Autonomous operator management for evolutionary algorithms

Jorge Maturana, Frédéric Lardeux, Frédéric Saubion

Journal of Heuristics manuscript No.
(will be inserted by the editor)

Autonomous Operator Management for Evolutionary Algorithms

Jorge Maturana[1] · **Frédéric Lardeux**[2] · **Frédéric Saubion**[2]

Received: date / Accepted: date

Abstract The performance of an evolutionary algorithm strongly depends on the design of its operators and on the management of these operators along the search; that is, on the ability of the algorithm to balance exploration and exploitation of the search space. Recent approaches automate the tuning and control of the parameters that govern this balance. We propose a new technique to dynamically control the behavior of operators in an EA and to manage a large set of potential operators. The best operators are rewarded by applying them more often. Tests of this technique on instances of 3-SAT return results that are competitive with an algorithm tailored to the problem.

Keywords Parameter control · Adaptive search · Hyper-heuristics · Algorithm design

1 Introduction

Evolutionary algorithms (EA) [19,16,31,26] have been widely used for discrete and continuous optimization problems, covering a wide range of applications. Originally inspired by the principles of natural evolution, evolutionary algorithms manage a set of possible configurations of the problem, which are progressively modified by variation operators, in order to converge to an optimal solution or, at least, to a sub-optimum of good quality. The evolutionary

[1]**Instituto de Informática**
Universidad Austral de Chile
General Lagos 2086, Campus Miraflores
5111187 Valdivia, CHILE
E-mail: jorge.maturana@inf.uach.cl
[2]**Laboratoire LERIA**
Département Informatique
Université d'Angers
UFR Sciences, 2 Boulevard Lavoisier
49045 Angers, FRANCE
E-mail: {lardeux,saubion}@info.univ-angers.fr

metaphor considers configurations as individuals that belong to a population which evolves by means of genetic operators, namely mutation and crossover. Evolutionary algorithms belong to the more general class of metaheuristics [25].

Algorithm Design and Parameters

The performance of evolutionary algorithms relies mainly on the definition of an appropriate encoding of the problem and on the design of efficient operators. Once this structure is defined, the user has to adjust the behavior of the algorithm by means of parameters.

The most straightforward parameters are the operator application rates. These parameters, along with population size, are generally considered as minor modifiers since the algorithm remains essentially unchanged, regardless of their values. Recent advances in parameter setting techniques have lead to handle other parameters, such as parameters related to the encoding, the evaluation and the selection processes, which may modify substantially the algorithm. For instance, Simulated Annealing could be seen as a particular instance of an Evolutionary Algorithm using a one-individual population, without crossover nor mutation, but with specialized local search and reinsertion operators.

It is difficult to assess when –by modifying parameter values– an algorithm becomes another one. Designing and tuning the most suitable algorithm for a given problem is an important issue that has been studied for many years. The algorithm selection problem was formerly defined by John Rice in the 70's [47] and is now at the crossroads of several computer science research areas [52].

Although it is difficult to clearly assess which parameters can lead to an algorithm transformation, we may intuitively distinguish between two general classes of parameters: the *behavioral* parameters (mainly operator application rates or population size) and the *structural* parameters, that could eventually transform an algorithm (e.g., those related with the encoding and the choice of operators). This classification is related to the distinction between *numerical* and *symbolic* parameters pointed out by Smit et al.[51].

Even though the setting of behavioral parameters has been widely studied (see the recent textbook [37]), it remains a particularly difficult task for the user. This setting often relies on empirical rules and/or problem-domain knowledge. Typically, the adjustment of parameters relies on a series of time-consuming experiences. These approaches reduces thus the generality of the obtained values when considering other problems. On the other hand, *control* methods work directly on the values of the parameters while solving the problem, i.e., on-line. Such kind of mechanisms for modifying parameters during an algorithm execution were invented early in EC history, and most of them are still being investigated nowadays. Indeed, there is at least two strong arguments to support the idea of changing the parameters during an EA run:

– As evolution proceeds, more information about the landscape is known by the algorithm, so it should be possible to take advantage of it. This applies to global properties (for example, knowing how rugged is the landscape)

and to local ones (for example, knowing whether a solution has been improved lately or not).
- As the algorithm proceeds from a global (early) exploration of the landscape to a more focused, exploitation-like behavior, the parameters should be adjusted to take care of this new reality. This is quite obvious, and it has been empirically and theoretically demonstrated that different values of parameters might be optimal at different stages of the search process (see [18] and references therein).

In addition to behavioral parameters, the choice of the structural components of the evolutionary algorithm also requires expertise from the user. In many application domains, that directly pertain to standard representations, users who are not EA-experts can simply use off-the-shelf EAs, with classic (and thus non-specialized) variation operators to solve their problems. However, the same users will encounter great difficulties when faced to problems that fall out of the basic frameworks. Even if standard variation operators exist in literature (such as the uniform crossover [53]), acceptable results depend necessarily on the specialization of the algorithmic scheme, which usually requires the definition of appropriate operators. The design of problem-specific operators requires much expertise, though some advanced tools are now available [13]. In any case, the impact on the computation process of problem-specific operators is even more difficult to forecast than those of well-known operators, and thus their associated parameters are harder to be correctly estimated *a priori*.

Hyper-heuristics and Related Works

The problem of finding the best configuration in a search space of heuristic algorithms is related to the more recent notion of Hyper-heuristics [7,9, 11]. *Hyper-heuristics* is a family of methods that aim at automating the process of selecting, combining, generating or adapting multiple simpler heuristics (or components of such heuristics) to efficiently solve computational search problems. Hyper-heuristics, defined as "heuristics to choose heuristics" [10] or "heuristics to generate heuristics" [1], address the problem of finding a (quasi-)optimal solution in the (meta)heuristics search space. This idea was pioneered in the early 60's with the combination of scheduling rules [21,12]. Hyper-heuristics have been widely used for solving combinatorial problems (see Burke et al. [7] for a recent survey).

Burke et al. [8] propose a comprehensive classification of hyper-heuristics considering two dimensions: the nature of the search space and the source of the feedback for learning. They distinguish between heuristics that select heuristics from a pre-existing set and heuristics that generate new and more complex ones, from basic components. Concerning the feedback, they identify three categories: online learning, offline learning and no learning. The distinction between online and offline learning was previously proposed in order to classify parameter setting in evolutionary algorithms [17], differentiating parameter tuning (offline) from parameter control (online).

As classical offline mechanisms, we may mention *portfolio* algorithms [32, 59], where previously acquired knowledge is used in order to select a suitable resolution method for a given problem instance. Hutter et al. [33] have also proposed to use a local search algorithm that automatically finds the correct configuration in a search space composed of solving algorithms.

Online control of algorithms has been fully integrated in evolutionary computation through self-adaptive algorithms [44] and adaptive strategies [55]. Focusing initially on local search algorithms, Battiti et al. [4,3] have also federated a reactive search community around online control. This general paradigm, that aims at producing autonomous algorithms [30], is now emerging mixing several computer science areas, including machine learning, combinatorial optimization and constraint programming [2, 29].

Hyper-heuristics may also be used to discover new heuristics and thus automatically design new solving algorithms. For instance, Fukunaga [23] has used genetic programming to learn new performing heuristics for local search algorithms in order to solve SAT problems.

An Autonomous Approach for Operator Management

In this work, we propose a new approach to build an autonomous EA, that handles parameters that determine its behavior, and also decides which operators will be included in the algorithm. Since both operations are performed online, our approach qualifies as a two-stage online selection hyper-heuristic. In this paper, the operators are generated by composing basic sub-operators, thus this approach could also be classified as a hyper-heuristic that discover new good operators by combination.

The performance of the evolutionary algorithm is assessed by means of measures that evaluate the current state of search. Two well-known criteria are commonly used: *diversification* and *intensification*. Diversification reflects the trend to explore various areas of the search space. Intensification is related to the convergence of the search in a specific area. As a consequence, we use two measures previously introduced to control the balance between intensification and diversification [42], namely the average quality of the population and its genetic diversity.

