Constraint Programming Letters 4 (2008) 39-65 Submitted 7/2008; Published 12/2008

From parameter control to search control: Parameter Control Abstraction in Evolutionary Algorithms

Jorge Maturana MATURANA@INFO.UNIV-ANGERS.FR
Frédéric Saubion SAUBION@INFO.UNIV-ANGERS.FR
LERIA, Université d'Angers
2, Bd Lavoisier 49045 Angers (France)

Editors: Youssef Hamadi, Eric Monfroy and Frédéric Saubion

Abstract

This paper presents a method to encapsulate parameters of evolutionary algorithms and to create an abstraction that simplifies the control and the understanding of the internal behavior of the algorithm. A fuzzy model is used to learn the effects of parameters over the search process. Then, high-level strategies can be defined to modify parameters automatically in order to achieve a scheduled level of balance between exploration and exploitation during the search. We experimented supervised control strategies and autonomous schemes that adjust parameters dynamically. Experiments have been performed on the Quadratic Assignment Problem in order to analyze the strengths and weaknesses of each approach. Possible improvements of the general methodology are also discussed.

Keywords: Parameter control, evolutionary algorithms, fuzzy logic controllers, machine learning, adaptive control

1. Introduction

Evolutionary algorithms (EAs) (Michalewicz, 1996) have been originally inspired by natural evolution. Given a problem, a population of individuals that encodes candidate solutions, evolves by means of genetic operators. Those operators, namely mutation and crossover, may alter one individual or combine the information of two individuals to produce offspring. The best individuals are then selected to survive, depending on a fitness measure. Given this general formulation, EAs have been used as general purpose solvers and successfully applied to a wide range of optimization problems in various domains including combinatorial optimization, such as planning, timetabling, scheduling or global optimization (i.e., with continuous variables). Specific knowledge on problems domains and structures can be used to design specific operators, which often improve the search process.

Several EA features such as application rates of operators, population size, selection pressure or even characteristics of particular operators may be subjected to parameters. The correct setting of those parameters has a crucial effect on the ability of the EA to properly solve specific problems. This setting is required because every problem has different characteristics and must be solved in a special way. The "No Free Lunch" principle (Wolpert and Macready, 1997) stands that a particular solving method is efficient only within a restricted scope. Therefore, the most natural way to improve the efficiency of EAs is to parameterize them to handle various problems with different characteristics. However, parameter setting and tuning are difficult to achieve for, at least, the following reasons:

- Parameters are problem-dependent.
- "Optimal" values of parameters are not stable along the search. For instance, it is frequently admitted that the search space must be widely explored before concentrating in the most interesting areas. Therefore, static parameters lead to sub-optimal searches.
- The effect of some parameters is often a priori unknown. For example, it is difficult to forecast precisely how and how much a specific operator will affect the concentration of individuals in the search space.
- Parameters usually interact in a complex way, so a single parameter will have a different effect depending on the value of the others.

Parameter control is not a new issue and huge efforts have been done in order to discover the influence of canonical and special parameters (cf. Section 2). The main directions of these efforts are *parameter setting*, i.e., finding optimal fixed parameters for the whole run, and *parameter control* where parameters are adjusted during the run based on several criteria (see Eiben et al., 2007).

It must be noted, however, that most of the literature on this subject focuses on the study of specific parameters within specific algorithms to solve particular problems. There is a lack of high level criteria when designing control strategies. From a general point of view. This restricts parameter control to specialists and limits the potential use of EA by a more extended range of users on a wider set of problems. Excluding Evolutionary Strategies (see Beyer and Schwefel (2002)), when EAs are applied to real world problems, parameter tuning (control is almost nonexistent) is carried out by quite rudimentary methods, usually time-consuming series of trial and error runs. Therefore, it would be interesting to propose a method that could be used by non-specialists, including the following characteristics:

- It should provide an abstraction of parameters, to focus the control on *"how to guide the search"* rather than *"how to adjust the parameters"*. This abstraction must be general enough to be applicable to a broad range of algorithms.
- It should work with nonstandard parameters, in order to do not restrict user's possibilities to create new features. A method that works only with some specific parameters would not be useful to applied-EA practitioners.
- It should be easy to integrate within any EA. The goal of the user is not to create a complex control mechanism, but merely to solve her/his problem. Therefore, control must be available at minimum effort.
- It should help to save user's time. Although adjusting parameters is indeed a part of problem solving, it is a mechanical process that deviates the users from their primary goal. Control must be as autonomous as possible.

This paper discusses these issues, based on a method previously investigated in Maturana and Saubion (2007a,b). An intuitive idea of this approach is schematized in Figure 1: the controller assigns values to parameters of the EA. The EA computes a generation with this setting and informs the controller about the diversity of the population and the quality associated to this setting.

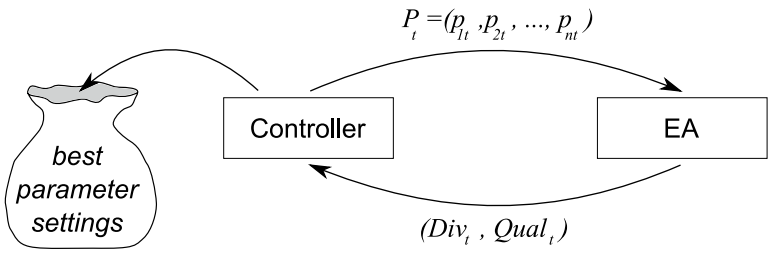

Figure 1: General scheme of the interaction between controller and EA

The controller uses this measures to model the influence of parameter settings over the algorithm performance, and keep the best settings –according to a given diversity and quality– to be used later.

