry. Blacksmith is the component in charge of the management of these operators, in such way that the EA always benefit from useful operators. Since dismissed operators could eventually be useful in the future, Blacksmith keeps a trace of them. We distinguish the following three main states for the operators :

- **Unborn**: operators that have never been used during the execution of the EA.
- **Alive**: operators that are currently present in the Credit registry and therefore being used by the EA.
- **Dead**: operators that have been discarded from the Credit Registry.

Note that, besides their performance measures, all the information known by the controller about the operators is their name.

We used a simple strategy to manage the operators in the registry. A fixed number of operators is kept in the registry, and they are evaluated at regular intervals. Only operators that have been applied a sufficient number of times can be eliminated. This minimum is required to be rather sure that the operator has a low performance. The weakest of those "known-enough" operators is deleted

and a new one is inserted in the registry. In order to give all operators a chance to show their skills, all unborn operators are tried before Blacksmith starts reviving dead operators. Unborn operators are tried in a random order.

3 Designing a Multi-Crossover EA for SAT

3.1 Evolutionary Algorithms for SAT

The seminal SAT problem [4] consists in finding an assignment that satisfies a Boolean expression. An instance of this problem is a Boolean formula that can be written in conjunctive normal form (CNF), i.e., as a conjunction of clauses where clauses are disjunctions of literals (variables or negated variables). When all the clauses can be satisfied, the problem is called satisfiable. Otherwise, it is interesting to minimize the number of false clauses. EAs for SAT [3, 5, 13, 7] consider individuals as assignments of Boolean variables, being the aim to generate individuals that produce the least possible false clauses. Different operators are used in the EAs: selection, crossover, mutation, insertion, etc.

We used an algorithm based on GASAT [7], which is currently one of the most effective EAs for SAT. Our implementation is a steady-state EA that replaces the oldest individual by a child obtained applying a crossover operator. An initial population is randomly generated. Two individuals are randomly selected and recombined to obtain a new one that is added to the current population. The whole process is repeated until a solution is found or until a fixed maximum number of crossovers is reached.

3.2 A family of crossover operators

We use a wide set of crossovers (307 in total), each one having a predisposition to either improve quality or to promote diversity. Only crossovers that produce a single child from two parents are considered.

Most of the crossovers try to transfer the good properties of the parents to their child. Consider the following crossovers examples.

- **Uniform**: keeps the value of variables that are identical in both parents;
- **FF** [3]: uses the set of clauses that are true in one parent and false in the other. Only the values of the variables appearing true in the clauses of this set are kept;
- **CC** [6]: deals only with clauses which are false in both parents and makes them true in the child by flipping a variable in each one;
- **CCTM** [6]: operates like CC but it also works on clauses that are true in both parents to ensure that the clauses will be also true into the child.

Crossover operators are composed by four basic actions defined by a quadruplet and performed consecutively. The actions are as follows:

A. Selection of clauses that are false in both parents: 1) select none, 2) select one in chronological order, 3) choose one randomly, 4) choose one randomly from the set of smallest clauses, 5) choose one randomly from the set of biggest clauses.

B. Action on each one of the false clauses: 1) do nothing, 2) flip the variable that maximizes the number of false clauses that become true, 3) same as previous one, but also minimizing the true clauses that become false, 4) flip all the variables, 5) flip the literal which appears less often in the others clauses.

C. Selection of clauses that are true in both parents: Same as in **A**.

D. Action on each of the true clauses: 1) do nothing, 2) set to true the variable whose flip minimizes the number of false clauses, 3) set all the literals to true, 4) set to true the literal whose negation appears less often in the other clauses, 5) set all the literals to false.

All variables that remain undefined in the child after the crossover are valued using the uniform crossover process explained before. It must be noted that some quadruplets are not valid (1i** or **1i with $i \in \{2, 3, 4, 5\}$).

