Adaptive Operator Selection and Management in Evolutionary Algorithms

Jorge Maturana, Alvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, Michèle Sebag

▶ **To cite this version:**

Jorge Maturana, Alvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, et al.. Adaptive Operator Selection and Management in Evolutionary Algorithms. Adaptive Operator Selection and Management in Evolutionary Algorithms, Springer-Verlag, pp.161-189, 2012, 978-3-642-21434-9. 10.1007/978-3-642-21434-9_7 . hal-03256764

HAL Id: hal-03256764

https://univ-angers.hal.science/hal-03256764v1

Submitted on 23 Aug 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Adaptive Operator Selection and Management in Evolutionary Algorithms

Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer,
Frédéric Lardeux, and Michèle Sebag

7.1 Introduction

Evolutionary Algorithms (EAs) constitute efficient solving methods for general optimization problems. From an operational point of view, they can be considered as basic computational processes that select and apply variation operators over a set of possible configurations of the problem to be solved, guided by the Darwinian "survival of the fittest" paradigm.

In order to efficiently apply an EA to an optimization problem, there are many choices that need to be made. Firstly, the design of the general skeleton of the algorithm must include the selection of a suitable encoding for the search space at hand, the management of the population (i.e., size setting, selection and replacement processes, and so on), and the definitions of the variation operators that will be used, namely the mutation and recombination operators. These components can be

Jorge Maturana
Instituto de Informática, Universidad Austral de Chile, Valdivia, Chile.
e-mail: jorge.maturana@inf.uach.cl

Álvaro Fialho
Microsoft Research – INRIA Joint Centre, Orsay, France. e-mail: alvaro.fialho@inria.fr

Frédéric Saubion
LERIA, Université d'Angers, Angers, France. e-mail: saubion@info.univ-angers.fr

Marc Schoenauer
Project-Team TAO, INRIA Saclay – Île-de-France and LRI (UMR CNRS 8623), Orsay, France and Microsoft Research – INRIA Joint Centre, Orsay, France.
e-mail: marc.schoenauer@inria.fr

Frédéric Lardeux
LERIA, Université d'Angers, Angers, France. e-mail: lardeux@info.univ-angers.fr

Michèle Sebag
Project-Team TAO, LRI (UMR CNRS 8623) and INRIA Saclay – Île-de-France, Orsay, France and Microsoft Research – INRIA Joint Centre, Orsay, France. e-mail: michele.sebag@inria.fr

considered as the *structural parameters* of the algorithms that define its operational architecture. However, once the structure is defined, a difficult and crucial task remains: how to control the general computation process? This control is usually embedded in a set of *behavioral parameters* that can be related to the data structures or to the computation steps, e.g., the application rate of each variation operator.

All these parameters should then be tuned, depending on the problem at hand. In the early days of Evolutionary Computation, these numerous possible choices were in fact considered as richness, providing very useful flexibility to EAs, that could indeed be applied to a very wide range of scientific fields. Nowadays, the contemporary view of EAs acknowledges that specific problems require specific setups in order to obtain satisfactory performance [13]. In other words, when it comes to solving a given problem, all practitioners know that parameter setting is in fact the Achilles' heel of EAs (together with their high computational cost).

As the efficiency of Evolutionary Algorithms has already been experimentally demonstrated on many difficult problems out of the reach of other optimization methods, more and more scientists (researchers, engineers) are trying them out on their specific problems. However, they very often fail in getting interesting results precisely because there is a lack of general methods for tuning at least some of the involved parameters (and also because they are not, and do not want to become, "Evolutionary Engineers"). Of course, the specialization of an EA to a given problem has an impact on its performance on other problems (according to the No Free Lunch Theorems for Optimization [44]). In fact, through parameter setting, the main challenge is to set (or select) the right algorithm for the given problem, which is an old problem in computer science [39]. Therefore, it is time that we *Cross the Chasm* [33], bringing the benefits of EAs to the whole scientific community without its main burden, that of parameter setting.

A current trend in EA is to focus on the definition of more autonomous solving processes, which aim at allowing the basic user to benefit from a more efficient and easy-to-use algorithmic framework. Parameter setting in EAs is a major issue that has received much attention in recent years [14], and its research is still very active nowadays, as a book recently published [27], and from the numerous recent references cited in this document. This subject is not limited to EAs, being also investigated in operations research and constraint programming communities, where the current solving technologies that are included in efficient solvers require huge expertise to be fully used (see, for instance, [4]).

Parameter setting in EAs may be considered at two complementary levels as follows [36]:

Design: 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 with problems outside the basic frameworks. Even if standard variation operators exist in the literature (such as the uniform crossover [40]), the achievement of acceptable results depends 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 [11]. In any case, the impact on the computation process of problem-specific operators is even more difficult to forecast than of well-known operators, and thus their associated parameters are harder to correctly estimate a priori.

Behavior: Once the general architecture of the algorithm has been defined, the user needs to configure the behavior of these components through parameters. This has a direct impact on the performance and reliability of the algorithms. Indeed, its efficiency is strongly related to the way the Exploration versus Exploitation (EvE) dilemma is addressed, determining the ability of the EA to escape from local optima in order to sparsely visit interesting areas, while also focus on the most promising ones, to thus reaching global solutions. However, it is known that more exploration of the search space is necessary in the initial generations, while more exploitation should be done when the search is approaching to the optimum; thus, a static definition of the application rates for each operator will be always suboptimal.

We propose transparently including these two possible levels of control into the basic algorithmic process, as illustrated in Figure 7.1. Firstly, at the *design* level, the *Adaptive Operator Management* (AOM) aims at handling the suitable subset of operators that is made available to the algorithm, excluding the useless operators and including possibly useful ones. These operators are extracted from a set of potential operators (either automatically generated or predefined). Based on such subset of available operators, defined and maintained by the AOM, the *Adaptive Operator Selection* (AOS) is used to control the *behavior* of the algorithm, handling the problem of selecting the operators to be applied at every instant of the search. It clearly appears that the concepts of AOM and AOS are fully complementary, and very useful for making an EA more autonomous. Within the combination of both approaches, the design and the behavior of the algorithm are automatically adapted while solving the problem, from continuous observation of the performance of the operators during the search, based on its needs with regard to exploration and exploitation.

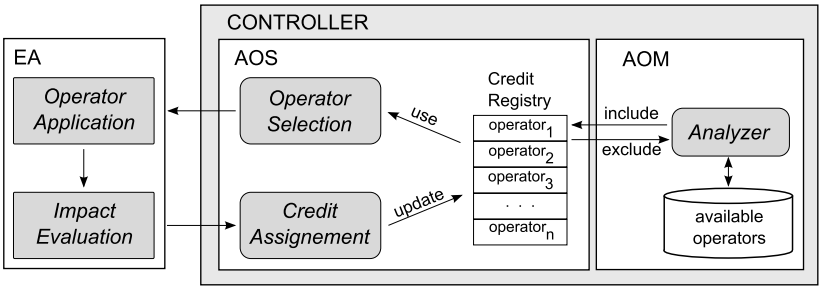

Fig. 7.1: General scheme of the controller, depicting its two main components, *Adaptive Operator Selection* and *Adaptive Operator Management*

This chapter will focus on these control mechanisms, by reviewing existing solutions previously proposed by the authors, and providing examples of their possible applications by means of cases studies. Firstly, an overview of the current state of the art in parameter setting in EAs and, more specifically, in adaptive control of operators, is presented in Section 7.2. Then, some efficient AOS combinations are presented in Section 7.3. A case study of the application of an AOM technique, combined with the AOS, is presented in Section 7.4. In both cases, experimental results are presented, applying the adaptive EA to the solution of SAT problems. Finally, conclusions and possible paths for further research are drawn in Section 7.5.

7.2 Parameter Setting in Evolutionary Algorithms

Slightly departing from the point of view adopted in previous surveys [14, 13], we shall firstly distinguish here between *external tuning* and *internal control* of the parameters.