This method simplifies parameter control by creating an abstraction of multiple parameters. This abstraction permits to control the EA with a single parameter, which is related to a high level concept: the balance between exploration and exploitation. In this way, the control strategy can be handled in terms of increasing or relaxing exploitation, which is easy to understand by any user.

When dealing with huge a search space, the management of the balance between exploration and exploitation constitutes a key factor of a successful search. On the one hand, an algorithm should be able to visit scattered areas of the search space, on the other hand it should have the ability to focus on specific zones in order to identify local optima. These two complementary tasks are required to reach a global optimum or, at least, a sub-optimum of good quality. Nevertheless, as mentioned above, the relationship between the parameters of the algorithms and these two high level search strategies is difficult to manage.

In this work, our main motivation is to provide an abstraction of parameters to generalize control. This abstraction will allow the user to think in more general terms and will facilitate her/him the task of constructing autonomous control approaches (cf. Section 4).

Our method consists of two phases:

1. *Learning* is dedicated to understand how the parameters affect the search and to model their behavior. Examples are generated for combinations of parameters values and a fuzzy logic model is used to store the acquired knowledge.

2. *Control* uses the acquired knowledge to guide the search. The model built in the previous phase is used, altogether with a search strategy, to dynamically adjust the parameters during the run, with regard to the required level of exploration and exploitation.

In this context, three important aspects must be considered: how to collect examples for learning, how to obtain the model, and how to create the strategy to guide the search.

Paper overview

This paper is organized as follows. Section 2 presents briefly the relevant work on this subject. Section 3 presents the method, discussing the different our aspects mentioned above. Section 4 shows experimental setup of experiences, and Section 5 discusses results. Finally, Section 6 provides some conclusions an future guidelines.

2. Related Work

2.1 Parameter Control in EA

Figure 2 presents the taxonomy of parameter setting methods, proposed by Eiben et al. (2007). Two main groups are defined, depending on whether parameters are modified (control) or remain static during the run (tuning). A subclassification of parameters control distinguishes different ways to adapt parameters: if parameters are changed in a deterministic way, for instance as a function of the number of elapsed generations, control is called *deterministic*. If changes are related to the current state of the search it is called *adaptive*. Finally, if parameters are coded inside individuals and evolve with them, control is called *self-adaptive*.

Each approach has its own advantages and drawbacks. Parameter setting is simple to implement, but convenient parameter values are difficult to find. Moreover, they can be well suited for a particular moment of the search, rather than for the whole execution. Deterministic control solves this problem by adjusting parameters based on deterministic rules. However, the timing of application of these rules can be inaccurate and depends on the search. A good knowledge is therefore required to set up a good schedule. Self-adaptive has the problem of increasing search space, because the EA must solve two problems simultaneously: to find the correct parameters and to solve the original problem. Adaptive control is faster than auto-adaptive control, but there exists a problem for defining a performance measure to provide a suitable feedback to control.

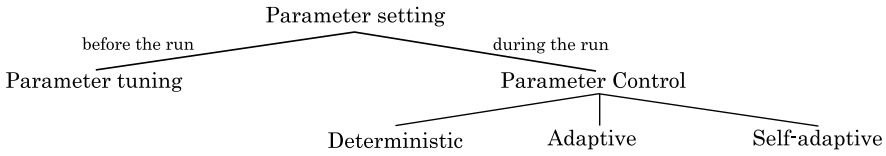

Figure 2: Taxonomy of parameter setting proposed by Eiben et al. (2007)

Within adaptive control, the state of the search is constantly monitored and changes are made in values of parameters according to some criteria. The measure of parameters performances can be expressed as a fixed aim, such as the 1/5 success rule of Rechenberg (1973), that expects one successful mutation out of five; or in a competitive way, as in Thierens (2007), where the most successful parameter combinations are rewarded. Common measures of performance involve the ability of parameters to produce improved offspring, despite the fact that it is necessary to accept fitness worsening to escape from local optima.

Within adaptive parameter control we may distinguish two main perspectives. The first one involves a learning method to understand the effect of parameters over the performance of the EA. The second one, on the contrary, assumes a rule that links the performances with parameter values. We now detail these approaches.

- In the first approach, adopted in our work, a function $performance = P_1(parameters)$ is obtained by learning how the parameters affect the performance of the algorithm. To determine P_1, a series of experiments with different parameters values are run and resulting performances are monitored during some generations. Once the experiments are done, the function P_1 is adjusted. Since the shape of P_1 (i.e., the shape of the plot: linear, polynomial, exponential, periodic, etc.) is a priori unknown, a flexible enough modeling technique is needed to adjust it. The advantages of this method is that there is no assumption about P_1,

so several different parameters can be studied and modeled at the same time. Function P_1 can also be used to provide information about internal behavior of the algorithm. It can be useful to understand, for example, the effect of some operator rates over quality, or how the population size prevents the loss of diversity. The drawbacks include the extra execution time corresponding to learning generations and the fact that some effects could be not stable, thus gathered information can expire in a short time. Wong et al. (2003) proposes an algorithm divided in periods of learning and control of parameters, by adjusting central and limit values of them. Kee et al. (2001) presents two methods including a learning phase that tries different combinations of parameters and encodes the results in tables or rules.