This measures are considered in order to control the algorithm at two different levels: the *behavioral* level deals with the application of the operators while the *structural* level corresponds to the inclusion and/or deletion of operators, according to their observed performance.

For experimental purposes, we use an evolutionary algorithm that solves the canonical problem of satisfaction in propositional logic (SAT) [24, 49, 6]. The algorithm GASAT [36], proposed a few years ago, has obtained interesting results at the SAT competition in 2004. We have modified this algorithm in order to create more than 300 varieties of crossover operators, that will be managed by our controller.

Our goal is twofold:

– Show that an evolutionary algorithm may select and apply autonomously operators adapted to the current state of the search, thanks to our control mechanism and,
– Show that, without any previous knowledge about the most successful operators, the controller can identify the most successful ones and reach a comparable performance w.r.t. those obtained with the GASAT using hand-crafted operators issued from a long and deep study.

The article is organized as follows. After this general introduction to settle down our approach, some key concepts related to parameter settings for evolutionary algorithms are presented in section 2. A short review about evolutionary algorithms applied to the SAT problem is provided in section 3. The details of our implementation and the experimental settings are presented in sections 4 and 5. Experimental results are discussed in section 6 to finally draw conclusions in section 7.

2 Autonomous Parameter Control for Evolutionary Algorithms

Besides the classification of hyper-heuristics presented in the previous section, another taxonomy of parameters setting can be considered. Many previous studies have addressed the problem of parameter setting for evolutionary algorithms [45]. We refer the reader to an exhaustive survey published recently [37]. In this section, we will focus on the points that are directly related to our method.

2.1 Parameter Setting in Evolutionary Algorithms

Parameter setting can be classified using the taxonomy proposed by Eiben et al. [17], shown in figure 1.

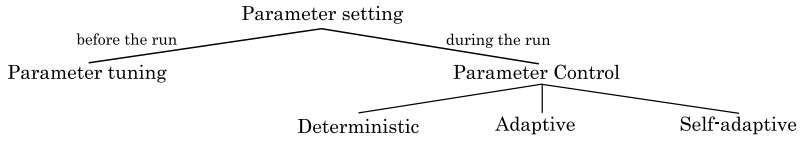

Fig. 1 Control taxonomy proposed by Eiben et al. [17]

In this taxonomy, setting methods are classified depending on whether they attempt to set parameters before the run (*tuning*) or during the run (*control*). The goal of parameter tuning is to obtain parameters values that could be useful over a wide range of problems. Such results require a large number of experimental evaluations and are generally based on empirical observations.

Parameter control is divided into three branches according to the degree of autonomy of the strategies. Control is *deterministic* when parameters are changed according to a previously established schedule, *adaptive* when parameters are modified according to rules that take into account the state of the search, and *self-adaptive* when parameters are encoded into individuals in order to evolve conjointly with the other variables of the problem. In this work, we focus on adaptive control.

2.2 Adaptive Operator Selection

During the last years, Adaptive Operator Selection (AOS) have been used as a generic framework to control parameters of evolutionary algorithms. The main idea (illustrated by figure 2) is to consider the impact of an operator over the search and update a credit registry, that keeps a trace of the success or failure of operators during the last applications. The knowledge stored in this registry is then used to choose the next operator to apply. The update of the registry is done by the *Credit Assignment* module, while the selection of the operator to apply is performed by the *Operator Selection* module. Therefore, adaptive operator selection uses reinforcement learning to control parameters of heuristics.

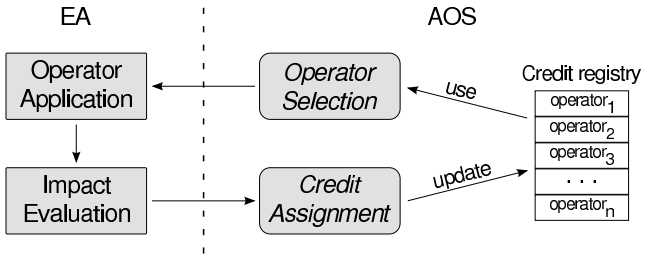

Fig. 2 General approach for the adaptive selection of operators

2.2.1 Credit Assignment

Traditionally, operator evaluation is exclusively based on population quality [55, 20]. This quality can be measured comparing the offspring and their parents [38, 56] or by considering the best [14] or median [34] individuals. It may be interesting to use more sophisticated statistical tools that detect high fitness values in order to reward operators that have a beneficial effect at some specific instant instead of a good average behavior, such as in [57]. The genealogy of the individual (i.e., the fitness of the ancestors), can be also considered [38, 56].

In previous works, we have proposed to use conjointly the average quality of the population and its diversity [41]. These two measures can be handled

differently The main objective is to propose a reward system as generic and as robust as possible. This reward system should also be sensitive enough to variations in the effect of the operators, in order to promptly adapt to the new situation.

2.2.2 Operator Selection

Operator selection selects the operator to apply for the next search step, based on the credits previously assigned. Again, there are many possible choices, such as selecting operators with a probability proportional to their rewards [27], or using more sophisticated models. Fialho et al. [20] use a multi-armed bandit, issued from game theory, to select the operator, ensuring that an operator cannot be infinitely unused. Other approaches include adaptive pursuit [54] or APGAIN [58], where learning and solving stages alternate in order to adjust the operator selection. The main idea is to propose a selection mechanism able to automatically adapt to the current performance of the operators, in order to always use the best suited ones.

3 Evolutionary Algorithms for SAT

This section describes the evolutionary algorithm that will be used by the controller in order to solve the SAT problem, focusing on its crossover operators.

3.1 The SAT problem

The seminal SAT problem [24, 49] consists in finding an assignment that satisfies a Boolean expression. An instance of this problem is a Boolean formula written in conjunctive normal form (CNF), i.e. a conjunction of clauses. Clauses are disjunctions of literals. A literal is a variable or a negated variable. When all the clauses can be satisfied, the problem is said to be satisfiable. Whether the problem is satisfiable or not, it is always interesting to minimize the number of false clauses (MaxSAT). Traditionally, two families of methods can be used to solve this problem: exact methods, which find an answer to the decision problem (i.e., if it is satisfiable or not) and approximate methods that address the optimization problem (i.e., minimize the number of false clauses).

The motivation of considering the SAT problem as testbed is that there exists a large variety of instances, coming from different kinds of problems (random, hand-made and industrial). The instances have then different properties and search landscapes.

3.2 Evolutionary Algorithms for SAT

Evolutionary algorithms for SAT [15, 22, 39, 28, 48, 36] belong to approximate methods. Different operators can be used in evolutionary algorithms: mutation,

crossover and local search operators. Specialized operators are often used in order to improve the performance of these algorithms.

We use an evolutionary algorithm based on GASAT [36]. GASAT is currently one of the most effective evolutionary algorithms for SAT. Its basic principles can be sketched as follows:

```
initialize the population P randomly;
while stop condition is not satisfied do
    identify the subpopulation B of the best individuals in P;
    choose randomly two individuals x,y from B;
    apply the CC crossover over x and y to obtain z;
    perform local search over z;
    if fitness(z) is better than the worst individual in B then
        delete the oldest individual in P;
        insert z in P;
    end
end
```

Algorithm 1: General GASAT formulation

Several stopping conditions can be used, including finding a solution to the problem, or a maximum number of elapsed crossovers, to name a couple.

Since the purpose of our work is to highlight the properties of the controller, GASAT has been slightly modified in order to keep only its basic skeleton and to focus on the effects of the controller. Therefore, the selection is performed randomly and the local search has been removed. The new individuals always replaces the oldest one, regardless of their quality.