Methods that perform *external tuning* consider the algorithm as a black box: some external process is run (e.g., another optimization algorithm, or some standard statistical analysis), to determine a good (range of) value(s) for the parameters at hand. Most of these methods are not limited to Evolutionary Algorithms and can be applied to any type of algorithm, although some of them have been designed in the evolutionary framework. The most straightforward approach is the complete factorial Design of Experiments, in which the same number of runs is executed for each setting, and the one that achieves the best performance on average is selected for the algorithm. Other approaches have been recently proposed, such as the SPO [3], in which the optimal values are refined after a few runs; the Racing methods [6, 46], in which settings are discarded once proven worst than others ***; and the recent Meta-EA approaches [8], in which some runs are actually stopped at some point of the evolution, and some of their parameters are modified before they are restarted.

On the other hand, *internal 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 are at least two strong arguments to support the idea of changing the parameters during an *Evolutionary Algorithms* 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 the landscape is) 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 op-

timal at different stages of the search process (see [13, p. 21] and references therein).

The different approaches that have been proposed to internally adapt the parameters can be classified into three categories, depending on the type of information that is used to adjust the parameters during the search (such classification, used by many authors in the 1990s, has been made in [14] and republished in the corresponding chapter of [13]).

Deterministic parameter control implements a set of deterministic rules, without any feedback from the search. This is, in general, hard to achieve, because of a simple reason: it relies heavily on knowing beforehand how long it will take to achieve a given target solution with the running algorithm and, obviously, this cannot be easily predictable. But even if it were, the way to balance exploration and exploitation can hardly be guessed. This situation is worsened by two facts: first, there is usually a big variance between different runs *on the very same problem*; and second, these methods often require additional parameters that are used to tune the deterministic parameter (starting with the total number of generations the algorithm will run), and even though these parameters can be considered second-order, their influence is nevertheless critical.

Since our knowledge about the way the search should behave is always limited, it is sometimes possible, and advantageous, to let evolution itself tune some of the parameters: such a *Self-Adaptive* parameter control adjusts parameters "for free", i.e., without any direct specification from the user. In other words, individuals in the population might contain "regulatory genes" that control some of the parameters, e.g., the mutation and recombination rates; and these regulatory genes are subject to the same evolutionary processes as the rest of the genome [23]. For quite some time in the 1990s, self-adaptation was considered as the royal road to success in Evolutionary Computation. First of all, the idea that the parameters are adapted *for free* is very appealing, and its parallel with the behavior of self-regulated genes is another appealing argument. On the practical side, as self-adaptive methods require little knowledge about the problem and, "what is probably more important", about the way the search should proceed, they sometimes are the only way to go when nothing is actually known about the problem at hand. However, by using such an approach, the algorithm needs to explore, in parallel, the search space of the variables of the problem and also the search space of the parameter values, which potentially increases the complexity of the search.

Then, it is clear that, whenever some decisions can be made to help the search follow an efficient path, this should be done. *Adaptive* or *Feedback-based* methods follow this rationale, being the most successful approaches today in on-line parameter control. These methods are based on the monitoring of particular properties of the evolutionary process, and use changes in these properties as an input signal to change the parameter values. The controllers presented in this work are included in this category.

Adaptive Operator Selection (AOS), presented in Section 7.3, is used to autonomously select which operator, among the available ones, should be applied by the EA at a given instant, based on how they have performed in the search so far. The

set of operators available to the EA is usually static, defined *a priori* by an expert or an off-line method. However, since the performance of operators is continuously changing during the search, useless operators might be deleted, while possibly useful ones might be inserted in such a set. This is the underlying idea proposed by the *Adaptive Operator Management* (AOM), presented in Section 7.4.

To achieve the goals associated with AOS, two components are defined (as shown in Fig. 7.1): the *Credit Assignment* - how to assess the performance of each operator based on the impact of its application on the progress of the search; and the *Operator Selection* rule - how to select between the different operators based on the rewards that they have received so far. The rest of this section will survey existing AOS methods, looking at how they address each of these issues.

7.2.1 Learning by Credit Assignment

The AOS learns the performance of each operator based on a performance measure, assessed by what is usually referred to as the *Credit Assignment* mechanism. Starting with the initial work in AOS, from the late 1980's [12], several methods have been proposed to do this, with different ways of transforming the impact of the operator application into a quality measure.

Most of the methods use only the fitness improvement as a reward, i.e., the quality gain of the newborn offspring compared to a base individual, which might be (i) its parent [26, 42, 2], (ii) the current best individual [12], (iii) or the median individual [24] of the current population. If no improvement is achieved, usually a null credit is assigned to the operator.

Regarding which operator should be credited after the achievement of a given fitness improvement, the most common approach is to assign a credit to the operator that was directly responsible for the creation of the newborn offspring. Some authors, however, propose assigning credit to the operators used to generate the ancestors of the current individual (e.g., using some bucket brigade-like algorithm [12, 24]), based on the claim that the existence of efficient parents is indeed as important as the creation of improved offspring. Others, however, do not consider ancestors at all ([26, 42]), and some even suggest that this sometimes degrades the results [2].

The existing approaches also differ in the statistics that are considered in order to define the numerical reward. Most of the methods calculate their rewards based just on the most recent operator application, while others use the average of the quality achieved over a few applications. More recently, in [43], the utilization of extreme values over a few applications was proposed (statistical outliers), based on the idea that highly beneficial but rare events might be better for the search than regular but smaller improvements. The reported comparative results with other *Credit Assignment* mechanisms show the superiority of this approach over a set of continuous benchmark problems. Though used in a different context, the methods presented in this paper are also based on this idea.

7.2.2 Adaptation by Operator Selection Rule

The most common and straightforward way of doing *Operator Selection* is the so-called *Probability Matching* (PM) [20, 26, 2]. In brief, each operator is selected by a roulette-wheel-like process with a probability that is proportional to its known empirical quality (as defined by the *Credit Assignment* mechanism, e.g., the average of the received rewards). A user-defined α parameter might be used to introduce some relaxation in the update of this empirical estimation. Besides, a minimal probability (p_{min}) is usually applied, so that no operator is "lost" during the process: one operator that is currently bad might become useful at some further stage of the search. If an operator receives just null rewards (or the maximal reward) for some time, its expected reward will go to p_{min} (or $p_{max} = 1 - K * p_{min}$). However, this convergence is very slow, and, experimentally, mildly relevant operators keep being selected, which badly affects the performance of the algorithm [41].

Originally proposed for learning automata, *Adaptive Pursuit* (AP) [41] is an *Operator Selection* technique that partly addresses this drawback by implementing a winner-take-all strategy. Another user-defined parameter, β, is used to control the greediness of the strategy, i.e., how fast the probability of selecting the current best individual will converge to p_{max} while that of selecting the others will go to p_{min}. In its original proposal for the AOS problem [41], AP was applied in a set of artificial scenarios, in which the operator qualities were changing every Δt applications, achieving significant improvements over the performance of PM.

Different approaches, such as APGAIN [45], propose an *Operator Selection* divided into two periods. During a first learning stage (which is repeated several times during the run, so the changes can be followed), the operators are randomly selected, and the rewards gathered are used to extract initial knowledge about them. In a later stage, such knowledge is exploited by the technique in order to efficiently select the operators. The main drawback in this case is that roughly a quarter of the generations are dedicated to the learning, thus doing random selection, which may strongly affect the performance of the algorithm if disruptive operators are present in the set.

7.2.3 Meta-Parameters vs. Original Parameters

Clearly, the main objective of parameter setting in EAs is to automate some of the choices that should be made by the user. What happens in reality is that, although some of the parameters are efficiently defined by these autonomous controllers, the controllers themselves also have their own parameters that need to be tuned. This section considers this issue.

If one wants to define "manually" (or by the use of some off-line tuning technique) the parameters of an EA related to the variation operators, several choices need to be made. In addition to defining the list of operators that will be used by the algorithm, we need at least one other parameter to define *for each operator*: its application rate. Additionally, some operators also require some extra settings,

e.g., the rate of flipping a bit in a bit-flip mutation operator, or the number of crossing points that will be used by an n-point crossover operator. Thus, considering a static definition of the operators and their respective rates, the number of original parameters to be defined is a multiple of the number of operators that is used by the EA.

All these parameters are automatically handled by the proposed adaptive controllers while solving the problem. The list of operators is managed by the *Adaptive Operator Management*, while the application rates are abstracted by the *Adaptive Operator Selection*. The extra parameters are abstracted as well, by considering different variants of the same operator as different operators, e.g., 1-point and 2-point crossover are treated as different operators instead of different (discrete) settings of the same operator.