4 Experimental Results

4.1 Experimental Setting

We compare different configurations of the controller against the state of the art crossovers, presented in section 3.2 (FF, CC and CCTM). As baseline, we compared also with the Uniform crossover, and with a controller that simply apply one of the 307 possible operators randomly (named *R307*).

We have selected 8 instances from different SAT [14, 1] and Beijing competitions. Benchmarks are selected to cover the different families of instances (random, handmade and industrial). The basic algorithm used in our experiments is applied 50 times for each controller (or state of the art crossover) and instance. We used a population of 100 individuals and the number of crossovers allowed is 10^5.

The objective is to test different combinations of Credit Assignment and Operator Selection mechanisms. These combinations will be identified by the notation $X - Y$, where $X \in \{C, PD, PR\}$ is the AC mechanism, and $Y \in \{M2, R, M2D, PM\}$ is the mechanism of SO.

The parameters of the controller are those of Blacksmith and those of the OSs MAB2 and M2D. Regarding Blacksmith, the registry has a fixed size of 20 operators. Every 50 generations, the Analyzer is invoked in order to find a weak operator and replace it by another. If an operator has been applied a sufficient number of times ($\frac{1}{2}$ of the size of the registry window, i.e., 5 times) and if its reward is in the lower third compared to the other operators, it is discarded. Parameters of M2 are $p = 0.2$ and $x = 1.5$. M2D uses the data of the last 100 generations to calculate the linear regression. Diversification period

is triggered when the value of the slope is within ± 0.0001 and the difference between maximal and minimal values are less than 0.001.

4.2 Discussion

Figure 3 shows the convergence of the best individual for different configurations of controllers and state of the art crossovers for the instance *ibm*.

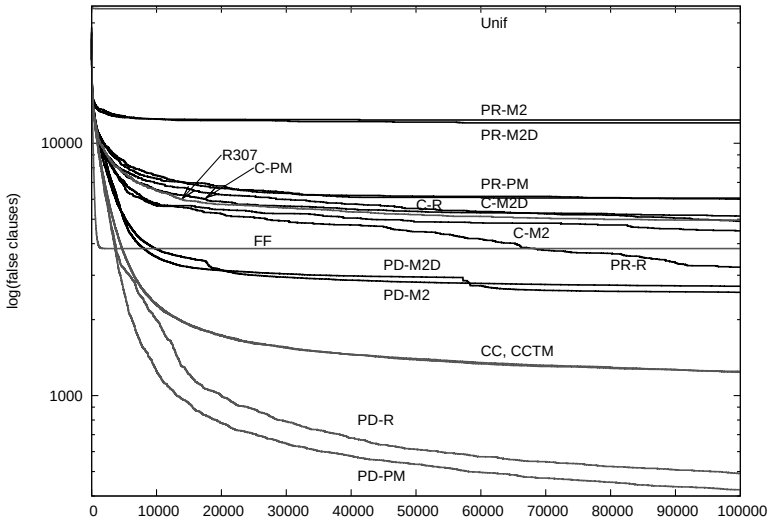

Fig. 3. Number of false clauses of best individual so far, obtained by different controllers and state of the art crossovers, solving the instance *ibm* (mean of 50 runs)

Figure 4.a shows the population diversity produced by the controllers with somewhat similar results (PR-R and PD-M2). Note that the controllers that obtain similar results in terms of quality, do not produce necessarily the same level of diversity. This illustrates two different behaviours: while PD-M2 produces a strong exploitation that improves the quality quickly until generation 30 000, PR-R explores the search space to produce slower yet constant improvements all along the search. Figure 4.b shows the diversity of the population produced by the controllers that produced the best results (PD-PM, PD-R), together with these produced by the state of the art crossovers. An intermediate level of diversity can be observed in the best configurations, mainly due to their exploratory OSs (PM and R), which allow a fast –though prudent– convergence to better results.