3.3 A family of crossover operators for SAT

The design of operators has a great influence over the search. One way to measure their impact is to consider their effect over the diversity of the population. Some operators may induce a concentration of individuals in some specific area of the search space (exploitation), while others may promote exploration by spreading them. These two tendencies are required at different steps of the search, and it is desirable to use appropriate operators. In this work, we define a wide set of crossover operators, that have different effects in terms of Exploration versus Exploitation (EvE). All crossovers produce one child from two parents. Most of these crossovers aim to transfer the good properties of the parents into their descendants. Consider for instance the following operators (note that since an individual represents a Boolean assignment, we say that a clause is true for an individual if the corresponding assignment satisfies the clause):

– The Uniform crossover [53] keeps the value of variables that are identical in both parents

- The FF crossover, proposed by Fleurent and Ferland [22], uses the set of clauses that are true in one parent and false in the other. Only the values of the true literals appearing in these clauses are kept.
- The CC crossover [35] deals only with clauses that are simultaneously false in both parents and turn them into true in the child by flipping a variable in each one of them.
- The CCTM crossover [35] 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.

Most of the time, few crossover operators are designed for diversification purposes. In order to induce different levels of EvE, we have modified several crossovers in order to detect similar properties and break them. Consider for instance, the following possible basic actions:

- A "reverse" uniform crossover, that flips all true values of variables that are identical in both parents
- The Not True Maintenance crossover that deals only with clauses that are true in both parents and makes them false in the child.

We have defined a large set of crossover operators (307 in total), which correspond to combinations of four basic features. In this set, we find some crossovers that are able to diversify, others that will improve quality and finally many of them whose effect lies in between. Most crossovers are built with the following basic features, being defined by a quadruplet as follows:

I. Selection of clauses that are false in both parents:

1. select none
2. select them in chronological order
3. choose randomly one
4. choose randomly one from the set of smallest clauses
5. choose randomly one from the set of biggest clauses

II. Action on each 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 rest of the clauses

III. Selection of clauses that are true in both parents:

1. select none
2. select them in chronological order
3. choose randomly one
4. choose randomly one from the set of smallest clauses

5. choose randomly one from the set of biggest clauses

IV. 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 literals to true
4. set to true the literal whose negation appears less often in the rest of the clauses
5. set all literals to false

All variables that remain undefined in the child are valued using the uniform crossover process explained before. We can obtain many different crossover operators. For instance 2211 corresponds to CC and 2222 to CCTM. Some quadruplets are not valid (1i** or **1i with $i \in \{2, 3, 4, 5\}$)

Finally, the FF crossover is combined with all the basic features previously mentioned, and the reverse uniform crossover is added, obtaining a total of 307 operators.

4 Controller Description

This section presents the general control mechanism, whose architecture is depicted by figure 3. Two main components can be identified:

- *Adaptive Operator Selection (AOS)* [40]: as discussed in section 2, adaptive operator selection communicates with the evolutionary algorithm in order to decide which operator will be applied. Adaptive operator selection also receives feedback from the algorithm in order to update the credit registry, that stores the rewards assigned to each operator. This component deals with behavioral parameters, related with the application rate of the operators that are available to the algorithm at a given time.
- *Blacksmith* is related to the structural parameters of the EA. It constitutes the "operator manager" that decides which operators will included in the adaptive operator selection (and therefore available to the EA) at each step of the search. The operators are built according to a specification or simply taken from a list of operator names.

We want to highlight the conceptual difference between the two components of the controller. Blacksmith designs the evolutionary algorithm that is managed by the Adaptive Operator Selection. Back to the classification of hyper-heuristics [8], adaptive operator selection can be seen as a heuristic to select the suitable operator while Blacksmith is a heuristic to generate operators. In our approach, the choices concerning the adaptive operator selection are thus dependent on those concerning Blacksmith. The next sections will be devoted to the description of these two components.

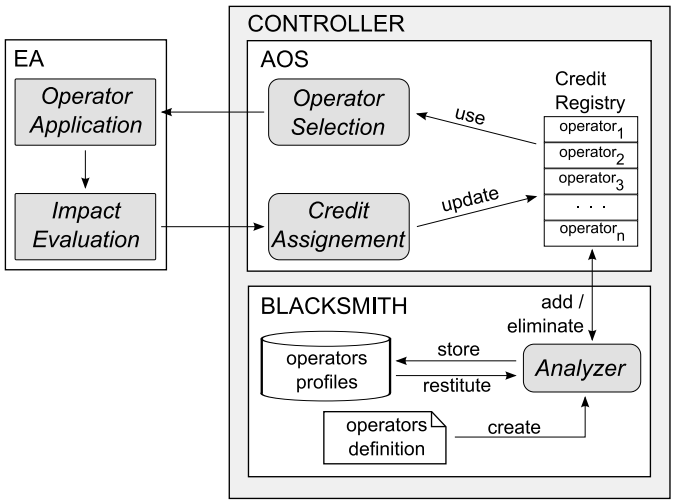

Fig. 3 General scheme of the controller, depicting its two main components, adaptive operator selection and Blacksmith

4.1 Adaptive Operator Selection

Roughly speaking, given a set of operators, we want to apply the best possible ones. However, an optimal choice is difficult because of the dynamic nature of evolutionary algorithms. On the one hand, it is preferable to apply operators that have shown a good performance in the recent past. On the other hand, other operators must be tried occasionally, in order to discover those that could become appropriated to the current state of the search. This situation is indeed a typical Exploration vs Exploitation dilemma, not at the evolutionary algorithm level, but at adaptive operator selection level.

4.1.1 Credit Assignment

As mentioned in section 2, it is necessary to evaluate the performance of the operators after their application. The Compass [43] and ExCoDyMAB [40] methods consider three different criteria: variation of diversity, variation of quality and execution time. $\Delta Diversity$ represents the genotypic diversity variation, according to the chosen diversity measure, and $\Delta Quality$ is the variation of mean quality, according to the fitness function of the problem. The objective is then to maximize the first two criteria and to minimize the third one. Since these objectives may be contradictory, we need to precisely define how to combine these criteria in order to obtain a single evaluation value.

The method *Compass (C)* [43] (Figure 4.a), considers the distance from a point that represents the operator o in the $(\Delta Diversity, \Delta Quality)$ space to a line tilted with an angle of $\Theta = \pi/4$, which correspond to the desired balance between quality and diversity.

In this work, two other schemes of evaluation are compared, both based on the concept of Pareto dominance [46]. 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 a_i *is better than* b_i $\forall i = 1 \ldots n$. Here the word "*better*" is used in the context of the aim of the optimization problem: if we consider a maximization problem in the dimension i, then a dominates b if $a_i > b_i$, on the opposite, if the objective is to minimize, then a dominates b if $a_i < b_i$. When none of the two points dominate each other, they are said incomparable. In our case, we have a 2-dimensional space ($\Delta Diversity, \Delta Quality$) with two criteria that we want to maximize.

The first method, *Pareto Dominance (PD)*, counts the number of operators dominated by each one (see figure 4.b). The best one corresponds to the highest value. The *Pareto Rank (PR)* method counts the number of operators that each operator dominate, (figure 4.b) and lowest values are thus the best ones. Operators with a PR value of 0 belong to the Pareto frontier. There is an important difference between these two evaluations: whereas PR encourage exclusively non-dominated operators, PD also rewards those which are in strong competition with the others.

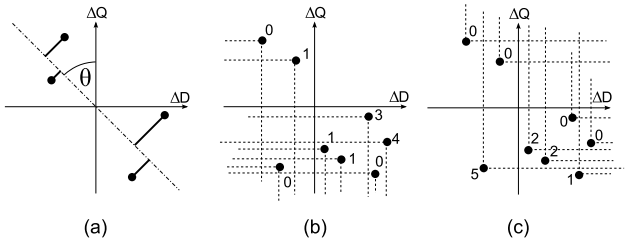

Fig. 4 Credit Assignment Schemes. Compass (a), Pareto Dominance (b), Pareto Rank (c)

After the application of an operator, the values of $\Delta Diversity$ and $\Delta Quality$ are sent to the controller. The credit assignment module computes the evaluation (using C, PD or PR), and normalizes the values w.r.t. all operators. Normalized values are stored into the Credit Registry as the assigned rewards. A list of the last m rewards of each operator (corresponding to its last m applications) is recorded in the registry, in order to provide updated information about the performance of each operator to the operator selection module.