As previously mentioned, to be able to do this, the controllers also introduce their own parameters, usually called hyper- or meta-parameters. But, firstly, there is a fixed number of meta-parameters instead of a multiple of the number of operators; secondly, these parameters are (supposed to be) less sensitive than the original ones; and finally, they are adapted online according to the needs of the search, while a static setting would always be suboptimal, as discussed in Section 7.2. For example, concerning the Operator Selection techniques reviewed in Section 7.2.2, Probability Matching needs the definition of the minimal probability p_{min} and the adaptation rate α; while the Adaptive Pursuit needs the definition of a third parameter, the learning rate β. For Credit Assignment, usually at least one parameter should be considered: the number of operator applications that are taken into account to calculate an empirical estimation that the controller keeps about each operator.

Even though the number of meta-parameters is fixed and small, given the initial lack of knowledge about the behavior of the controllers and what would be a good setting for them, off-line parameter tuning methods can be used to facilitate such definitions.

The cost of acquiring information about parameters is variable: whereas some parameters can be easily recognized as preponderant and easy to tune, others could have little effect over the search (i.e., different parameter values do not significantly affect the results), or could require many experiments to discover a correct value. For example, information theory could be used to characterize the relevance of the parameters, as in [34, 35]. The proposed method, called REVAC, uses Shannon and differential entropies to find the parameters with higher impact on the efficiency of the algorithm while estimating the utility of the possible parameters values.

Another possibility is the utilization of Racing methods: instead of performing m runs on each of the n instances in order to find the best configuration (i.e., a full factorial Design of Experiments), the candidate configurations are eliminated as soon as there is enough statistical evidence that they are worse than the current best one. Cycles of "execution/comparison/elimination" are repeated until there is only one configuration left, or some other stopping criterion is achieved. So, instead of wasting computing time to estimate with precision the performance of inferior candidates, Racing allocates the computational resources in a better way, by focusing on the most promising ones and consequently achieving lower variance estimates

for them. Racing typically requires 10-20% of the computing time of a complete factorial Design of Experiments.

7.3 Adaptive Operator Selection

In this section, we review three AOS methods from our previous work. *Compass* [31] (Section 7.3.1) presents a *Credit Assignment* mechanism able to efficiently measure the impact of the operators application, while *Ex-DMAB* [15] (Section 7.3.2) proposes an *Operator Selection* that quickly adapts to the dynamics of the search. These two elements were combined into *ExCoDyMAB* [28], which is described and empirically compared to the original methods, respectively, in Sections 7.3.3 and 7.3.4.

7.3.1 Compass: Focusing on Credit Assignment

The *Adaptive Operator Selection* technique proposed in [31] uses a very simplistic *Operator Selection* mechanism, *Probability Matching*. However, its *Credit Assignment* mechanism is very efficient in measuring the impact of the operator's application, thus being the main contribution of this work. As is known, in case of multimodal functions, fitness improvement is not the only important criterion for the progress of the search; some level of diversity should also be maintained in the population; otherwise the search will quite probably converge prematurely and get trapped into a local optimum. Based on these assumptions, *Compass* provides a way to encourage improvements for both criteria, and works as follows.

Firstly, each time an operator is applied, three measures are collected: the variations in the population diversity and in the mean fitness of the population, respectively ΔD and ΔQ, and also its execution time T, as shown in Fig. 7.2(a). The average performance of each operator w.r.t. ΔD and ΔQ over the last τ applications is displayed in a "diversity vs. fitness" plot, represented by the points in Fig. 7.2(b). A user-defined angle θ defines the compromise between obtaining good results and maintaining diversity in the population, addressing the EvE dilemma. In practice, such an angle defines the plane from which perpendicular distances to the points are measured. Finally, these measures (δ_i) are divided by the operator's execution time to obtain the final aggregated evaluation of each one (Fig. 7.2(c)), which is the reward used to update the selection preferences.

This approach was used to control a steady state evolutionary algorithm applied to the well-known Boolean satisfiability problem (SAT) [9], automatically selecting from six ill-known operators. The SAT problem was chosen because it offers a large variety of instances with different properties and search landscapes, besides allowing the scaling of the instance difficulty. The experiments have demonstrated that this *AOS* method is efficient and provides good results when compared to other

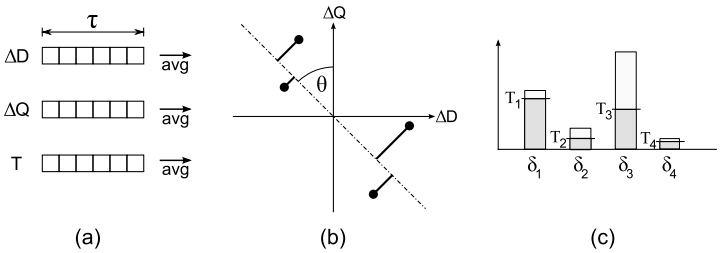

Fig. 7.2: *Compass* credit assignment: Sliding windows of three measures are maintained (a). Average measures of ΔD and ΔQ are plotted and distance of those points are measured from a plane with a slope of θ (b). Finally, those distances are divided by the execution time, resulting in the reward assigned to the operator (c)

existing mechanisms, such as Adaptive Pursuit [41] and APGAIN [45], based on the fitness improvement.

Such performance was achieved mainly due to the strength of the *Credit Assignment* mechanism proposed, which provides a robust measurement of the impact of the operator application by simultaneously considering several criteria. The *Operator Selection* rule used (PM) is rather simple, known being quite conservative and slow w.r.t. the adaptation, as already discussed in Section 7.2.2.

7.3.2 Ex-DMAB: Focusing on Operator Selection

Adaptive Operator Selection proposed in [15], uses extreme fitness improvement as *Credit Assignment*, based on the assumption that attention should be paid to extreme, rather than average, events [43]. This is implemented simply by assigning as a reward the maximum of the fitness improvements achieved by the application of a given operator over the last κ applications.

Concerning the *Operator Selection* mechanism, the idea is that such a task might be seen as another level of the Exploration vs. Exploitation (EvE) dilemma: there is the need to apply as much as possible the operator known to have brought the best results so far (exploitation), while nevertheless exploring the other options, as one of them might become the best at a further stage of the search. The EvE dilemma has been intensively studied in the context of *Game Theory*, and more specifically within the so-called Multi-Armed Bandit (MAB) framework. Among the existent MAB variants, the *Upper Confidence Bound* (UCB) [1] was chosen to be used as it provides asymptotic optimality guarantees, although it is described as "Optimism in front of the Unknown".

More formally, the UCB algorithm works as follows. Each variation operator is viewed as an *arm* of an MAB problem, and is associated with (i) its empirical reward $\hat{p}_j$, i.e., the average performance obtained by it so far; and (ii) a confidence interval,

based on the number of times n_j such an operator has been tried. At each time step t, the algorithm selects the arm with the best upper bound w.r.t. the following quantity:

$$\hat{p}_{j,t} + C\sqrt{\frac{\log \sum_k n_{k,t}}{n_{j,t}}} \tag{7.1}$$

The first term of this equation favors the best empirical arm (exploitation) while the second term ensures that each arm is selected infinitely often (exploration). In the original setting [1], all rewards are Boolean, and hence their empirical means $\hat{p}_{i,t}$ are in $[0,1]$. As this is not the case in the context of AOS, a *Scaling factor* C is needed in order to properly correct the balance between the two terms.

The most important issue, however, is that the original MAB setting is static, while the AOS scenario is dynamic, i.e., the quality of the operators is likely to vary during the different stages of the search. Even though the exploration term in the UCB algorithm ensures that all operators will be tried infinitely many times, in case a change occurs (w.r.t. the operator's performance), it might take a long time before the new best operator catches up. To be able to efficiently follow the changes in dynamic environments, it has been proposed [21] to combine a statistical test with the UCB algorithm that efficiently detects changes in time series, the *Page-Hinkley* (PH) test [37]. Basically, as soon as a change in the reward distribution is detected (based on a threshold γ), e.g., the "best" operator is probably not the best anymore, the UCB algorithm is restarted from scratch, thus being able to quickly rediscover the new best operator.