The upward trend of diversity for PD-PM starting from the generation 8 000 is due to the inclusion of the diversification criteria in the evaluation of the operators. At the beginning of search, the initial population is composed of randomly generated individuals. This facilitates the task of exploratory operators, which

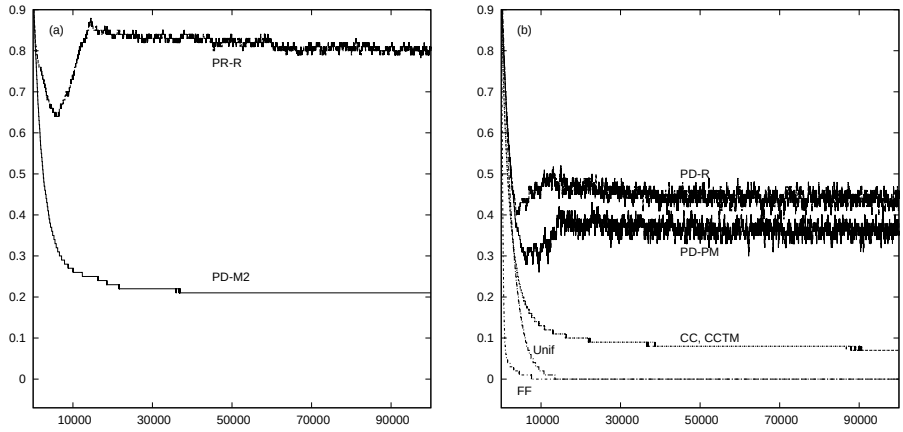

Fig. 4. Diversity of different methods solving the instance *ibm*. (a) that obtain similar results, (b) of better ones, and state of the art crossovers

decrease diversity while increase quality. However, from a given moment, improvement becomes harder, thus the controller turns to exploration, by moving individuals away from local optima. This behavior is precisely the one we sought by including diversity among the measures sent to the controller.

The table 1 shows the average number of false clauses and their standard deviation (in parentheses) over 50 executions of the different controllers, and executions with the state of the art crossovers, Uniform, FF, CC and CCTM. The best results (and those that are indistinguishable from them, using a T Student test with a 95% of confidence) appear in boldface. The best results were obtained by the configurations PD-PM and PD-R, that outperformed the state of the art crossovers in 7 out of 8 instances.

Uniform crossover produced the worst results by far, reason why it will be ignored from here on. One may notice that controllers PD-PM and PD-R produce results comparable with those obtained with the best crossovers without controller (CC and CCTM). This is interesting because the development of CC and CCTM relies on a work of several weeks of comparisons, analyses and experiments [7], while the controller does a similar work in a few minutes (overhead is about 10% of total execution time), and, what is more important, without human intervention. It is also remarkable the performance of controllers PD-PM and PD-R in industrial instances *ibm* and *engine*, where the average number of false clauses is equivalent to at least $\frac{1}{3}$ of those obtained with CC and CCTM.

Since PR considers similarly all the operators placed on the Pareto frontier (points in the figure 2.c with value 0), it induces a balance between exploration and exploitation forces, preventing the EA to lean to one side or the other.A similar behavior could be observed when using Compass, given its performance measure method. By contrast, PD better appreciates the operators which follow the general tendency (points in the figure 2.b with higher values), allowing the EA to break the *status quo*, inclining the search towards exploitation to finally