4.1.2 Operator Selection

The operator selection module selects the next operator to be applied. ExDMAB [40] is inspired by the methods of multi armed bandits used in game theory. This strategy always chooses the most efficient operator according to the formula:

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

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 (r_{ot}) favors the use of the best operators, while the right term favors the operators that have been applied less often. Additionally, the values of $r_{o,t}$ and $n_{o,t}$ are reset when a change of the operators' behavior is detected, in order to speed up the identification of better operators.

The expression 1 relies on the assumption that all operators are present in the evolutionary algorithm from the beginning of the run. If an operator is inserted during the execution, its value of $n_{o,t}$ would be so low that it would have to be applied many times to adjust the value of the expression 1 w.r.t. the rest of the operators.

Here we have reformulated the expression 1 in order to deal with a dynamic set of operators. The measure corresponding to the number of times that an operator has been applied is replaced by another criterion that corresponds to the number of generations elapsed since the last application of the operator (i.e., its idle time). The evaluation of a new operator is immediately raised by applying it only once. The new MAB formula is then defined as:

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

where $i_{o,t}$ is the idle time of operator o at time t. NO_t is the number of operators considered by the adaptive operator selection at time t, x expresses how many times NO_t the controller must wait before applying o compulsory. The behavior of the exploration component is highlighted by figure 5. The value stays close to zero except when $i_{o,t}$ is close to $x \times NO_t$. Since $r_{o,t} \in [0, 1]$ (normalization), if an operator has not been applied for a long time then its application becomes mandatory. p is a parameter that adjusts the slope of the exponential.

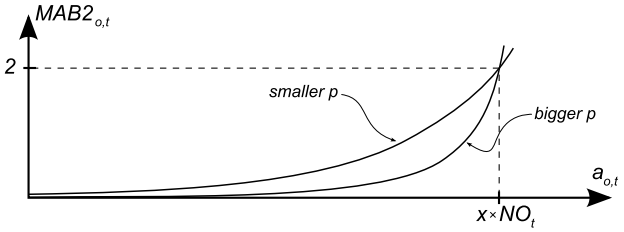

Fig. 5 behavior of exploratory component of expression 2

In our work, we compare the following four different operator selection modules:

- **Random (R)**, that simply chooses randomly among the operators currently available to the EA.
- **Probability Matching (PM)**, that chooses the operator with a probability proportional to the reward values stored by the credit assignment 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 if the population is trapped into a local optimum. This test is performed 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 maximum and minimum values of mean quality is small enough, a diversification stage is decided, carried on by choosing exclusively operators with an exploratory profile. This diversification stage is maintained until the diversity reaches a range over the original value, when there are no exploration operators, or when a number of generations have passed without being able to reach the desired diversity.

4.2 Blacksmith

Blacksmith is the component that manage the inclusion or exclusion of the operators, in order to provide the evolutionary algorithm with operators. Since deleted operators could eventually be useful in the future, Blacksmith keeps a trace of them. An operator may have three main states (figure 6).

- **Unborn**, corresponds to the operators that have never been used during the execution of the algorithm.
- **Alive**, corresponds to operators that are currently in the Credit registry and therefore available to the EA. Operators in this state have two pieces of information attached: the *data*, that corresponds to recent measures of performance, and the *profile*, which summarizes the information in *data*, by calculating meaningful statistics.
- **Dead**, corresponds to operators that have been deleted from the Credit Registry. Dead operators lost their data structure, but keep their profile. A future insertion back into the credit registry (*revival*) would not be performed blindly.

Note that, besides their performance measures, all the information known by the controller about the operators is their name. The controller is independent from the evolutionary algorithm and thus no implementation details are included in the algorithm. The evolutionary algorithm must implement and apply the operators. This architecture ensures the independence of the controller, and its possible use with different evolutionary algorithms without any noticeable change.

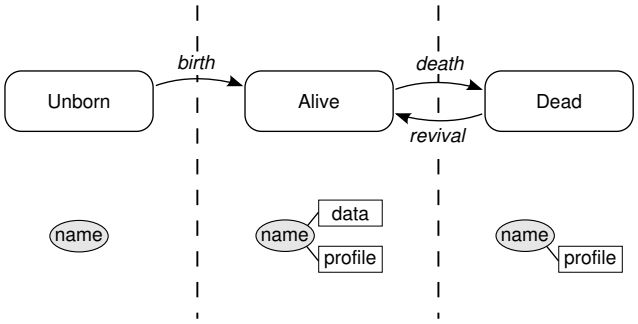

Fig. 6 Operator states and corresponding information associated to operators

Blacksmith controls the structural parameters, deciding whether and when the operators will be available to the EA. As shown in figure 3, its specific tasks, are the following:

- **Create** the operators, according to their definition. This task could be performed by combining different basic features or simply by taking their definition from a list of possible operators. In this work, the combination is done by combining the basic features given in section 3.3.
- **Add** operators to the Credit registry, to make them available to the EA. This corresponds to the *born* transition shown in figure 6.
- **Analyze** the operators that currently belong to the Credit Registry, in order to decide if some of them must die or if a new operator (either unborn or dead) must be (re)inserted to the registry
- **Eliminate** operators from the registry (*death* transition shown in figure 6).
- **Stores** the profile of eliminated operators.
- **Restitute** dead operators back into the Registry when needed (*revive* transition shown in figure 6).

Note that the elimination of an operator does not necessarily mean that it is essentially bad since it is performed according to the current state of the search and the remaining operators in the registry. If the registry contains several good diversification operators, but the search requires an intensification stage, one of them will be deleted. The deleted operator could be useful later and that is why dead operators keep their profile.

We used a simple strategy to manage the operators in the registry. A fixed number of operators is kept in the registry, and evaluated at regular intervals. Operators that have been applied a sufficient number of times are considered for deletion. This condition is required to guarantee 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 inserted in a random order.

5 Experimental Setting

In this section we present the experimental framework to test our method on the resolution of the SAT problem. We compare different configurations of the controller against the state of the art crossovers presented in section 3.3 (i.e., FF, CC and CCTM). As baseline, we use an EA with only Uniform crossover and an EA with a controller that simply chooses randomly one of the 307 possible operators, called *R307*.

5.1 Testbed

In our experiments we have selected 8 instances from different SAT and Beijing competitions [50, 5]. These benchmarks were selected from different families of instances (random, handmade and industrial):

- *f500* and *uf250-010* are randomly generated threshold instances,
- *aim-100-1_6-yes1-1* (noted aim-100 in the following) is a random instance modified to have only one solution,
- *engine_4_nd.cnf* (noted engine in the following) is an instance of very large scale integration (VLSI),
- *ibm-2004-29-k55)* (noted ibm in the following) is an industrial instance of bounded model checking (BMC),
- *simon-s02b-r4b1k1.2* (noted simon in the following) is a difficult instance from SAT competition,
- *bw_large.d* is a planning instance (blocks world),
- *flat200-19* is a graph coloring instance.

The algorithm (see section 1) is applied 50 times for each controller (or crossover) and instance. The population has 100 individuals and the number of allowed fitness evaluations (corresponding, in this case, to crossover applications) is 100 000.

5.2 Controller Meta-Tuning

Our objective is to test different combinations of Credit Assignment and Operator Selection mechanisms introduced in 4.1.1 and 4.1.2. These combinations will be identified by the notation $X - Y$, where $X \in \{C, PD, PR\}$ denotes the Credit assignment mechanism used, and $Y \in \{M2, R, M2D, PM\}$ is the mechanism for operator selection.