- The second approach requires some prior knowledge about the algorithm. Here, the function $parameters = P_2(performance)$ is a priori known. This approach is typically used when adjusting application rates, by awarding successful operators to raise their future probability of being chosen. Since no learning period is performed, the main advantage of this approach is the speed of execution. However, the encoding of some parameters other than operators rates is not obvious. This is by far the most common approach. Thierens (2007) presents a controller that adjusts operators rates according to recent performances. Similar ideas are presented in Igel and Kreutz (2001) and Lobo and Goldberg (1997). In Whitacre et al. (2006), this approach is extended by considering several statistics of individuals fitness and survival rate to evaluate operator quality. In Eiben et al. (2004), the population is resized, depending of several criteria based on the improvement of the best historical fitness. Eiben et al. (2006) modify parameters according to best fitness value. Some methods in this class require special features from the GA, such as Lis (1996), that maintains several populations with different parameter values and moves the values of parameters toward the value that produces the better results. In Tsutsui et al. (1997), a forking scheme is used: a parent population is in charge of exploration, while several child populations exploit particular areas of the search space. In Harik and Lobo (1999), a parameterless GA gets rid of $popsize$ parameter by comparing the performance of multiple populations of different size.

As far as we know, no effort was made in order to build a real abstraction of parameter control. Let us illustrate the importance of this abstraction with an example. Imagine that we are using an EA to solve an optimization problem and that we have noticed that the population tends to concentrate to one local optima. Thus, we decide to raise mutation rate in order to escape from there, and later reach a global optimum. This could seem obvious, since mutation is seen as the exploring operator *par excellence*. However, what we wanted was *diversity*, not more mutation. What if there is another operator that could spread the population without the disruption that mutation causes?. Indeed, when dealing with specialized operators with ill-known effects, there might be another operator capable to produce “good diversity”, i.e., diversity with a controlled loss of quality.

The relevance of building an abstraction of EAs lies in a human factor: a good abstraction is much easier to handle than a set of low level variables complexly related. Cars would have never become popular if, instead of a gear and pedals, users would have to deal with geared wheels, differential gears, and valves to control fuel and oxygen flows. A clear and simple interface facilitates the encapsulation of modeling mechanisms, not to mention that an autonomous control scheme is also easier to be defined by the user when a good abstraction is used.

2.2 Fuzzy Logic Controllers

In order model functions with an unknown shape, it is necessary to use a support flexible enough to approximate any kind of functions. Analytic models (e.g., linear, polynomial, or other) are useless because they assume a shape that could not represent the effects of parameters over search performance. In this work, we use Fuzzy Logic Controllers (FLC).

Fuzzy Logic (FL) is an extension of classic Boolean logic. In Boolean logic, a sentence is either absolutely *true* or *false* while FL admits an infinite number of levels of truth that are expressed by a membership function with values ranging from 0 (false) to 1 (true). FL better expresses imprecise notions such as "cold", "far" or "slow".

One of the most useful applications of FL are FLCs (Kulkarni, 2001; Piegat, 2001). FLCs permit to infer answers from rules such as *"IF car_speed is high AND road is dry, THEN risk is medium"*. Figure 3 shows the general structure of a FLC. The first step is the transformation of crisp input –real numbers– to their corresponding fuzzy expression. Then, an inference engine obtains the fuzzy output based on fuzzy rules, and at last, fuzzy output is translated into a crisp output using a defuzzifier.

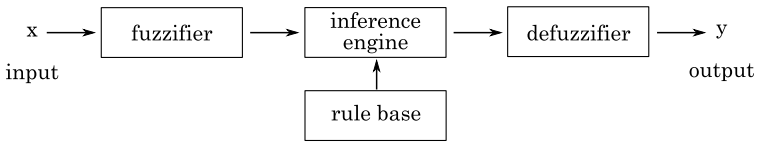

Figure 3: General scheme of a Fuzzy Logic Controller (FLC)

There are several variants of the "standard" FLC described above. The particular FLC used in this work is known as Takagi-Sugeno (Takagi and Sugeno, 1985). In this controller, the output variable is not expressed in a fuzzy way, but directly by a function of input values, thus defuzzification is not necessary. Since FLCs are universal approximators of continuous functions (Buckley, 1993) they act as modeling tools that express the output with relation to inputs.

A pioneer work applying Mamdani FLCs was proposed by Wang and Mendel (1992). In this work, a function $y = f(x_1, x_2)$, is modeled using Mamdani FLC from experimental data (x_1, x_2, y). This is done by dividing the input space in a grid, and finding the characteristic value of y for each cell. Many methods have been based on this article. Costa-Branco and Dente (1999) have studied the effects of noise and the quality of examples in the generation of Mamdani FLCs, pointing out that Wang and Mendel's method, which uses just a few examples to create FLCs, is vulnerable to noisy data.

3. Method Overview

This section presents the method we have developed to create an abstraction of parameter control. The method includes an initial gathering of examples, which is explained in Section 3.1. Later, the collected examples are used to build a model (Section 3.2). Finally, the model is used to control the search by adjusting the value of a single parameter, guided by a strategy (Section 3.3). Of course, all this processes is presented as a black box to the user, which benefit from a simplified view of the method, explained in Section 3.4. Additionally, the user can take advantage of the data gathered

during the process, in order to obtain a deeper knowledge about the EA. This is explained in Section 3.5.