Table 1. Number of false clauses and standard deviation

	f500		aim-100		ibm		simon	
C-M2	25.9	(24.0)	2.4	(1.9)	6009.7	(3024.6)	189.0	(17.3)
C-M2D	16.8	(19.7)	2.6	(1.9)	6063.4	(2171.8)	194.4	(23.9)
C-PM	56.3	(33.6)	2.2	(1.9)	5151.9	(2758.8)	209.2	(41.8)
C-R	21.0	(14.4)	1.1	(0.2)	4908.4	(1623.0)	183.7	(16.7)
PD-M2	67.4	(62.7)	3.3	(2.2)	2712.0	(3523.9)	94.3	(103.0)
PD-M2D	53.3	(59.0)	2.5	(1.5)	2567.1	(4206.3)	107.9	(102.5)
PD-PM	**6.0**	(1.4)	**1.0**	(0.0)	**423.8**	(75.2)	93.5	(7.7)
PD-R	**5.6**	(1.2)	**1.0**	(0.0)	491.1	(66.9)	102.9	(9.7)
PR-M2	98.2	(53.8)	2.9	(1.7)	12370.3	(5214.2)	201.0	(188.7)
PR-M2D	104.2	(52.5)	2.6	(1.8)	12050.3	(5141.1)	277.9	(200.0)
PR-PM	7.8	(1.4)	**1.0**	(0.0)	4495.9	(791.9)	148.9	(11.9)
PR-R	6.9	(1.1)	**1.0**	(0.0)	3228.6	(913.8)	145.2	(9.2)
R307	49.4	(4.3)	1.0	(0.0)	4962.7	364.9)	187.4	(11.7)
Unif	218.4	(5.7)	11.2	(1.0)	34149.6	(138.5)	2872.6	(33.9)
FF	30.2	(4.9)	1.9	(0.6)	3827.8	(160.5)	137.5	(9.7)
CC	7.2	(1.3)	1.9	(0.6)	1247.7	(98.7)	**81.6**	(5.4)
CCTM	7.3	(1.4)	1.8	(0.6)	1237.2	(78.1)	**81.2**	(5.3)

	bw-large.d		flat200-19		uf250		engine	
C-M2	427.1	(287.1)	54.4	(40.1)	12.3	(13.9)	761.6	(477.3)
C-M2D	596.3	(329.1)	40.7	(37.4)	7.5	(11.5)	1045.0	(369.8)
C-PM	650.5	(816.6)	67.3	(41.9)	21.9	(15.2)	752.8	(490.4)
C-R	306.4	(194.4)	52.5	(28.9)	6.7	(5.8)	577.3	(262.3)
PD-M2	1575.6	(1697.3)	58.3	(44.9)	30.8	(23.0)	838.9	(836.3)
PD-M2D	1553.9	(1804.1)	52.7	(48.4)	26.3	(25.2)	911.4	(824.0)
PD-PM	**78.1**	(3.2)	10.7	(2.1)	2.2	(1.3)	**15.4**	(3.3)
PD-R	83.2	(3.5)	**9.2**	(2.1)	**1.5**	(0.9)	18.4	(3.1)
PR-M2	2455.1	(1678.0)	100.7	(56.7)	41.0	(24.3)	1267.1	(966.1)
PR-M2D	2367.9	(1713.0)	99.8	(54.0)	34.9	(24.0)	1064.7	(964.8)
PR-PM	187.9	(146.9)	31.6	(20.5)	**1.7**	(0.8)	462.1	(328.8)
PR-R	272.5	(268.5)	16.3	(10.5)	**1.5**	(0.5)	415.6	(302.4)
R307	207.5	(22.9)	25.3	(7.2)	17.8	(2.6)	534.8	(54.0)
Unif	27237.3	(301.9)	408.7	(20.6)	98.8	(3.6)	12663.6	(289.1)
FF	126.6	(10.6)	44.9	(4.4)	15.1	(3.4)	465.3	(49.8)
CC	579.0	(17.6)	12.0	(2.1)	3.4	(1.4)	67.9	(12.8)
CCTM	580.3	(17.6)	12.7	(2.1)	3.2	(1.2)	68.9	(11.4)

improve the quality of the population. This "flexible balance" is the main asset of this CA method.

It is interesting to note that the most exploratory OSs (PM and R) produced some of the better results. It could seem surprising –and contradictory with studies in the literature– that a random OS outperforms sophisticated methods that carefully try to balance EvE at the operator selection level.

This can be explained by the following hypothesis. When Blacksmith analyzes the set of operators to replace some of them, it always chooses the worst ones. This is indeed an act of exploitation, based on the rewards stored in the Credit Registry. The fact that we are constantly choosing the operators that will be available to the EA produces a displacement of the exploitation at AOS level from the OS module to Blacksmith. In this new scenario, the OS is mainly in charge of the exploration, which is restricted to the operators that the Blacksmith has allowed.