The UCB algorithm involves one meta-parameter, the scaling factor C, while the PH test involves two parameters, the threshold γ for the change detection, and δ, which enforces the robustness of the test when dealing with slowly varying environments. Note that, according to initial experiments in [10], δ has been kept fixed at 0.15. The UCB algorithm, coupled with the Scaling factor and the PH change-detection test, has been termed as *Dynamic MAB* (*DMAB*), with the complete *AOS* combination being denoted by *Ex-DMAB*.

Ex-DMAB has been used to adapt a (1+λ)-EA by efficiently choosing on-line from four mutation operators to solve the OneMax problem [10], and has been tried on yet another unimodal benchmark problem, the Long k-Path [16], this time effectively selecting from five mutation operators. In both cases, the optimal operator selection strategy was extracted by means of Monte Carlo simulations, and *Ex-DMAB* was shown to perform statistically equivalently to it while significantly improving over the naive (uniform selection) approach. It has also been used to adapt a (100,100)-EA with four crossover operators and one mutation operators in the Royal Road problem [17], also performing significantly better than the naive approach. For the three problems, we have also used other *AOS* combinations as baselines for comparison, namely Adaptive Pursuit, Probability Matching and static MAB (UCB without restarts), coupled with Extreme or Average rewards. *Ex-DMAB* was shown to be the best option.

However, as its *Credit Assignment* considers only the fitness improvement, *Ex-DMAB* would probably be less efficient on rougher landscapes, quickly converging to local optima. In order to solve multimodal problems, diversity should also be considered somehow. Besides being problem-dependent, another drawback of *Ex-DMAB* is that the variance of the fitness improvements changes as the search advances (in the beginning of the search larger improvements are more easily achieved, while in the fine-tuning final stage only very small improvements are possible). Thus, there do not exist any robust values for the meta-parameters C and γ that make it likely to perform well during the whole search. A possible solution to this is the utilization of comparison-based rewards (that, by definition, always keep the same variance, no matter what the problem and the search stage), e.g., a credit based on ranks or the use of some normalized measures.

7.3.3 ExCoDyMAB = Compass + Ex-DMAB : An Efficient AOS Combination

The previous sections showed that the strengths and weaknesses of both *Compass* and *Ex-DMAB* methods are complementary: *Compass* measures in a holistic way the effects of the operator applications over the population, but the operator selection rule is rather rudimentary, while *DMAB* has an effective way to adapt and select the operators, but its credit assignment mechanism is probably too simplistic. It seems hence natural to combine the *Credit Assignment* of the former with the *Operator Selection* mechanism of the latter.

However, even though merging both modules seems to be straightforward, some important issues need to be further explored:

- *Compass* uses sliding windows of size τ in the "measurement stage", with a unique reward value in its output, while *Ex-DMAB* stores in a sliding window the last κ outputs (rewards) of its *Credit Assignment* module. Should we keep both windows, or would it be redundant? And if only one is kept, which one should it be? From here on, these two windows will be referred to as $W1$ for the measurement window and $W2$ for the reward window.
- Another issue concerning the sliding windows is that of their usage: should the algorithm use their average (A), extreme (E), or simply instantaneous (I) value (equivalent to using no window at all)? The *Extreme* was shown to be stronger in unimodal problems, but how do such results hold in this completely different scenario?
- The last issue concerns the other meta-parameters. Besides tuning the size and type of $W1$ and $W2$, we also need to tune the values of the angle θ in *Compass*, and the scaling factor C and change detection threshold γ in *DMAB*. Since the idea is not to simply replace some parameters (the operator application probabilities) by other ones at a higher level of abstraction, we need to better understand their effects. One way to find answers to this

Table 7.1: Racing survivors

Name	$W1$ type, size	$W2$ type, size	C	γ
A	Extreme, 10	Instantaneous	7	1
B	Extreme, 10	Average, 10	7	1
C	Extreme, 10	Average, 50	7	1
D	Extreme, 10	Extreme, 10	7	3

questions is to experimentally study their influence on the performance of the *AOS* in the situation, and propose some robust default values whenever possible.

To analyze such issues, an empirical study was done, considering the following values for each of the meta-parameters: $C \in \{5,7,10\}$; $\gamma \in \{1,3,5\}$; and the window type(size) combinations $\in \{A(10),A(50),E(10),E(50),I(1)\}$ for both $W1$ and $W2$, summing up to a total of 225 possible configurations. The angle θ for *Compass* was fixed at 0.25, based on preliminary experiments (see [28] for further details). *ExCoDyMAB* was experimented within the same Evolutionary Algorithm used in [31], with the objective of selecting from six ill-known variation operators while solving different instances of the well-known combinatorial SAT problem [9].

Given the high number of possible configurations and the initial lack of knowledge about which values should be good for each of the mentioned meta-parameters, an off-line tuning technique was used for their setting, the F-Race [6], a Racing method that uses the Friedman's two-way Analysis of Variance by Ranks as a statistical test to eliminate the candidates. The stopping criteria for the Racing was set to 80 runs over all the instances (a "training set" of 7 instances was used), with eliminations taking place after each run, starting from the 11^{th}.

At the end of the Racing process, four configurations were still "alive", presented in Table 7.1. This clearly indicates that the most important sliding window is $W1$, and it should be used in its Extreme configuration with a size of 10 (i.e., taking as *Compass* inputs the maximal of the last ten values), no matter which kind/size of $W2$ is being used. This fact emphasizes the need to identify rare but good improvements, greatly supporting the idea raised in [15]. Besides, the size of 10 for $W1$ could be interpreted as follows: with the Extreme policy, a larger τ would produce high durability of the extreme values, even when the behavior of the operator has changed. On the other hand, a smaller value $\tau = 1$ (i.e., the same as choosing Instantaneous policy) would forget those "rare but good" cases. One could suppose that an optimal size for $W1$ depends on the fitness landscape and the operators used. Further research is needed to better understand the setting of τ. The configuration "C" was found to be the best among them, and was thus used in the empirical comparison with other techniques, presented in the following.

Table 7.2: Comparative results on the 22 SAT instances: average (std dev) number of false clauses (over 50 runs)

Method / Problem		*Compass*	*Ex-DMAB*	*Uniform Choice*
4blocks	**2.8 (0.9)**	6 (0.9)	6.2 (0.9)	13.4 (0.6)
aim	**1 (0)**	**1 (0)**	1.2 (0.3)	3.6 (1.8)
f1000	**10.3 (2.3)**	30.9 (6.2)	16.4 (2.6)	55.8 (8.6)
CBS	**0.6 (0.6)**	**0.4 (0.5)**	1 (0.9)	7 (2.7)
Flat200	**7.2 (1.7)**	10.6 (2.1)	10.7 (2.2)	37.7 (5.5)
logistics	**6.5 (1.3)**	7.6 (0.5)	8.8 (1.5)	17.9 (4.1)
medium	1.5 (1.5)	**0 (0)**	1.8 (1.6)	8.8 (3.4)
Par16	**15.2 (3.1)**	64 (10.2)	24.1 (5.7)	131.1 (14.5)
sw100-p0	**9.2 (1.2)**	12.8 (1.4)	12.5 (1.7)	25.9 (3.4)
sw100-p1	**0 (0)**	0.5 (0.6)	1.1 (0.8)	11.3 (3.5)
Uf250	**0.9 (0.7)**	1.8 (0.9)	1.7 (0.8)	9.1 (3.3)
Uuf250	**2.5 (1)**	4.5 (1.2)	3.1 (1.1)	12.7 (3.2)
Color	**48 (2.5)**	61.3 (2.2)	49.3 (3.4)	80.4 (6.6)
G125	**8.8 (1.3)**	20.6 (2)	13.5 (1.7)	28.8 (4.6)
Goldb-heqc	**72.9 (8.5)**	112.2 (15.2)	133.2 (15.9)	609.7 (96.2)
Grieu-vmpc	16.7 (1.7)	**15.2 (1.7)**	19.6 (1.8)	24.1 (3.3)
Hoons-vbmc	**69.7 (14.5)**	268.1 (44.6)	248.3 (24.1)	784.5 (91.9)
Manol-pipe	**163 (18.9)**	389.6 (37.2)	321 (38.1)	1482.4 (181.5)
Schup	**306.6 (26.9)**	807.9 (81.8)	623.7 (48.5)	1639.5 (169.9)
Simon	**29.6 (3.3)**	43.5 (2.7)	35.3 (6.3)	72.6 (11.3)
Velev-eng	**18.3 (5.2)**	29.5 (7.3)	118 (37.1)	394 (75.8)
Velev-sss	**2 (0.6)**	4.6 (1)	5.9 (3.9)	62.7 (25.2)
Comparison	-	18 - 2	21 - 0	22 - 0

7.3.4 Experimental Results

The performance of *ExCoDyMAB* (C) was compared with the baseline techniques, the original *Compass* and *Ex-DMAB*, and also with the *Uniform* (naive) selection, with 50 runs being performed on 22 SAT instances, obtained from [22] and from the SAT Race 2006. A preliminary off-line tuning of the meta-parameters by means of F-Race [6] was also done for the other techniques, in order to compare them at their best (more details in [28]).