The method consists of two main phases. The first one is dedicated to understanding the effect of parameters over the search. We propose to use two measures of performance: diversity and quality of population. In this work, we use the mean fitness of the individuals as quality and a measure of dissimilarity among individuals for diversity. The diversity measure depends on the encoding used (see Section 4.3 to obtain details of the diversity measure used in this work), while the mean fitness could eventually be replaced by the fitness of the best individual, or another central tendency measure.

Diversity has been chosen because it is highly related with the balance between exploration and exploitation (EEB). A low level of diversity means that all individuals are concentrated in some areas, evidencing an exploitation stage. On the other hand, a high level of diversity indicates that individuals are spread over the search space and reveals an exploration stage. Note that this is not always true, since an algorithm that efficiently solves a multimodal problem with a small population could have diverse individuals at maximal exploitation. Even if one could discuss whether this situation is uncommon or not, diversity seems to be a good compromise measure between a genuine expression of EEB and easiness of understanding and implementation.

After the model has been obtained, it is necessary to find out the better parameter combinations. Here two characteristics are considered as desirable: a high diversity, in order to avoid getting stuck in local optima, and high quality. Since these are conflicting objectives, the combinations of parameters corresponding to Pareto front (Pareto, 1896) are identified. Pareto front is used in multiobjective optimization and corresponds to the set of points from which no other point is better in all individual objectives measures.

Note that obtaining the Pareto front of parameters values only reduces the number of possible settings. Another criterion is still necessary to choose which of those settings will be applied in a specific situation, i.e., which level of diversity will be required by the algorithm. Indeed, the learning phase only builds the abstraction, so in the second phase a diversity variation strategy will be defined.

Figure 4 shows the main states of the controller, invoked by the EA in each generation. States 1 to 5 correspond to the first Learning phase, and State 6 to Control phase.

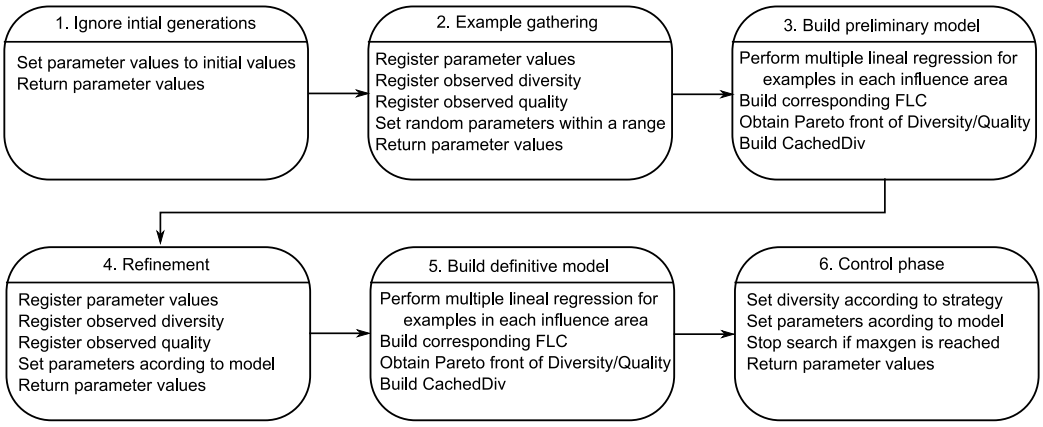

Figure 4: Main states of the controller

3.1 Learning

In order to collect the examples, the space of parameters is divided by placing fuzzy partitions for each parameter. Each intersection of fuzzy partitions is called an *Influence Area*, depicted as a round-corner square in Figure 5.a. This division is further subdivided to obtain a finer training grid (the motivations of this subdivision will be explained later). The factor of this subdivision is called *fineness*.

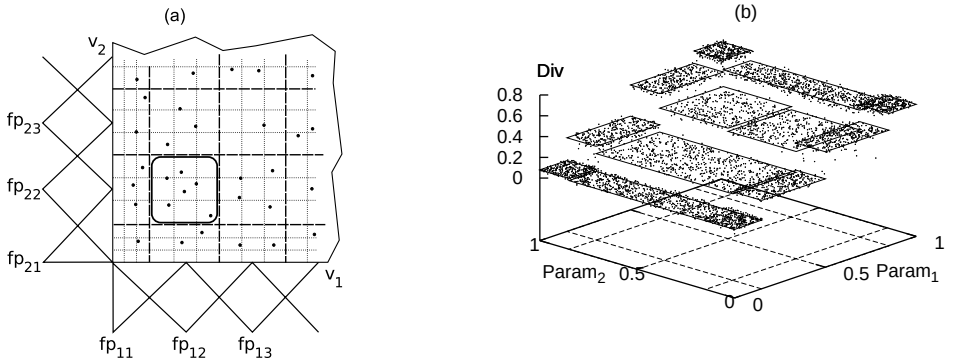

Figure 5: (a) influence area (fp_{12},fp_{22}) for two dimensions, in a partition with fineness of 3. (b) Formation of platforms (emphasized by squares) in a 4x4 coarse training grid

The learning phase is divided in five subphases:

1. *Ignoring initial generations*. In order to wipe out the high diversity and low fitness of the random-generated initial population, a number of generations is ignored at the beginning of the run;

2. *Example gathering*, in which learning examples for every fuzzy partition combination are generated;

3. *Preliminary model building*, where diversity and quality FLCs are built, based on earlier collected examples

4. *Refinement*, in which new examples, focused in the most promising areas, are generated to fine-tune the model;

5. *Definitive model building*, where all examples are used to build the definitive model, which is released to be used during *Control* phase.