In order to check the generality of controllers PD-PM and PD-R, we have extended the comparison with the state of the art crossovers over a set of 26 supplementary instances, mixing crafted, random and industrial ones. Table 2 shows the mean and standard deviation (in parentheses) of 25 executions of

Table 2. Comparison of PD-PM and PD-R with state of the art crossovers over crafted (C), random (R) and industrial (I) instances. Number of false clauses and standard deviation.

	PD-PM		PD-R		FF		CC		CCTM	
C1	**35.4**	(5.4)	**34.8**	(2.8)	503.2	(41.0)	44.7	(5.2)	42.1	(4.7)
C2	**35.8**	(2.6)	38.0	(4.2)	509.4	(31.6)	46.0	(4.4)	47.6	(4.9)
C3	**35.4**	(3.7)	**35.6**	(3.6)	490.0	(37.7)	48.4	(4.1)	47.1	(3.3)
C4	45.1	(3.8)	**43.4**	(4.6)	491.6	(36.5)	48.7	(3.0)	48.2	(3.4)
C5	**10.5**	(1.8)	**9.8**	(2.8)	47.9	(4.2)	11.6	(1.8)	10.2	(1.5)
C6	**8.6**	(1.9)	**8.3**	(1.7)	36.9	(3.3)	**8.4**	(1.6)	**8.7**	(1.4)
C7	8.8	(1.8)	**8.0**	(1.9)	38.7	(4.2)	**8.4**	(1.2)	**8.7**	(1.7)
C8	**10.0**	(2.4)	**9.7**	(2.5)	48.2	(4.1)	11.3	(1.4)	11.6	(1.6)
C9	150.9	(31.2)	**123.3**	(28.8)	973.2	(77.4)	214.7	(15.9)	217.0	(14.8)
R1	**7.5**	(1.5)	**7.2**	(1.1)	34.2	(5.4)	9.5	(1.9)	9.7	(1.8)
R2	6.4	(1.3)	**5.7**	(1.4)	30.6	(3.8)	7.3	(1.4)	7.7	(1.6)
R3	**8.4**	(1.4)	**8.2**	(1.5)	32.1	(3.8)	10.6	(1.6)	10.9	(1.9)
R4	4.2	(1.5)	**3.5**	(1.4)	26.3	(3.8)	7.4	(1.2)	7.4	(1.8)
R5	**8.2**	(2.1)	**7.8**	(1.8)	40.0	(6.0)	**8.4**	(1.5)	9.1	(1.4)
R6	**6.7**	(1.6)	**7.9**	(1.6)	44.2	(6.4)	8.7	(1.5)	8.8	(1.4)
R7	**6.1**	(1.7)	**5.8**	(2.1)	39.4	(5.5)	7.6	(1.6)	7.8	(1.4)
R8	**9.0**	(1.2)	**8.8**	(1.6)	49.2	(5.3)	10.3	(1.9)	9.9	(1.7)
R9	**9.1**	(1.6)	**9.0**	(1.7)	41.9	(5.7)	10.0	(1.7)	**9.0**	(1.5)
R10	**110.1**	(5.7)	115.1	(8.3)	654.0	(39.5)	153.0	(9.2)	150.0	(7.9)
I1	**123.6**	(11.4)	167.6	(32.3)	439.3	(27.3)	354.4	(11.4)	349.6	(11.7)
I2	**99.7**	(8.2)	134.7	(22.5)	469.1	(26.3)	372.0	(35.5)	367.8	(32.2)
I3	2.7	(2.9)	8.6	(7.7)	216.5	(18.9)	**1.0**	(0.0)	**1.0**	(0.0)
I4	2.8	(2.4)	6.3	(4.8)	116.2	(11.2)	**1.0**	(0.2)	**1.1**	(0.4)
I5	59.4	(98.5)	**38.0**	(1.4)	12567.6	(547.1)	10044.2	(384.4)	9928.1	(382.0)
I6	127.8	(317.1)	**35.1**	(1.5)	9736.2	(404.9)	7567.7	(238.0)	7521.2	(272.9)
I7	**44.2**	(1.3)	48.4	(2.2)	1877.6	(195.1)	61.8	(1.7)	61.6	(1.8)