The results are presented in Table 7.2. The columns show the mean number of false clauses after 5,000 function evaluations, averaged over 50 runs, and the standard deviation in parentheses, with the best results for each problem presented in boldface. The last line of the table summarizes the comparison, by showing the number of "wins - losses" of *ExCoDyMAB* compared to each of the baseline techniques, according to a Student t-test with 95% confidence.

Note that the purpose of this work was not to build an amazing SAT solver, but rather to experiment with a different *AOS* and validate *ExCoDyMAB* within an EA solving a general, difficult combinatorial problem. The results of Table 7.2 show that

a basic EA using rather naive operators can indeed solve some instances. The main interesting result is that this set of benchmarks was difficult enough to highlight the benefit of using the proposed combination of *Compass* and *Ex-DMAB* rather than either separately – or a naive, blind choice. The deliberate choice of several non-specialized operators was also important for validating the control ability of *ExCoDyMAB* when facing variation operators of very different characteristics and performance. Competing for SAT races implies using highly specialized operators, such as the ones implemented in GASAT [25], and is left for further consideration.

This study highlights the importance of both control stages of *AOS*, namely credit assignment and operator selection rules. Both features, *Compass Credit Assignment*combined with *DMAB Operator Selection*, contribute to the overall performance of the proposed autonomous control method, explaining the efficiency gain over each previous method used alone. Additionally, using the Extreme values from the aggregated performance measure allows the algorithm to identify occasional but highly beneficial operators, while the inclusion of population diversity with the traditional fitness improvement measure contributes to escaping from local optima.

One main drawback of *ExCoDyMAB* is the tuning of its meta-parameters. Though the normalization of fitness improvements and diversity by *Compass* might result in a possible robust setting for the scaling parameter of the MAB balance (i.e., the value found here using Racing), further work is needed for a deeper understanding of how to tune the meta-parameter γ that triggers the change detection test. As previously mentioned, the off-line meta-parameter tuning using the F-Race paradigm can be done in a fraction (15%) of the time needed for a complete factorial DOE, but this is still quite costly.

Although its good performances rely expensive procedures, *ExCoDyMAB* was found to outperform the main options available to the naive EA user, namely (i) using a fixed or deterministic strategy (including the naive, uniform selection); and (ii) using a different *AOS* strategy, including the combinations previously proposed by the authors. Furthermore, *ExCoDyMAB* involves a fixed and limited number of parameters, whereas the number of operator rates increases with the number of operators, as discussed in Section 7.2.3.

7.4 Adaptive Operator Management

The design of the algorithm has great influence on the search performance. It can be seen as a parameter setting at a higher level of abstraction: one must decide, from several choices, the best configuration for the algorithm in order to maximize its performance. However, in this case "structural" parameters are considered instead of "behavioral" ones. In the case of operator control, instead of deciding on different values for the application rate of a given operator, it decides whether a given operator should be considered or not by the algorithm. It can be seen as a parameter optimization problem over a search space composed of algorithms.

This is usually done manually: the user decides, based mainly on his experience, which operators should be included in the search algorithm; or he tries a few configurations, assessing their performance in order to set down the definitive algorithm. As well as for the setting of the behavioral parameters, this design could also be assisted by parameter setting techniques, such as Racing or REVAC (see Section 7.2).

While the utilization of off-line tuning techniques is straightforward, providing the user a static "good" configuration, one might wonder how to do this in a dynamic way, while solving the problem. The motivations to do so would be the same as those that led us to propose the *AOS* methods: the efficiency of the operators changes according to the region of the search space that is currently being explored; thus a useless operator might be excluded from the algorithmic framework, while possibly useful ones might be included. The autonomous handling of the operators included in the algorithm while solving the problem is what we refer to as *Adaptive Operator Management* (AOM). Besides being yet another approach of doing parameter control, such an approach can also be described in the light of the recent field of hyper-heuristics [7].

7.4.1 Basic Concepts for Operator Management

There exist several criteria to determine whether an operator must be included or not in the search. The most straightforward one is the operator's recent success: if an operator has shown a good performance during the few last generations, it seems reasonable to keep it and exclude another with a lower performance. Another criterion could be based on a higher-level strategy of search, which encourages different levels of exploration and exploitation, as needed by the search. The latter implies the definition of a set of rules capable of guiding the search to efficiently balance intensification and diversification.

Regardless the criteria chosen to handle the operators, in general we can distinguish between the three following possible states for each operator, shown in Figure 7.3:

- **Unborn**, when the operators have never been used during the execution of the algorithm, or no information about their performance is available from previous runs.
- **Alive** refers to the operators that are currently being used by the algorithm. In this state, a trace of their performance is maintained.
- **Dead**, when the operators have been excluded from the search. The difference between this state and the *unborn* one is that here the controller has had the opportunity to assess the behavior of the operator; thus it could be eventually re-included later, with a clue about the effect it could produce.

It is arguable whether the observed performance of a given dead operator can be extrapolated to a later state of the search. Indeed, a dismissed "bad" operator could become useful in a different stage of the search, i.e., when the characteristics of the fitness landscape being currently explored by the population have changed and/or

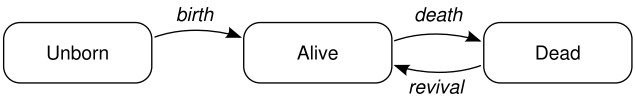

Fig. 7.3: Operator states

the placement of the population over the search space has changed. This could mean that the information stored in the profiles of the dead operators is no longer useful; thus there are only two states: *dead* and *alive*. Nevertheless, we prefer to consider the tree-state model, which offers wider possibilities in terms of previous knowledge inclusion.

Such states are linked by the following three transitions:

- **Birth**, when an operator is included for the first time into the search.
- **Death**, when an operator has been considered useless for the current state of the search, and thus removed from it.
- **Revival**, when a previously dismissed operator is brought back to the search.

The core of *Adaptive Operator Management* lies in defining the conditions under which the transitions must take place. These conditions correspond to the different criteria described above, which define whether and when an operator should be included and eliminated from the search.

7.4.2 Blacksmith: A Generic Framework for AOM

The framework presented here, *Blacksmith*, aims to control the design by modifying the set of operators available to the algorithm during its execution. The structure of the entire controller is shown in Figure 7.4. In addition to comprising the *Adaptive Operator Selection* component, *Adaptive Operator Management* is composed of a set of operator definitions from which the operators are extracted. This set could be simply a list of possible operators that can be included in the search, or an entire module that mixes subcomponents to create operators in an intelligent manner.

Blacksmith manages operator states by defining a couple of data structures that are attached to the operator definition. The rules that define the transition from one state to another can be summarized as follows:

1. Initially, when an operator is *Unborn* state, Blacksmith only knows its name.
2. Once operators are born (i.e., they pass from *unborn* to *alive* state) two data structures are associated with the operator name: *data*, which stores recent measures of performance, and *profile*, which summarizes the information in *data*, by calculating meaningful statistics. A fixed number of operators is kept in the registry, and they are evaluated at regular intervals.
3. Operators that have been applied a sufficient number of times are considered for elimination. This condition is required to ensure that the operator has low

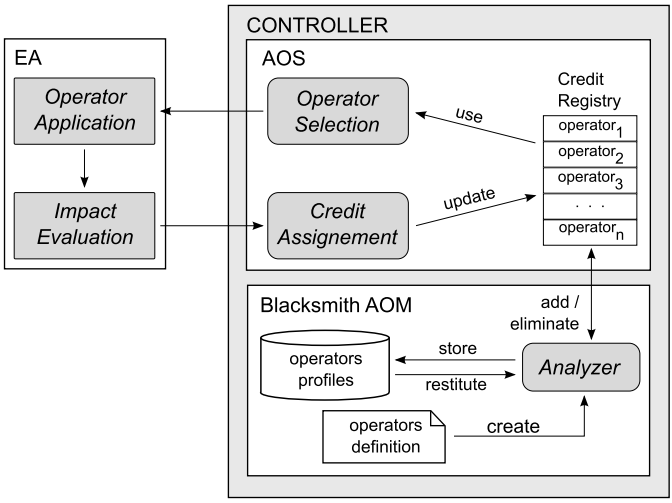

Fig. 7.4: Scheme of the controller used in this work, depicting its two main components, *Adaptive Operator Selection* and *Adaptive Operator Management*

performance. The weakest of those "known enough" operators is deleted (passing to the *dead* state) 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 *dead* ones are revived.