Three main problems arise during this phase: *dimensionality*, *inertia* and *noise*. *Dimensionality* is related to the fact that the amount of examples to be generated depends exponentially on the number of controlled parameters. *Inertia* is related to the resistance to the change of diversity and mean fitness values between consecutive generations. Here, we understand *noise* as the short-term variation product of random operators that induce inaccuracy in modeling.

A symptom of inertia can be observed in Figure 5.b. Here, a coarse grid of 4×4 divides the 2-dimensional parameter space (shown in the base of the graphic), and the z-axis corresponds to

diversity. All the examples within each influence area were generated before passing to the next one. Although the surface is designed to be continuous, the inertia of diversity flattens the data in each cell. Even when a 4×4 fuzzy partition is enough to model the surface, a finer subdivision is required to generate examples, what justifies the subdivision of influence areas.

In order to avoid abrupt changes in parameter values, we have defined a special visiting order, called *smooth*, that moves between positions with the minimal possible change. Figure 6 shows examples for 2 and 3 parameters in contrast with classical "nested loop" visiting order.

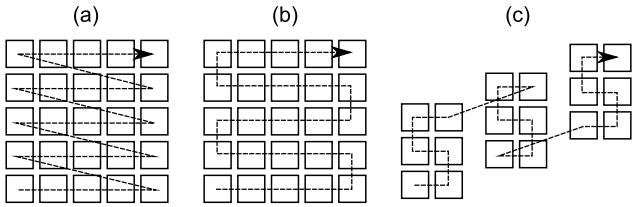

Figure 6: Visiting orders: (a) classical "nested loop", (b) smooth in 2D, (c) smooth in 3D

In order to exclude initial generations that present diversities and fitness levels caused by random population creation, the algorithm ignores a number of generations in the beginning of the run. To consider long-term operators (like mutation, whose beneficial effects are not appreciated instantaneously), mean fitness is corrected by assigning an exponentially-descending weighted average of their own values and the following ones.

Once all examples have been generated, the algorithm models the functions $Div(P)$ and $Fit(P)$, that express the relation of parameters with diversity and mean fitness respectively, using FLCs (cf. Section 3.2).

Before passing to *Control* phase, a second example gathering is performed, using the values obtained during the first modeling phase, plus a normal-distributed error, to finely explore the most interesting combinations and their surroundings. Later, all collected examples are considered to build the models of Diversity and Fitness again.

3.2 Model building

In Takagi-Sugeno FLCs, output variables are expressed by a function of input variables. When FLCs are used with modeling purposes, it is necessary to infer this function from examples. Figure 7 shows an example of fuzzy modeling. Suppose that we want to model a noisy function $f(x)$ with data pairs (x, y) obtained experimentally (points in the figure). The first thing to do is to divide the domain, in order to model the unknown function $f(x)$ by intervals. In each partition, a polynomial function is adjusted, for example, by minimizing the mean root square of errors. Figure 7.a shows 4 partitions and the polynomials of degree 1 (lines) that adjust each partition. To obtain the whole model of $f(x)$, the polynomials are combined according to the value of membership functions, shown at the bottom of the plot in Figure 7.b. Dashed line shows the final model.

In this work, the regression is performed with regard to the m parameters controlled, i.e., the hyperplane $\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m = 0$ is calculated for each influence area, and then used to model the entire function.

Figure 8 shows an example of resulting models. Here two parameters, mut and rep, are considered. Those parameters control the application rates of two operators, mutation and repairing, respectively. The surfaces represent diversity and mean fitness as function of the parameter values.

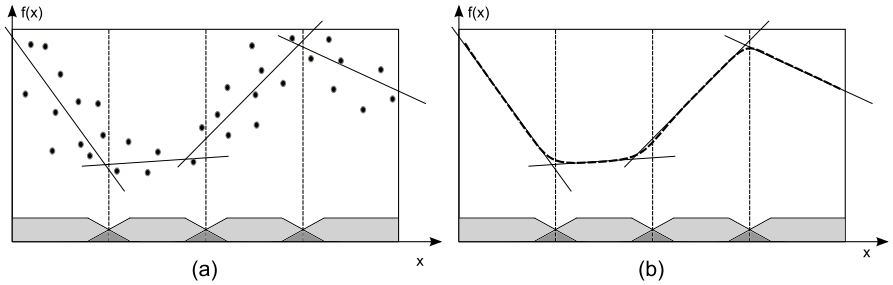

Figure 7: Fuzzy Modeling: (a) domain is divided and a linear regression is performed for the points in each partition, (b) Function is assembled from the polynomials

However, what we need during the Control phase is exactly the opposite: to fix the values of parameters that would produce a required level of diversity. In order to build the inverse function, we are interested only in the values that maximize the expected mean fitness of the population for different levels of diversity.