The parameters of the controller are the parameters of the Blacksmith and the parameters of MAB2 and M2D. The registry has a fixed size of 20 operators. Every 50 generations[1] , the Analyzer is invoked in order to find a weak operator to replace it by a fresh one. If an operator has been sufficiently

[1] According to the usual taxonomy, this algorithm is a steady state evolutionary algorithm, thus a generation corresponds to the application of one operator

applied ($\frac{1}{2}$ of the size of the registry, i.e., 5 times) and if its reward is in the lower third compared to the other operators, it is selected to be deleted.

The parameters of M2 are $p = 0.2$ and $x = 1.5$. M2D uses the data of the last 100 generations to compute the linear regression. The diversification stage is triggered when the value of the slope is within ± 0.0001 and the difference between maximal and minimal values is less than 0.001.

Of course, it can be argued that we have replaced the original parameters of the evolutionary algorithm by new ones, so the problem of their setting remains the same. We must remark that we are dealing with several hundred of operators, deciding whether they will be included or not in the evolutionary algorithm and when they will be applied. The combinatorial nature of this problem produces a huge number of parameters, compared to the few ones of the controller. Moreover, a bad parameterization of the controller produces less noticeable impact than a bad setting of the original parameters. We must also consider a temporal issue: it is not only a matter of finding the correct set of operators, but also to find them at each step. If we consider the number of operators, their effect on the performance and their impact on the execution time, the few parameters of the controller are justified.

6 Results and Discussion

In this section we present the experimental results, focusing on the most interesting aspects. For the sake of readability, we recall the notations in table 1.

Table 1 Summary of notations

Notation	Description
Credit Assignment	
C	Compass
PD	Pareto Dominance
PR	Pareto rank
Operator Selection	
M2	MAB2
R	Random
M2D	MAB2 + Stuck detection
PM	Probability Matching
Crossover	
Unif	uniform crossover
R307	one crossover is randomly chosen in a set of 307 crossovers
FF	Fleurent and Ferland crossover
CC	Corrective Clause crossover
CCTM	Corrective Clause and Truth Maintenance crossover

Figure 7 shows the convergence of the best individual of the population for different configurations of controllers and state-of-the-art crossovers on the instance *ibm*.

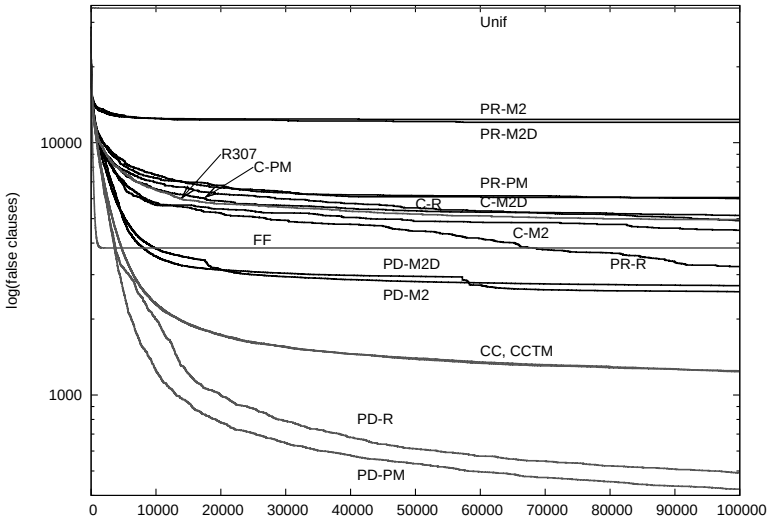

Fig. 7 Number of false clauses of best individual so far, obtained by different controllers and state of the art crossovers, solving the instance *ibm*

Figure 8.a shows the diversity of the population produced by the controllers with quite similar results (PR-R and PD-M2). Note that controllers that obtain similar levels in terms of quality, do not produce necessarily the same level of diversity. We observe different behaviors: while PD-M2 induces a strong exploitation that improves the quality quickly until generation 30 000, PR-R explores the search space and produce slower but constant improvements along the search. Figure 8.b shows the diversity of the population for controllers that obtained the best results (PD-PM, PD-R), together with state of the art crossovers. An intermediate level of diversity can be observed on the best configurations, mainly due to their operator selection methods (PM and R), which allow a fast –though prudent– convergence to better results.

The upward trend of diversity for PD-PM starting from generation 8 000 is due to the diversification criteria in the evaluation of the operators. At the beginning of search, the initial population consists of randomly generated individuals. It favors the exploitation operators, which decrease diversity at the same time that quality increases. However, from a given search step, improvements become harder, thus the controller turns to exploration, aiming at escaping from local optima. This behavior is precisely what we looked for when including diversity in the measures sent to the controller.

This tendency is particularly strong for the credit assignment methods C and RP. Figure 9 shows the evolution of the diversity for different credit

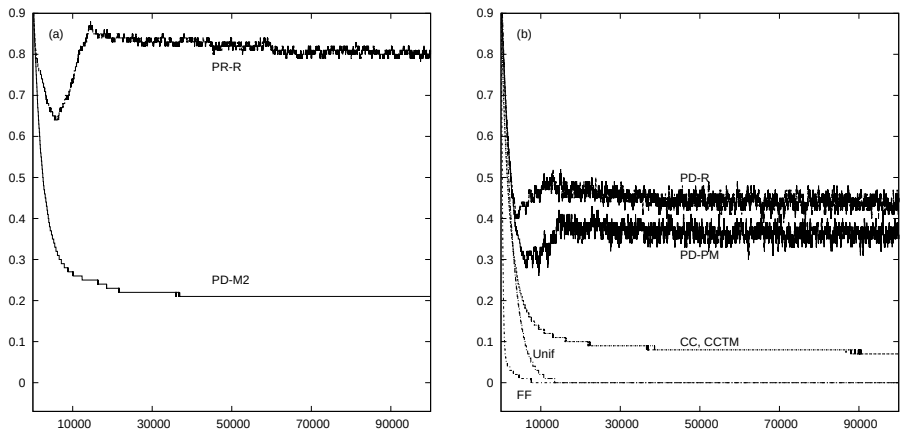

Fig. 8 Diversity of different methods solving the instance *ibm*. (a) that obtain similar results, (b) of better ones, and state of the art crossovers

assignment methods: C, PD and PR, combined with the operator selection R, on the instance *simon*. In this figure, one may remark the excessive exploration induced by C. This could explain why its results are not the best of the series.

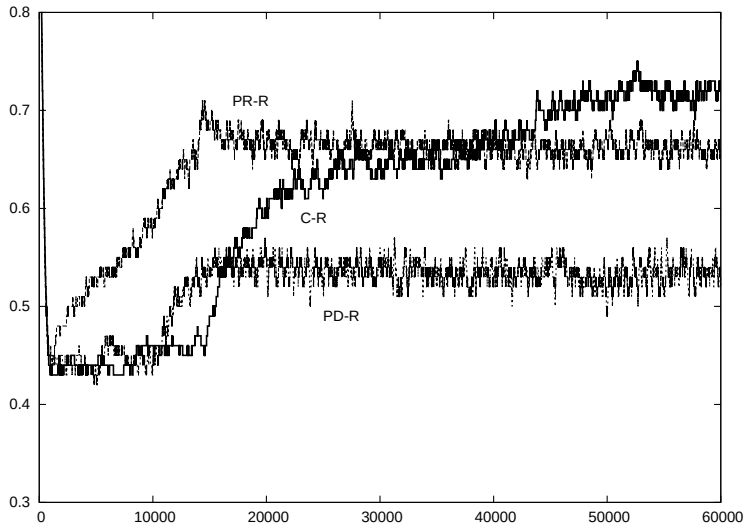

Fig. 9 Diversity of credit assignments C, PD and PR, jointly with operator selection R, showing the shift from exploitation to exploration

Table 2 shows the average number of false clauses and their standard deviation (between brackets) over 50 executions of the different controllers against the state of the art crossovers, Uniform, FF, CC and CCTM. The best results

appear in boldface. We assess the statical significance of the results with a T Student test with an 95% of confidence (therefore, several results can be boldfaced).