40 000 generations each one. The best results (and those that are indistinguishable from them, using a T Student test with a 95% of confidence) are marked in boldface.

PD-R obtains top results on 19 instances and PD-PM in 17, while CC and CCTM does it only 5 times. Even though the controller configurations obtained worst results in two industrial instances, we observe the best improvements on this family of instances, especially on I5 and I6, where the controlled EAs obtained until 260 times less false clauses than the best state of the art crossover. The results presented in this comparison confirm the generality of PD-PM and PD-R on different families and types of instances.

5 Conclusion

In this paper we have presented an autonomous controller that manages an important number of crossover operators in an EA. The control occurs at two levels: (1) at *design* level, deciding which operators will be included and used by the EA, and, (2) at *behavioural* level, choosing, among the available operators, which ones will be applied at each step of the search.

We have tested several controller configurations that deal with an EA that has 307 available crossover operators. The EA solved 34 SAT instances from different families, in order to check the generality of our approach. The results were advantageously compared with state of the art crossovers.

The main contribution of this work is the generic framework that could be applied without major modifications to other operators than crossovers and for

other metaheuristic algorithms. Future work include extending this work in these directions, as well as analyzing results obtained during the run, in order to identify the most successful operators and to detect their relationships.

References

1. D. Le Berre, O. Roussel, and L. Simon. The SAT2007 competition. Technical report, Tenth International Conference on Theory and Applications of Satisfiability Testing, May 2007.
2. A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 3(2):124–141, 1999.
3. C. Fleurent and J. A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652, 1996.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability , A Guide to the Theory of NP-Completeness.* W.H. Freeman & Company, San Francisco, 1979.
5. J. Gottlieb and N. Voss. Adaptive fitness functions for the satisfiability problem. In *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, September 16-20 2000. Springer Verlag. LNCS 1917.
6. F. Lardeux, F. Saubion, and J-K. Hao. Recombination operators for satisfiability problems. In *Artificial Evolution, 6th International Conference, Evolution Artificielle, EA 2003*, volume 2936 of *LNCS*, pages 103–114. Springer, 2004.
7. F. Lardeux, F. Saubion, and J-K. Hao. GASAT: A genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2):223–253, 2006.
8. F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.
9. J. Maturana, E. Fialho, F Saubion, M. Schoenauer, and M. Sebag. Compass and dynamic multi-armed bandits for adaptive operator selection. In *Proceedings of IEEE Congress on Evolutionary Computation CEC*, pages 365 – 372, 2009.
10. J. Maturana and F. Saubion. Towards a generic control strategy for EAs: an adaptive fuzzy-learning approach. In *Proceedings of IEEE International Conference on Evolutionary Computation (CEC)*, pages 4546–4553, 2007.
11. J. Maturana and F. Saubion. A compass to guide genetic algorithms. In G. Rudolph et al., editor, *Proc. PPSN'08*, pages 256–265. Springer, 2008.
12. V. Pareto. Cours d'économie politique. in Vilfredo Pareto, Oeuvres complètes, Genève : Librairie Droz, 1896.
13. C. Rossi, E. Marchiori, and J. N. Kok. An adaptive evolutionary algorithm for the satisfiability problem. In *Proc. of the ACM Symposium on Applied Computing (SAC '00)*, pages 463–470. ACM press, 2000.
14. L. Simon and D. Le Berre. The SAT2005 competition. Technical report, Eighth International Conference on the Theory and Applications of Satisfiability Testing, June 2005.