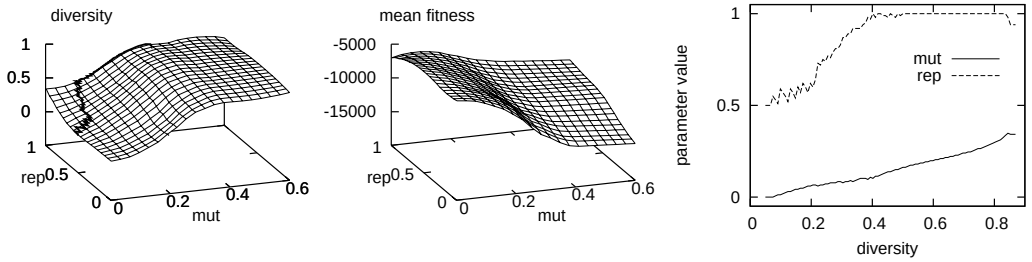

Figure 8: Information obtained from *Learning* phase: Diversity and mean fitness surfaces for two parameters, and plot of corresponding values of cache table

The strong line over the diversity surface shows the values of parameters that produce all values of diversity –within the reachable range– with the higher level of quality, according to the mean fitness model. This line is stored as a table (at right in the figure) named *CachedDiv*, that shows, for every value of diversity (in the x-axis) the optimum combinations of values of parameters. From here on, the controller only refers to this table.

3.3 Control phase and strategy design

One advantage of handling Learning and Control separately is that we can make a total abstraction of algorithmic details in the following. Note that during this phase, the control is fully abstracted, since the only issue to consider is to modify the balance between exploration and exploitation along search. This is done by modifying the value of diversity.

The challenge in this phase is to find a strategy to set diversity values along search, in order to avoid being trapped in local optima and, at the same time, to properly exploit the search space to speed-up the search. Several criteria can be considered here: Should diversity stand in an inter-

mediate level?, Must it move to produce alternate between exploration and exploitation?, Must it decrease slowly in an exploration to exploitation like in simulated annealing?.

If an excessive diversity is allowed, a sparse exploration will occur, loosing computation time. On the other hand, if diversity is not sufficient, it is likely that premature convergence will occur, losing potential useful information. These two arguments support the use of an intermediate "correct" level of diversity. Naturally, different problems have different values for this "correct" diversity. Therefore, the controller algorithm must be able to find it. An approach to do this consists in considering the variation of fitness value of the better individual of the population, during recent generations. If the same value is often repeated, is likely that the population is converging to the area where the related point is located.

Another possible approach is to start from an initial period of exploration, shifting to an exploitation one. This could help to first identify the most promising areas to progressively concentrate the search on them. However, note that if the search space is too wide or rugged the population could not well identify the best areas and could be trapped in local optima. In this case, a possible strategy could be to perform this shifting from exploration to exploitation several times. Actually, during preliminary experiments we have noted that some problems were easily solved just by zigzagging between the extreme values of diversity.

When diversity is increased to escape from local optima there are some aspects to consider. The first one is how high the diversity must be raised. If it is too low, individuals could stay in the same area of the search space and converge to the same optimum when performing exploitation again. If the value is too high, there exists the risk of loosing important information. The second aspect is the number of high-diversity generations that will be performed. Too few or too many can cause the same effects discussed before. Here we have considered a period of "forgetting", which consists in raising diversity to its maximal value during a number of generations. Since parameters are automatically set to obtain the higher possible mean fitness, the lowering on quality is roughly retained.

We have also experimented a small oscillation of diversity around nominal level, in order to perform a local exploitation/exploration and to help stabilizing the value of actual diversity compared to commanded one.

In order to compare different strategies, the considerations mentioned above were included in the following four strategies:

- **MX (Mixed):** that integrates first-explore-then-exploit, forgetting and the small oscillation. A series of intermediate descending diversity levels are commanded to the EA, with an oscillation above and below the nominal level. A number of generations are executed at each level, which are extended in case of finding an historical improvement. Once the algorithm has achieved the lower level, diversity is raised to its maximum value for a while, in order to escape from local optima. After this, the same scheme is repeated again.

- **CD (Correct Diversity):** to test the "correct" diversity concept. Every 10 generations, the fitness values of the best individuals of the last g generations are considered. If more than $\frac{g}{2}$ of those values are repeated, diversity is increased, and if less than $\frac{g}{8}$ are repeated, it is decreased.

- **ZZ (ZigZag):** that implements a wide oscillation around a central value of diversity. This value is given by the mean of commanded diversities corresponding to the last five historic

improvements. The oscillation, centered at this point, grows until the limits of possible diversity. If an historic improvement is reached, the amplitude of the oscillation is reset to zero, to start growing again.

- **FX (Fixed):** mainly for purposes of comparison. The algorithm is executed with a fixed value of diversity.

Different levels of autonomy can be identified in those strategies during the search. FX has no autonomy at all: it is not able to adapt itself to changes of the search states. ZZ and MX have a higher level of response to changes: the strategy is modified on line according to the observed events. Finally, the most reactive and autonomous strategy is CD, which constantly monitors the search state to adapt the level of diversity required.

3.4 From a user point of view

Figure 9 shows how the main loop in the EA links up with the controller. There are basically two calls to methods of the controller, one to ask for new parameters values, and the other to provide it with feedback.