4. When there is no unborn operator left, replacement of unsuccessful operators is performed by including dead ones. Such reinsertion of the operator is not performed in a blind way, since the *profile* of the dead operators is maintained (*data*, in contrast, is dismissed).

Note that this implementation is generic enough to manage any operator: all that the AOM knows about the operators is their name and a summary of their performance.

In the implementation presented here, ten operators are kept concurrently in the algorithm, and they are evaluated every 50 generations. An operator's performance is considered to be well known after five applications, and is thus eligible for deletion. An operator is deleted if its performance lies in the lower third of the alive operators. The performance is assessed by a *Credit Assignment* mechanism (indeed, the same information is shared between the AOS and AOM), presented in the following section.

This approach was applied to control an evolutionary algorithm based on GASAT [25], which solves the SAT problem [19, 5]. The controller deals with crossover operators that are created by combining four different criteria, summing up to 307 possible crossover operators. The criteria are related to how the clauses are selected and what action is performed on them, as follows (further details can be found in [29]):

1. **Selection of clauses that are false in both parents**: select none, in chronological order, randomly, randomly from the smallest and randomly from the biggest.
2. **Action to perform on the selected clauses**: either take no action or consider one of four ways to flip variables with different criteria.
3. **Selection of clauses that are true in both parents**: same as in (1).
4. **Action to perform on the selected clauses**: same as in (2).

7.4.3 Credit Assignment

In order to improve the Credit Assignment, beyond Compass (described in Section 7.3.1), we compare two other schemes of evaluation, both based on the concept of Pareto dominance [38]. In 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 optimization: if we consider a maximization problem in dimension i, then a dominates b if $a_i > b_i$; on the other hand, if the objective is to minimize a given function, then a dominates b if $a_i < b_i$. When neither of the two points dominates the other, they are said to be incomparable. In our case, we have a two-dimensional space $(\Delta Diversity, \Delta Quality)$ with two criteria that we want to maximize.

The first scheme is called *Pareto Dominance (PD)*, and it counts the number of operators dominated by other operators (see Figure 7.5(b)). The purpose here is to obtain a high value. The second evaluation method, called *Pareto Rank (PR)*, computes the number of operators that dominate a given operator (Figure 7.5(c)). Here the objective is to obtain low values. Operators with a PR value of 0 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 also rewards operators which are in strong competition with the others.

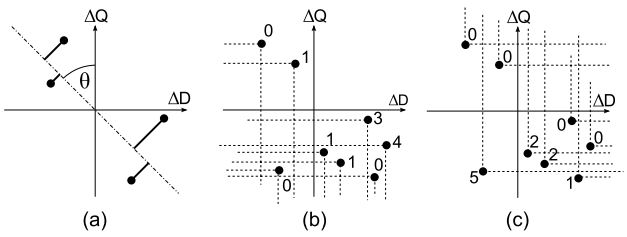

Fig. 7.5: Credit Assignment schemes. Compass (a), Pareto Dominance (b), Pareto Rank (c)

After the application of an operator, the values of $\Delta Diversity$ and $\Delta Quality$ are sent to the controller. The credit assignment module computes then the evaluation

(C, PD or PR, depending on the scheme selected), and normalizes the resulting values across all operators (normalization is achieved by dividing values by the highest possible value). The normalized values are stored in 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 to the operator selection module an updated history of the performances of each operator.

7.4.4 Operator Selection

The operator selection module selects the next operator to be applied based on its past performances, but without neglecting an exploration of the possible available operators. As mentioned in Section 7.3.2, *Ex-DMAB* is inspired by the multi-armed bandit approach, used in game theory. The strategy always chooses the operator that maximizes expression (7.1).

However, this expression 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 execution, its value of $n_{i,t}$ would be so low that the bandit-based AOS would have to apply it many times in order to adjust its exploration term with regards to the rest of the operators.

Since we are interested in operators that enter and exit the evolutionary algorithm during the search process, we have reformulated expression (7.1) in order to deal with a dynamic set of operators. This is mainly achieved by replacing the measure corresponding to the number of times that an operator has been applied with another criterion that corresponds to the number of generations elapsed since the last application of the operator (i.e., its idle time). This allows a new operator to immediately increase its evaluation by applying it once. The new evaluation of performance is then defined as follows:

$$MAB2_{o,t} = r_{o,t} + 2 \times exp(p \times i_{o,t} - p \times x \times NO_t) \tag{7.2}$$

where $i_{o,t}$ is the idle time of operator o at time t, NO_t is the number of operators considered by the *Operator Selection* at time t, and x is the number of times the controller must wait before compulsorily applying the operator o. The behavior of the exploration component is better understood by looking at Figure 7.6. The value 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 positive parameter that adjusts the slope of the exponential.

Besides *MAB2*, referred to as **M2** from here on, the following three different *Operator Selection* methods were also analyzed:

- **Random (R)**, simply chooses randomly which among the operators currently available in the EA.

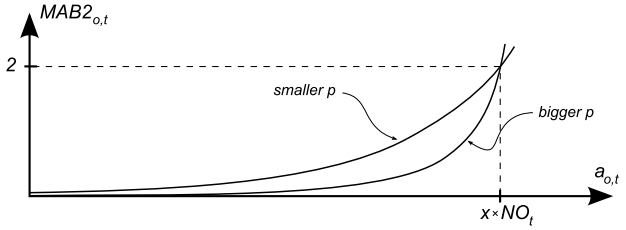

Fig. 7.6: behavior of exploratory component of expression (7.2)

- **Probability Matching (PM)**, which chooses the operator with a probability proportional to the reward values stored by the credit assignment module.
- **MAB2 + Stuck detection (M2D)**, which adds to M2 a method in order to detect the population is trapped in a local optimum. The detection is performed thanks to the linear regression of the values of the mean quality of the population 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 (i.e., almost a flat line), a diversification stage is performed, using only operators that have an exploration profile. This diversification stage is maintained until (i) the diversity reaches a range over the original value, or (ii) there are no exploration operators, or (iii) a predefined number of generations has passed without the desired diversity being reached.

7.4.5 Experimental Results

The combination of the three mentioned *Credit Assignment* mechanisms, namely Compass (C), Pareto Dominance (PD) and Pareto Rank (PR), with the four considered *Operator Selection* techniques (R, PM, M2, M2D), resulted in a set of 12 *Adaptive Operator Selection* combinations to be coupled with the proposed *Blacksmith Adaptive Operator Management*. These controllers were tested in the resolution of the satisfiability problem SAT, with the AOM continuously managing the operational set of ten operators by including and excluding them from a superset of 307 operators (described in Section 7.4.2); and the AOS autonomously selecting from this subset of ten available operators defined by the AOM which of them should be applied at each time instant, with both using the same assessment information provided by the *Credit Assignment* module.

The results were compared with state-of- the art crossovers FF, CC and CCTM [25, 18]. Besides, the Uniform crossover and a controller that simply applies one of the 307 possible combinations randomly (called *R307*) were used as baseline methods. Comparisons have been made on instances from different families and types of benchmarks, including nine crafted, ten random-generated and seven industrial

instances. One algorithm execution refers to 100,000 applications of crossover operators.

These combinations are 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 those of Blacksmith and those of M2 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 with a fresh one. If an operator has been sufficiently applied (half of the size of the registry, i.e., ten times) and if its reward is in the lower third of the 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.