Table 2 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)

The best results are obtained by the configurations PD-PM and PD-R, that outperform state of the art crossovers on 7 out of 8 instances. All the results are significantly different. Table 3 shows the percentage of improvement of PD-PM and PD-R, compared to state of the art crossovers.

The uniform crossover produces clearly the worst results and will be ignored in the following experiments. One may notice that controllers PD-PM and PD-R get results comparable with those obtained by the best operators without controller (CC and CCTM). However, let us remark that the design of CC and

Table 3 Improvement of controllers PD-PM and PD-R, compared with Uniform and the state of the art crossovers

PD-PM	f500	aim-100	ibm	simon	bw-large.d	flat200	uf250-010	engine-4-nd
Unif	97.23%	91.07%	98.76%	96.75%	99.71%	97.37%	97.79%	99.88%
FF	80.01%	47.92%	88.93%	31.99%	38.34%	76.09%	85.58%	96.68%
CC	16.57%	47.92%	66.03%	-14.62%	86.52%	10.20%	35.50%	77.25%
CCTM	17.03%	45.05%	65.74%	-15.15%	86.55%	15.17%	31.01%	77.59%

PD-R	f500	aim-100	ibm	simon	bw-large.d	flat200	uf250-010	engine-4-nd
Unif	97.44%	91.07%	98.56%	96.42%	99.69%	97.75%	98.48%	99.85%
FF	81.46%	47.37%	87.17%	25.16%	34.28%	79.51%	90.07%	96.05%
CC	22.22%	47.37%	60.64%	-26.10%	85.63%	23.33%	55.88%	72.90%
CCTM	23.29%	44.44%	60.31%	-26.72%	85.66%	27.56%	53.12%	73.29%

CCTM (the best crossovers) relies on a work of several weeks of comparisons, analysis and experiments [36]. The controller provides a similar performance in a few minutes, without human intervention. We want also to remark the performance of controllers PD-PM and PD-R on industrial instances *ibm* and *engine*, where the average number of false clauses is equivalent to at least $\frac{1}{3}$ of those obtained by CC and CCTM.

When comparing the different credit assignment methods, we note that PD is used in the most efficient controllers, followed by C and PR. In order to compare the two controllers based on the notions of Pareto dominance and to understand why PD performs better than the two others, it is necessary to study their behavior during individual executions. Figure 10 shows the average quality of the populations using PD-M2, PR-M2 and C-M2 on the instance *ibm*.

PR considers all the operators placed on the Pareto frontier equally (points in figure 4.c with value 0). This induces a balance between exploration and exploitation and prevents the evolutionary algorithm to lean to one side or to the other. Note that the attempts to increase the quality of PR-PM are moderated by this balance, forcing the average quality to come back to an intermediate value. A similar behavior could be observed when using Compass, according to its performance measure method. On the other hand, when using PD, the better evaluation of the operators, which follow the general tendency (points in figure 4.b with higher values), allows the evolutionary algorithm to break the *status quo* and finally to improve the quality of the population. This "flexible balance" is the main asset of this credit assignment method.

Interestingly, the most exploratory operator selection methods (PM and R) have produced some of the best results. It could seem surprising –and contradictory with studies in literature– that a random operator selection could be able to outperform sophisticated methods that carefully try to balance EvE at the operator selection level. A possible hypothesis for these good results is that the mix of crossovers works better than applying a single one. However, the poor results obtained by R307 prove that this is not the only reason.

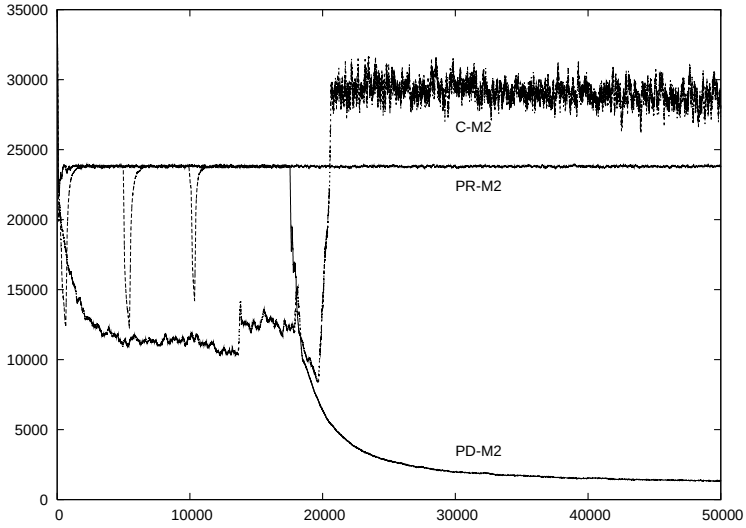

Fig. 10 Behavior of Pareto Rank vs. Pareto Dominance: Average number of false clauses in the instance *ibm*

When the Blacksmith analyzes the set of operators to replace some of them, it always chooses the worst ones. This corresponds to exploitation, based on the rewards stored in the Credit Registry. This choice of operators produces a displacement of the exploitation at EvE level from the operator selection to the Blacksmith. In this new scenario, the operator selection is only in charge of the exploration, which is restricted to the operators allowed by the Blacksmith. In this AOS+Blacksmith scheme, building credit assignment and structural control modules is more gainful than designing operator selection mechanisms.

In order to check the generality of the controllers PD-PM and PD-R, we have extended the comparison with the state of the art crossovers on a set of 26 new instances, mixing crafted, random and industrial instances presented in table 4. The column *Not.* gives the notations used for the instances in the next tables.

Table 5 shows the mean and standard deviation (between brackets) of 25 executions of 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 boldfaced. The last row shows the number of instances for which each controller or crossover has obtained the best results of the series.

PD-R obtains top results on 19 instances and PD-PM on 17, while CC and CCTM are better only 5 times. Even though controller configurations obtained the worst results on two industrial instances, we observe the best improvements on this family of instances, especially on I5 and I6, where the controlled evolutionary algorithms obtained up to 260 times less false clauses than the best state of the art crossover. The overhead of this improvement

Table 4 SAT instances used for algorithms comparisons

Name	Not.	Nb Var.	Nb Clauses
Crafted instances			
ezfact64_3.sat05-450.reshuffled-07	C1	3073	19785
ezfact64_4.sat05-451.reshuffled-07	C2	3073	19785
ezfact64_5.sat05-452.reshuffled-07	C3	3073	19785
ezfact64_6.sat05-453.reshuffled-07	C4	3073	19785
hgen2-v500-s1216665065.sat05-467.reshuffled-07	C5	500	1750
hgen3-v400-s344840348.sat05-470.reshuffled-07	C6	500	1750
hgen3-v400-s553296708.sat05-471.reshuffled-07	C7	500	1750
hgen3-v500-s1349121860.sat05-473.reshuffled-07	C8	500	1750
pyhala-braun-unsat-40-4-02.sat05-459.reshuffled-07	C9	9638	31795
Random instances			
unif2p-p0.9-v630-c2280-S1071799860-07-UNSAT	R1	630	2280
unif2p-p0.9-v630-c2280-S1244126495-18-SAT	R2	630	2280
unif2p-p0.9-v630-c2280-S1501024241-13-UNSAT	R3	630	2280
unif2p-p0.9-v630-c2280-S1788789488-19-SAT	R4	630	2280
unif-k3-r4.261-v650-c2769-S1089058690-02.SAT.shuffled	R5	650	2769
unif-k3-r4.261-v650-c2769-S1159448555-06.SAT.shuffled	R6	650	2769
unif-k3-r4.261-v650-c2769-S1172355929-14.SAT.shuffled	R7	650	2769
unif-k3-r4.261-v650-c2769-S1341479044-12.UNSAT.shuffled	R8	650	2769
unif-k3-r4.261-v650-c2769-S1470952774-07.SAT.shuffled	R9	650	2769
unif-k3-r4.2-v10000-c42000-S1173369833-06	R10	1000	42000
Industrial instances			
AProVE07-03	I1	3114	10827
AProVE07-21	I2	3189	11039
eq.atree.braun.13.unsat	I3	2010	6802
eq.atree.braun.10.unsat	I4	1111	3756
vmpc_26	I5	676	86424
vmpc_24	I6	576	67872
sortnet-6-ipc5-h11-unsat	I7	27724	95880

is reasonable since the control represents less that 10% of the total execution time.