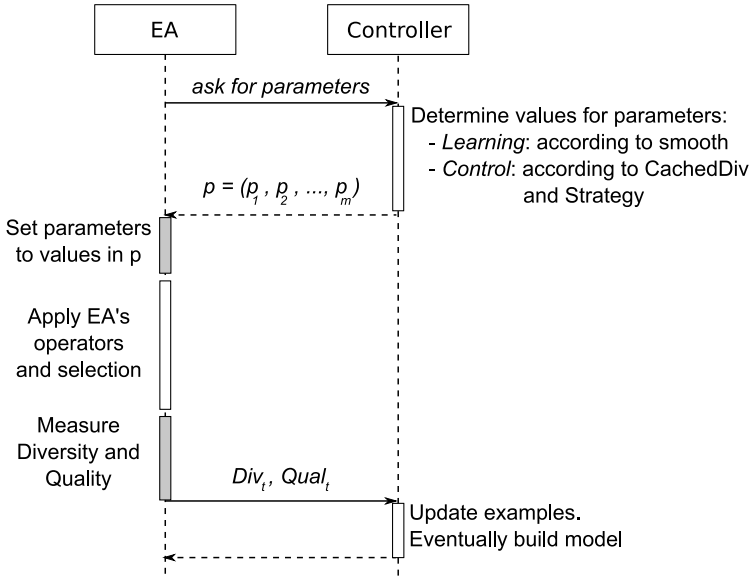

Figure 9: Interaction between the EA and the controller

From a user point of view, the implementation does not represent a big programming overhead. There are basically two invocations to the controller, and two new methods to implement (marked in grey in the figure). The first method must set the values suggested by the controller into parameters variables. The second one measures diversity and quality to provide feedback to the controller.

3.5 Analysis Tool

The learning phase produces two results that can be useful for the study of internal EA behavior. Let us consider the plots in Figure 8, that show diversity and mean fitness control surfaces, and a plot

of CachedDiv. By analyzing diversity FLC, the strong effect of mutation can be clearly observed: diversity increases with mut until reaching its maximal level around $mut = 0.3$. On the other hand, rep, that has less influence over diversity, has an important effect over mean fitness. Indeed, when both values are big ($mut > 0.3$ and $rep > 0.5$) a joint effect happens: while mutation rises the diversity, with the side effect of disrupting it, repairing fixes the errors and prevents an excessive fitness decrease.

Diversity and Fitness FLCs, and mainly CachedDiv can be used to obtain valuable information on the controlled EA. A somewhat flat line in CachedDiv indicates a parameter with a minimal or null effect over EEB, whose value is better to fix to the value that appears in the ordinates axis of CachedDiv. If two ore more parameters have a similar behavior they can be handled together. If two parameters are complementary (for instance, if their sum is roughly the same) one of them can be eliminated from control and replaced by a rule in order to obtain their value from the other.

Although this small explanation-oriented example is not very impressive, this approach becomes a valuable tool when dealing with three or more parameters, and the effort to figuring out the values of parameters in order to obtain a given diversity becomes intractable.

4. Experimental Setting

Since our method handles learning and control separately, validation is twofold. Firstly, we want to verify that the CachedDiv table contains the parameter combinations that actually produce the desired diversity and good-quality populations. Secondly, we want to find out which control strategy produces the best results for several problems. The controller algorithm works over an EA that solves several instances of Quadratic Assignment Problem (QAP). Our aim is not to be competitive for this particular problem, but to compare the performance of the controller using different strategies.

4.1 Quadratic Assignment Problem

The QAP is a well-known combinatorial optimization problem that can be stated as follows. Let us consider two matrices $A = (a_{ij})_{n\times n}$, $B = (b_{kl})_{n\times n}$, and a mapping function Π. The goal is to find a permutation $p_i = (\pi(1), \pi(2), \ldots, \pi(n))$ that minimizes:

$$f(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}$$

This problem was formulated by Koopmans and Beckmann (1957) for a facility allocation problem, in which a set of n facilities with physical flows between them (matrix A) must be placed in n locations separated by known distances (matrix B). The goal is to minimize the $flow \times distance$ of the whole system.

A set of 38 medium-size instances, obtained from the QAPLIB repository (Burkard et al., 1997), was selected to test the algorithm, covering instances from all families.

4.2 Evolutionary Algorithm

The individuals are encoded as permutations. Population size is fixed in 100 individuals and three operators are applied: standard exchange mutation, that interchanges two allocations randomly,

cycle crossover (Oliver et al., 1987), and a specialized operator (*remake*) that randomly erases four allocations, tries the $4!$ possible reconstructions and keeps the better one. Selection is achieved by roulette wheel (Holland, 1975). 15 runs of 10.000 generations (learning not considered) have been performed for each instance and strategy. This great amount of generations was defined to observe how definitive is premature convergence in every case.

4.3 Diversity measure

Using permutation encoding, we measure diversity as the complement of the similarity between individuals. Let the similarity of the population be

$$sim = \sum_{i=1}^{popsize} \sum_{j=1}^{popsize} \sum_{k=1}^{n} c(i,j,k) \quad i \neq j \tag{1}$$

Where n is the number of variables in the problem and the function c is defined as

$$c(i,j,k) = \begin{cases} 1 & \text{if variable } k \text{ in individuals } i \text{ and } j \text{ have the same value} \\ 0 & \text{otherwise} \end{cases}$$

In order to normalize similarity (thus diversity), we need to find lower and upper bounds. Upper bound is reached when all individuals are the same, therefore upper bound is

$$u_b = popsize \cdot (popsize - 1) \cdot n$$

To calculate the lower bound is less evident. If $popsize \leq n$ (Figure 10.a), lower bound is 0, with a population whose values are shifted each time. In general, lower bound is defined by the following formula

$$l_b = \begin{cases} 2 \cdot n \cdot A \cdot B & \text{if } A < 2 \\ 2 \cdot n \cdot A \cdot B + \frac{A!}{(A-2)!} \cdot n^2 & \text{if } A \geq 2 \end{cases}$$

Where A and B are the quotient and rest of the integer division between $popsize$ and n, respectively. A corresponds to the number of blocks of minimum similarity, shown in the figure as grey blocks. B is the number of individuals that do not belong to a complete block. The minimum similarity between two blocks is $2n^2$, and each individual alone contributes with $2n$ for each existing block (Figure 10.b). The expression $\frac{A!}{(A-2)!} \cdot n^2$ corresponds to similarity contributed by the permutation of all existing blocks (Figure 10.c), while $2 \cdot n \cdot A \cdot B$ expresses the similarity contributed by individuals alone.