Figure 7.7 shows the convergence of the EA algorithm coupled with the different controllers, as well as those of state-of-the-art and baseline crossovers, solving an industrial instance. It can be seen that the worst performance is obtained by the Uniform Crossover, and most of the 12 controllers do not overcome the state-of-the-art crossovers CC and CCTM. However, two of them (PD-R and PD-PM) are able to converge quite quickly to better solutions.

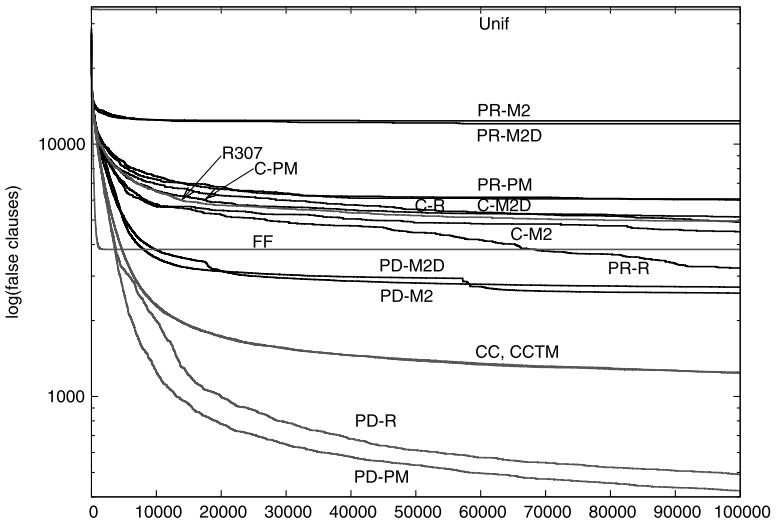

Fig. 7.7: Number of false clauses of best individual so far, obtained by different controllers and state-of-the-art crossovers when solving an industrial instance

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

There is a clear advantage with *Credit Assignment* based on the Pareto Dominance (PD), since it appears in the most efficient controllers, followed by C and PR. One possible explanation is that PR considers equally all the operators placed on the Pareto frontier (points in the Figure 7.5(c) with value 0). This induces a balance between the exploration and the exploitation tendencies, thus preventing the evolutionary algorithm from leaning to one side or to the other. 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 that follow the general tendency (points in the Figure 7.5(b) with higher values) allows the evolutionary algorithm to break the status quo and finally improve the quality of the population. This "flexible balance" is the main asset of this *Credit Assignment* method.

It is interesting to notice that the most exploratory *Operator Selection* methods (PM and R) have produced some of the best results. It could seem surprising that a random operator selection could be able to overcome sophisticated methods that carefully try to balance EvE at the operator selection level, such as the bandit-based approaches. A possible hypothesis for this could be that a mix of different crossover operators works better than a single one due the diversity of behavior the former can manifest; however, the poor results obtained by R307 show that simply mixing different operators does not produce good results.

Another, more plausible, hypothesis is that an exploratory *Operator Selection* method will provide updated information to AOM, since the rate of application of "bad" operators is higher than when an exploitative *Operator Selection* method is used. In this manner, R and PM rapidly produce the five applications needed to evaluate the unsuccessful operators and move them from *alive* to *dead* state.

Another factor to consider is that *Adaptive Operator Management* uses the same criterion as *Adaptive Operator Selection* to choose which operator will survive. Since AOM continually dismisses the worst operators, an important part of exploitation is done by AOM; thus AOS can focus exclusively on exploration. Differently stated, AOS performs exploration among the operators that AOM has decided to keep by using an exploitative criteria.

Table 7.3 shows the mean (standard deviation) of the performance of the algorithms for each problem over 25 executions. We show here only the two most successful controllers (PD-PM and PD-R). The best results (and those that are indistinguishable from them, using a Student t-test with 95% confidence) are in boldface. Here we focus on the most successful controllers, PD-R and PD-PM.

PD-R obtained top results on 19 instances and PD-PM on 17, while CC and CCTM were the best ones only five times. Even though the controller configurations obtained the worst results on two industrial instances, we observed the best improvements on this family of instances, especially on *I5* and *I6*, in which the adaptively controlled evolutionary algorithms obtained up to 260 times fewer false clauses than the best state-of-the-art crossover. The results presented in this comparison show the generality of PD-PM and PD-R on different families and types of instances. The overhead of this improvement is really moderate since the time dedicated to control represents less that 10% of the total execution time.

Table 7.3: 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)

7.5 Conclusions

Finding an appropriate parameter setting has a great effect on the performance of search algorithms. This parameterization can be fixed during the whole execution, or it can be continuously modified while the problem is being solved, adapting its features to the needs of each stage of the search, which is commonly known as parameter control.

Parameter control requires good knowledge of the mechanics of the search, and in many cases of the details of the problem. Actually, it is mostly achieved in a customized way, making it difficult to transfer the knowledge obtained through the control of one algorithm to another one. As a consequence, one of the main issues in parameter control lies in the creation of generic schemes that could be used transparently by the user of a search algorithm, without requiring prior knowledge about the problem scope and the details of the search algorithm.

In an intuitive way, we can distinguish betweentwo main types of parameters: those that control the *behavior* of the algorithm, such as the application rates of a given variation operator, and those that affect the core structure of the algorithm itself, called *structural*. In this chapter, we have discussed two approaches for dealing with both behavioral and structural parameters, namely *Adaptive Operator Selection* and *Adaptive Operator Management*.

ExCoDyMAB AOS, presented in Section 7.3.3, combines *Compass*, an efficient *Credit Assignment* approach proposed in [31] that associates two performance measures (quality and diversity) with a given operator, with *Ex-DMAB*, an engineered *Operator Selection* mechanism based on the multi-armed bandit approach, originally proposed in [15]. Such an AOS combination has been shown to be very efficient in selecting between ill-known operators within an EA applied to SAT problems. Although some meta-parameters still need to be tuned, off-line techniques such as the F-Race [6] can be used, which reduces the tuning time to around 15% of the total time that would be required by a complete factorial design of experiments.

To deal with *structural* parameters, we have defined a general framework in which operators can be in one of the following three states: unborn, alive or dead. The inclusion or exclusion of operators defines the transitions between these states. We have presented an *Adaptive Operator Management* called Blacksmith, which builds operators in a combinatorial way and attaches a profile to each one of them, in order to evaluate whether an operator must be kept, dismissed or reincorporated into the search. This AOM technique has been shown to have the ability to autonomously and continuously manage an efficient set of ten operators within an EA applied to SAT problems, extracted from a superset of 307 unknown operators. Even if one well-performing static configuration could be found, such a definition if done by hand would be very time-consuming.

Both AOS and AOM controllers are based on the same assessment of operators, referred to as *Credit Assignment*. In order to attain generality with regard to the implementation of the search algorithm, it is necessary to measure the performance of the controlled items in a generic way. This measure must make sense to a wide set of search algorithms and be compatible with the search itself. Two high-level criteria are used to evaluate the performance of operators: mean fitness and population diversity. These two measures are respectively aligned with exploitation and exploration, that constitute the two main goals of any search algorithm. Note that even though population diversity is closely related to population-based algorithms, an equivalent measure of exploration, such as a time-diversity diversity measure (e.g., one considering the amount of exploration that the point has performed in recent operations) could be used instead by single-point based algorithms.

Given this generality, the combination of both controllers, *Adaptive Operator Selection* and *Adaptive Operator Management*, could be used as a framework to manage and select variation operators not only for evolutionary algorithms, but for search algorithms in general.

A possible improvement to the approach presented in this chapter would be the inclusion of a high-level strategy to guide the behavior of both controllers. Notice that both AOM and AOS currently consider the same criteria to guide their choices,

i.e., Pareto optimality (or a similar idea) of both mean fitness and population diversity. In the way it is currently being done, the same weight is given to both trends, the exploration/diversification and the exploitation/intensification criteria; as is known, the needs of the search w.r.t. such trends vary as the search goes on; thus keeping the same balance might lead to overall suboptimal behavior. Intuitively, one idea would be to consider some strategy that starts by encouraging the exploration behavior, and gradually shifts to the exploitation; or one that searches for a certain compromise level of EvE [30, 32].