Until now, we have used a simplified evolutionary algorithm to better appreciate the performance of our controller. Compared to the original GASAT, our algorithm selects parents randomly from the whole population and does not perform mutation or local search on the obtained children. Since the final goal of our study is to create a controller that work in real conditions, we have restored the missing features of GASAT, selecting parents from the subset of the better individuals and using local search over the child for improving it[2]. However, we still ignore the condition of insertion (because it only accepts individuals with high quality, rejecting those with high diversity which can be provided by the controller in a diversification phase) and the intensity of the local search (10 steps instead of 1000), because of execution time constraints.

Table 6 shows the mean and standard deviation (between brackets) of 25 executions of 40 000 generations each one, using the complete algorithm. The best results are marked in boldface. In this scenario controllers keep their

[2] This corresponds to a so-called memetic algorithm, that mixes evolutionary algorithms and local search

Table 5 Comparison of PD-PM and PD-R with state of the art crossovers over crafted, random and industrial 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)
Wins		17		19		0		5		5

advantage mainly in crafted instances, while random and industrial ones (with a couple of meritorious cases, I5 and I6) are dominated by state of the art crossovers.

The loss of performance of controllers can be explained by the elitist configuration of the complete GASAT configuration. The fact that GASAT chooses the breeders from the subpopulation of the better individuals, along with the application of local search, restrict the attempts of the controller to diversify the population. As explained in a former research [36], GASAT uses several operators (selection, crossover, local search, condition of insertion) and its performance relies on the simultaneous use of all of them.

The results obtained by controllers using the simplified configuration of GASAT are better than those obtained with the complete version. Table 7 shows a comparison of PD-PM and PD-R using the simplified GASAT, and CC and CCTM using the complete version. The names of the best solved instances (using a T Student test with a 95% of confidence) and their number is shown for each configuration.

Table 6 Comparison of PD-PM and PD-R with state of the art crossovers over crafted, random and industrial instances, using an algorithm near of the complete GASAT implementation. Number of false clauses and standard deviation.

	PD-PM	PD-R	FF	CC	CCTM
C1	**126.5** (9.5)	135.1 (15.8)	1139.9 (51.9)	144.3 (9.8)	136.8 (13.3)
C2	**128.8** (10.8)	131.1 (9.4)	1159.0 (53.2)	136.8 (9.2)	136.3 (10.13)
C3	**129.3** (14.8)	136.0 (11.4)	1136.1 (49.0)	143.3 (10.9)	137.6 (10.6)
C4	**128.9** (9.8)	**129.7** (11.2)	1148.8 (58.0)	140.7 (9.5)	140.6 (10.75)
C5	18.8 (3.0)	19.3 (2.8)	111.0 (11.2)	**17.7** (2.0)	**17.6** (3.45)
C6	**13.4** (2.8)	14.8 (3.0)	88.5 (10.8)	**13.8** (1.8)	**13.4** (2.81)
C7	**13.9** (2.4)	**13.9** (2.5)	87.7 (12.4)	**13.9** (2.8)	**13.6** (2.15)
C8	18.2 (2.9)	**17.6** (2.8)	108.2 (10.4)	**17.1** (2.3)	18.1 (2.25)
C9	504.5 (28.6)	511.4 (30.5)	2923.2 (213.7)	451.6 (18.7)	**442.9** (17.2)
R1	18.9 (3.6)	18.2 (3.2)	100.4 (13.6)	**17.4** (2.3)	**16.8** (1.85)
R2	16.1 (2.5)	17.2 (3.5)	100.4 (16.6)	**14.9** (2.9)	**14.4** (3.21)
R3	18.4 (2.3)	18.8 (2.1)	112.2 (16.0)	**16.2** (2.5)	17.3 (2.41)
R4	13.9 (2.8)	14.3 (2.9)	91.9 (18.4)	**12.8** (2.7)	**12.2** (2.91)
R5	19.6 (4.5)	19.8 (3.2)	138.6 (15.4)	17.9 (2.5)	**16.4** (2.61)
R6	20.8 (4.3)	20.4 (4.1)	136.3 (14.7)	**18.4** (3.3)	**17.9** (2.87)
R7	20.3 (4.6)	20.2 (4.5)	134.4 (15.2)	**16.6** (2.7)	17.8 (3.29)
R8	23.8 (3.6)	25.5 (5.8)	143.0 (13.8)	22.4 (3.0)	**20.8** (3.79)
R9	20.6 (5.2)	19.6 (3.6)	137.6 (14.2)	**18.0** (2.3)	**18.1** (2.6)
R10	387.3 (58.6)	378.3 (31.4)	2343.5 (250.6)	288.5 (19.4)	**277.4** (19.41)
I1	173.5 (18.0)	173.0 (11.8)	937.4 (67.6)	**143.6** (8.1)	149.7 (7.95)
I2	161.7 (16.1)	166.2 (17.7)	992.9 (56.7)	130.4 (8.2)	**126.2** (8.42)
I3	90.3 (10.8)	93.5 (15.0)	576.6 (37.9)	**64.6** (5.6)	**65.4** (5.75)
I4	48.6 (8.8)	50.6 (10.3)	317.4 (19.6)	**39.3** (4.4)	**38.4** (6.11)
I5	**11.5** (2.4)	**10.9** (2.4)	12686.0 (353.3)	9627.4 (407.2)	9800.4 (467.41)
I6	**10.2** (2.1)	**10.2** (1.7)	9784.3 (291.9)	6135.9 (723.4)	5591.9 (1729.97)
I7	486.1 (415.1)	652.0 (402.1)	7935.0 (645.2)	**71.5** (4.3)	**71.7** (4.04)
Wins	8	5	0	15	16

Table 7 Comparison of PD-PM and PD-R using the simplified version of GASAT, with CC and CCTM controllers, using a more complete version

Version	Controller/Crossover	Successful instances	Wins
simplified GASAT	PD-PM	C1, C2, C3, C5, C6, C8 R1, R3, R5, R6, R7, R8, R9, R10 I1, I2, I3, I4, I7	19
	PD-R	C1, C3, C4, C5, C6, C7, C8, C9 R1, R2, R3, R4, R5, R7, R8, R9 I5, I6	18
complete GASAT	CC	—	0
	CCTM	—	0

7 Conclusions

We have presented a controller to handle two generic kind of parameters. *Behavioural* parameters tune the application of different operators in the algorithm, while *structural* parameters design the algorithm by deciding which operators will be included in the algorithm at each step of the search.

Our controller has two modules : the *Adaptive Operator Selection* receives feedback from the evolutionary algorithm in order to update the credit registry, which is used later to select the operator to apply. Therefore, the registry characterizes the set of operators that are currently available to the EA. The second module, called *Blacksmith*, decides which operators –from a wide set of available ones– will be included in the EA, based on their observed performance and the needs of the search.

The main contribution of this work is thus the generic framework that addresses the problems of selecting which operator to apply and of designing the best suited algorithm at each step of the search. The controller is implemented independently from the algorithm, setting a simple and minimal interface between them. The performance evaluation measures are related to two high-level criteria (exploration and exploitation) that are common to many search metaheuristics. Such a controller could be easily plugged into another metaheuristic algorithm.

We have also defined two interesting Credit Assignment methods, based into the notions of Pareto-dominance (PD and PR), which handles the compromise of several criteria according to the general tendencies of the operators. We have modified a previous Operator Selection method (MAB2) in order to deal with a dynamic set of operators.