Another path of further research concerns the identification of the best operators used during one or more runs. Even though our approach can choose appropriate operators given a search state, no information about this knowledge is delivered at the end. This automatically extracted knowledge could be used to guide the selection of the better operators for the next run, or to establish a map between operators and the search states for which they are best suited.

Some further analysis should be done for both *Adaptive Operator Selection* and *Adaptive Operator Management*concerning the relevance and sensitivity of their meta-parameters, which would help us to better understand their behavior.

One could also remark that, in this architecture, bad operators should be tried before assessing their poor performances. Another architecture could be envisaged to separate the evaluation of the operators and their effective use. For instance, in a distributed solving scheme, some controllers could be devoted to the evaluation of the performance according to given state of the search while other controllers could concentrate on the effective solving of the problem, the controllers exchanging information about the current state of the search.

References

[1] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2/3), 235–256 (2002)

[2] Barbosa, H. J. C., Sá, A. M.: On adaptive operator probabilities in real coded genetic algorithms. In: Proc. XX Intl. Conf. of the Chilean Computer Science Society (2000)

[3] Bartz-Beielstein, T., Lasarczyk, C., Preuss, M.: Sequential parameter optimization. In: Proc. CEC'05, pp. 773–780. IEEE Press (2005)

[4] Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization, *Operations Research/Computer Science Interfaces*, vol. 45. Springer (2008)

[5] Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, *Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press (2009)

[6] Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: W. B. Langdon et al. (ed.) Proc. GECCO '02, pp. 11–18. Morgan Kaufmann (2002)

[7] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.: Handbook of Meta-heuristics, chap. A Classification of Hyper-heuristics Approaches. Springer (2009)
[8] Clune, J., Goings, S., Punch, B., Goodman, E.: Investigations in Meta-GA: Panaceas or Pipe Dreams? In: H. G. Beyer et al. (ed.) Proc. GECCO'05, pp. 235–241. ACM Press (2005)
[9] Cook, S. A.: The complexity of theorem-proving procedures. In: Proc. ACM Symposium on Theory of Computing, pp. 151–158. ACM Press (1971)
[10] Da Costa, L., Fialho, A., Schoenauer, M., Sebag, M.: Adaptive operator selection with dynamic multi-armed bandits. In: M. Keijzer et al. (ed.) Proc. GECCO'08, pp. 913–920. ACM Press (2008)
[11] Da Costa, L., Schoenauer, M.: GUIDE, a Graphical User Interface for Evolutionary Algorithms Design. In: J. H. Moore et al. (ed.) GECCO Workshop on Open-Source Software for Applied Genetic and Evolutionary Computation (SoftGEC). ACM Press (2007)
[12] Davis, L.: Adapting operator probabilities in genetic algorithms. In: J. D. Schaffer (ed.) Proc. ICGA'89, pp. 61–69. Morgan Kaufmann (1989)
[13] Eiben, A., Michalewicz, Z., Schoenauer, M., Smith, J.: Parameter Setting in Evolutionary Algorithms, chap. Parameter Control in Evolutionary Algorithms, pp. 19–46. Vol. 54 of Lobo et al. [27] (2007)
[14] Eiben, A. E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evolutionary Computation **3**(2), 124–141 (1999)
[15] Fialho, A., Da Costa, L., Schoenauer, M., Sebag, M.: Extreme value based adaptive operator selection. In: G. Rudolph et al. (ed.) Proc. PPSN'08, *LNCS*, vol. 5199, pp. 175–184. Springer (2008)
[16] Fialho, A., Da Costa, L., Schoenauer, M., Sebag, M.: Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In: T. Stuetzle et al. (ed.) Proc. LION'09 (2009)
[17] Fialho, A., Schoenauer, M., Sebag, M.: Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In: F. Rothlauf et al. (ed.) Proc. GECCO'09, pp. 779–786. ACM Press (2009)
[18] Fleurent, C., Ferland, J. A.: Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, pp. 619–652 (1996)
[19] Garey, M. R., Johnson, D. S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Company, San Francisco (1979)
[20] Goldberg, D.: Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. Machine Learning **5**(4), 407–426 (1990)
[21] Hartland, C., Gelly, S., Baskiotis, N., Teytaud, O., Sebag, M.: Multi-armed bandit, dynamic environments and meta-bandits. In: Online Trading of Exploration and Exploitation Workshop, NIPS (2006)

[22] Hoos, H., Stützle, T.: SATLIB: An Online Resource for Research on SAT, pp. 283–292. IOS Press, www.satlib.org (2000)
[23] Jong, K. D.: Parameter Setting in Evolutionary Algorithms, chap. Parameter Setting in EAs: a 30 Year Perspective, pp. 1–18. Vol. 54 of Lobo et al. [27] (2007)
[24] Julstrom, B. A.: What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm on genetic algorithms. In: L. J. Eshelman (ed.) Proc. ICGA'95, pp. 81–87. Morgan Kaufmann (1995)
[25] Lardeux, F., Saubion, F., Hao, J. K.: GASAT: A genetic local search algorithm for the satisfiability problem. Evolutionary Computation **14**(2), 223–253 (2006)
[26] Lobo, F., Goldberg, D.: Decision making in a hybrid genetic algorithm. In: T. Bäck et al. (ed.) Proc. ICEC'97, pp. 121–125. IEEE Press (1997)
[27] Lobo, F., Lima, C., Michalewicz, Z. (eds.): Parameter Setting in Evolutionary Algorithms, *Studies in Computational Intelligence*, vol. 54. Springer (2007)
[28] Maturana, J., Fialho, A., Saubion, F., Schoenauer, M., Sebag, M.: Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In: Proc. CEC'09 (2009)
[29] Maturana, J., Lardeux, F., Saubion, F.: Autonomous operator management for evolutionary algorithms. Journal of Heuristics **16**, 881–909 (2010)
[30] Maturana, J., Saubion, F.: On the design of adaptive control strategies for evolutionary algorithms. In: Proc. EA'07, *LNCS*, vol. 4926. Springer (2007)
[31] Maturana, J., Saubion, F.: A compass to guide genetic algorithms. In: G. Rudolph et al. (ed.) Proc. PPSN'08, *LNCS*, vol. 5199, pp. 256–265. Springer (2008)
[32] Maturana, J., Saubion, F.: From parameter control to search control: Parameter control abstraction in evolutionary algorithms. Constraint Programming Letters 4, Special Issue on Autonomous Search, 39–65 (2008)
[33] Moore, G. A.: Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. Collins Business Essentials (1991)
[34] Nannen, V., Eiben, A. E.: A method for parameter calibration and relevance estimation in evolutionary algorithms. In: Proc. GECCO'06, pp. 183–190. ACM Press (2006)
[35] Nannen, V., Eiben, A. E.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proc. IJCAI'07, pp. 975–980 (2007)
[36] Nannen, V., Smit, S. K., Eiben, A. E.: Costs and benefits of tuning parameters of evolutionary algorithms. In: G. Rudolph et al. (ed.) Proc. PPSN'08, pp. 528–538. Springer (2008)
[37] Page, E.: Continuous inspection schemes. Biometrika **41**, 100–115 (1954)
[38] Pareto, V.: Cours d'économie politique. In: Vilfredo Pareto, Oeuvres complètes, Genève: Librairie Droz (1896)
[39] Rice, J. R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
[40] Sywerda, G.: Uniform crossover in genetic algorithms. In: Proc. ICGA'89, pp. 2–9. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)

[41] Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: H. G. Beyer (ed.) Proc. GECCO'05, pp. 1539–1546. ACM Press (2005)
[42] Tuson, A., Ross, P.: Adapting operator settings in genetic algorithms. Evolutionary Computation **6**(2), 161–184 (1998)
[43] Whitacre, J. M., Pham, T. Q., Sarker, R. A.: Use of statistical outlier detection method in adaptive evolutionary algorithms. In: M. Cattolico (ed.) Proc. GECCO'06, pp. 1345–1352. ACM Press (2006)
[44] Wolpert, D. H., Macready, W. G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)
[45] Wong, Y. Y., Lee, K. H., Leung, K. S., Ho, C. W.: A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. Soft Computing **7**(8), 506–515 (2003)
[46] Yuan, B., Gallagher, M.: Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In: X. Yao et al. (ed.) Proc. PPSN'04, pp. 172–181. Springer (2004)