We have tested twelve controllers, resulting from the combination of three Credit Assignment and four Operator Selection modules for the resolution of the satisfiability problem SAT. More than 300 crossover operators were delivered to the controller, and the results were compared with state of the art crossovers. Comparisons on instances from different families and types of benchmarks (crafted, random-generated and industrial) have highlighted a clear advantage for two of the controller configurations (PD-PM and PD-R). In order to measure exploration and exploitation, we have used population diversity and mean fitness, respectively. We used a very simple scheme for the Blacksmith, that keeps a fixed-length set of operators. Given the importance of the control of the structural parameters

From a set of 34 instances, PD-PM has obtained the best results on 27 instances, similar results on 4 and worst results only on 3. For the same number of instances, PD-R has obtained the best results on 28, similar on 3 and worst on 3. Using our controller in an algorithm very near of the state of the art evolutionary algorithm GASAT, the results are not good because this algorithm is developed to only improve the quality.

Although we have used only crossover operators for SAT as source of structural parameterization, this schema could be extended to other operators and problems without too much effort. The extension of our method to other population based approaches, including single-individual approaches, where the population diversity could be replaced by a temporal diversity, in order to evaluate the variation of the visited configurations.

In order to use the controller as a tool for the design of algorithms, a mechanism could be included in the controller to identify the most successful operators. It would be also interesting to detect the relationships between

operators. Note that we have not identified a unique best crossover and the good performance of PD-PM/R may rather be due to the joint effect of several crossovers.

On most of the instances, we have noticed a stabilization of diversity and quality levels, from generation 30 000. This situation suggests that we could obtain some improvements by incorporating a search strategy [41]. This strategy could include additional criteria in order to better guide the exploration and exploitation of the search space.

Acknowledgements We would like to thank warmly the anonymous reviewers for their helpful comments and remarks.

References

1. M. Bader-El-Den and R. Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In *8th International Conference, Evolution Artificielle, EA 2007. Revised Selected Papers*, number 4926 in Lecture Notes in Computer Science, pages 37–49, Tours, France, 2008. Springer.
2. R. Battiti and M. Brunato. *Learning and Intelligent Optimization Second International Conference, LION 2007 II, . Selected Papers*, volume 5313 of *LNCS*. Springer, 2008.
3. R. Battiti and M. Brunato. *Handbook of Metaheuristics (2nd edition)*, chapter Reactive Search Optimization: Learning while Optimizing. Springer, 2009. In press.
4. R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations research/Computer Science Interfaces*. Springer Verlag, 2008.
5. 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.
6. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
7. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. A survey of hyper-heuristics. Technical Report Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham, Computer Science, 2009.
8. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. *Handbook of Meta-heuristics*, chapter A Classification of Hyper-heuristics Approaches,. 2009. to appear.
9. E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Handbook of Meta-heuristics*, chapter Hyper-heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer, 2003.
10. P. Cowling and E. Soubeiga. Neighborhood structures for personnel scheduling: A summit meeting scheduling problem (abstract). In E. K. Burke and W. Erben, editors, *proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Constance, Germany, 2000.
11. P. I. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Applications of Evolutionary Computing, EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.
12. W. Crowston, F. Glover, G. Thompson, and J. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. Technical report, ONR Research Memorandum No. 117, GSIA, Carnegie-Mellon University, Pittsburg, PA, 1963.
13. L. Da Costa and M. Schoenauer. GUIDE, a Graphical User Interface for Evolutionary Algorithms Design. In Jason H. Moore, editor, *GECCO Workshop on Open-Source Software for Applied Genetic and Evolutionary Computation (SoftGEC)*. ACM Press, 2007. Software available at http://guide.gforge.inria.fr/.

14. L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
15. K. A. De Jong and W. M. Spears. Using genetic algorithm to solve NP-complete problems. In *Proc. of the 3rd International Conference on Genetic Algorithms (ICGA'89)*, pages 124–132, Virginia,USA, 1989.
16. K.A. De Jong. *Evolutionary computation: a unified approach*. MIT Press, 2006.
17. A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 3(2):124–141, 1999.
18. A.E. Eiben, Z. Michalewicz, M. Schoenauer, and J.E. Smith. *Parameter Setting in Evolutionary Algorithms*, chapter Parameter Control in Evolutionary Algorithms, pages 19–46. Volume 54 of Lobo et al. [37], 2007.
19. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.
20. A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In G. Rudolph et al., editor, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2008.
21. H. Fisher and L. Thompson. *Industrial Scheduling*, chapter Probabilistic learning combinations of local job-shop scheduling rules. Prentice Hall, 1963.
22. 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.
23. A. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.
24. 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.
25. F. Glover and G. Kochenberger. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer, January 2003.
26. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.
27. D. E. Goldberg. Probability matching, the magnitude of reinforcement, and classifier system bidding. *Machine Learning*, 5(4):407–425, 1990.
28. J. Gottlieb and N. Voss. Adaptive fitness functions for the satisfiability problem. In *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*. Springer Verlag, 2000. LNCS 1917.
29. Y. Hamadi, E. Monfroy, and F. Saubion. Special issue on autonomous search. *Contraint Programming Letters*, 4, 2008.
30. Y. Hamadi, E. Monfroy, and F. Saubion. What is autonomous search? Technical Report MSR-TR-2008-80, Microsoft Research, 2008.
31. J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI, 1975.
32. F. Hutter, Y. Hamadi, H. Hoos, and K. L. Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Twelfth International Conference on Principles and Practice of Constraint Programming*, volume 4204 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2006.
33. F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007.
34. B. A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 81–87. Morgan Kaufmann, 1995.
35. F. Lardeux, F. Saubion, and J-K. Hao. Recombination operators for satisfiability problems. In *Artificial Evolution, 6th International Conference, Evolution Artificielle*, volume 2936 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2004.
36. 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.

37. F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.
38. F. G. Lobo and D. E. Goldberg. Decision making in a hybrid genetic algorithm. In *IEEE International Conference on Evolutionary Computation*, pages 121–125. IEEE Press, 1997.
39. E. Marchiori and C. Rossi. A flipping genetic algorithm for hard 3-SAT problems. In *Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 393–400, 1999.
40. J. Maturana, A. 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*, 2009.
41. J. Maturana and F. Saubion. On the design of adaptive control strategies for evolutionary algorithms. In *Proc. Int. Conf. on Artificial Evolution. LNCS 4926, Springer*, 2007.
42. 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.
43. J. Maturana and F. Saubion. A compass to guide genetic algorithms. In G. Rudolph et al., editor, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2008.
44. S. Meyer-Nieberg and H.G. Beyer. *Self-Adaptation in Evolutionnary Computation*, pages 47–76. Springer Verlag, 2007.
45. V. Nannen, S. K. Smit, and A. E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 528–538. Springer, 2008.
46. V. Pareto. Cours d'économie politique. in Vilfredo Pareto, Oeuvres complètes, Genève : Librairie Droz, 1896.
47. J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.
48. 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.
49. L. Sais. *Problème SAT : progrès et défis*. Collection Programmation par contraintes. Hermès, 2008.
50. L. Simon and D. Le Berre. The SAT2005 competition. Technical report, Eighth International Conference on the Theory and Applications of Satisfiability Testing, June 2005.
51. S. Smit and G.Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedins of the IEEE Congress on Evolutionary Computation*, 2009.
52. A. K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25, 2008.
53. G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
54. D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In H.-G. Beyer, editor, *Proc. GECCO'05*, pages 1539–1546. ACM Press, 2005.
55. D. Thierens. Adaptive Strategies for Operator Allocation. In F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 77–90. Springer Verlag, 2007.
56. A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2):161–184, 1998.
57. J. M. Whitacre, T.Q. Pham, and R. A. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1345–1352, New York, NY, USA, 2006. ACM.
58. Wong, Lee, Leung, and Ho. A novel approach in parameter adaptation and diversity maintenance for GAs. *Soft Computing*, 7(8):506–515, 2003.
59. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.