Having lower and upper bounds of similarity, we can normalize it, or, what is more important in our case, find the normalized diversity of the population, which is given by:

$$div = \frac{u_b - sim}{u_b - l_b}$$

Computation of diversity Lower and upper bounds are calculated once, while similarity is computed for each generation. The computational complexity of equation 1 is $O(np^2)$, where n is the number of variables and p is the population size. Nevertheless, this complexity can be reduced by creating some data structures and dividing the computation into two steps:

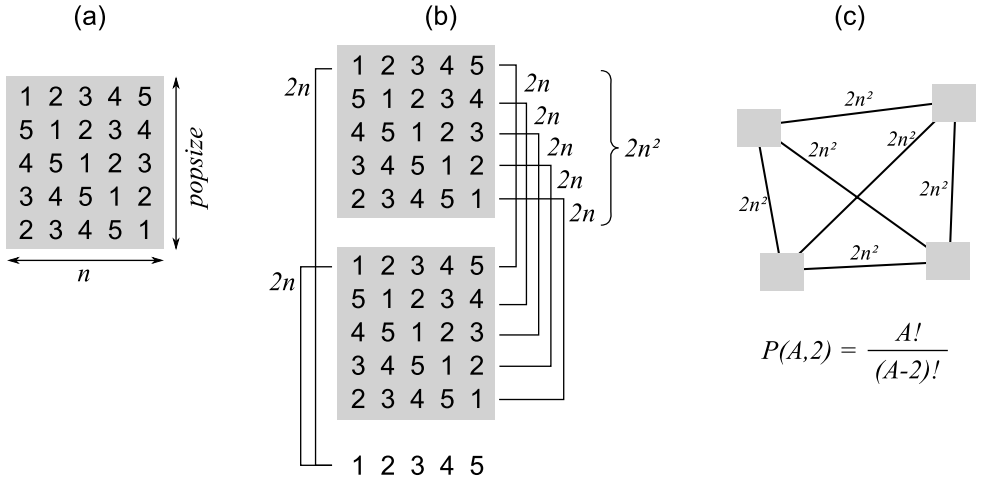

Figure 10: lower bound of similarity for permutation encoding

1. Create a matrix $A = (a_{ij})_{n \times n}$, that will store the number of appearances of variable i at position j. Visit all variables of individuals to fill matrix A (order $O(np)$).

2. Obtain the similarity as $\sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij} \cdot (a_{ij} - 1)$ (order $O(n^2)$).

This allows us to decrease the computational complexity from $O(np^2)$ to $O(np + n^2)$[1].

4.4 Learning Phase Parameters

During *Learning* phase, 2.000 generations were ignored at the beginning of *Example gathering* and *Refinement* phases. The ranges of parameter values were divided into 4 fuzzy partitions and subdivided with *fineness* of 3. Within each cell of the training grid 5 generations were executed. During *Refinement*, diversity descends and mounts linearly for 800 generations each one. In order to avoid the effects of the modeling in the strategy comparison, 15 preliminary runs were made for each instance and only one cachedDiv has been chosen for each instance. The chosen cachedDiv was the one that presented the smaller deviation of observed diversity from commanded diversity during test runs.

4.5 Diversity strategies

During control phase, six strategies (cf. Section 3.3) were compared, three dynamic (MX, CD, ZZ) and 3 static ones (FX).

Intermediate diversity levels of MX were 0.7, 0.6, 0.5, 0.4, 0.3 and 0.2 (in the range of reachable diversity, $[0, 1]$). Every level was maintained during 300 generations. Oscillation was $\pm 10\%$, and forgetting period was of 200 generations.

For CD, $g = 100$, and diversity increasing and decreasing were of 0.003 and 0.001, respectively.

The three static strategies were fixed in values of 0.4, 0.5 and 0.6. They were named FX.4, FX.5 and FX.6, respectively.

1. The values for n of instances of QAPLIB ranges from 12 to 256 and p is 100, thus in practice time savings are effective.

5. Results and Discussion

Table 1 presents the mean percentual error with regard to the best known cost and the standard deviation (in parenthesis) for the different strategies and instances (i.e., a value $0.07(0.05)$ means that the mean result was 0.07% above the best known solution for this instance, and the standard deviation was 0.05% of this value). The column on the right shows the best known solution published in QAPLIB (in June 2008). Average number of optimal solutions (over 15 runs) are shown at the bottom of the table. The number of problems in which each method has outperformed the others is also shown at the bottom of the table. Comparisons among methods were done using a Student T test with a significance level of 5%.

Strategies effectiveness can be appreciated by looking at the average number of times that the algorithm has reached the optimum. Using this criteria, the best strategy is MX, followed by FX.5, CD, FX.4, ZZ and FX.6. However, if we are interested in strategies that could be applied to different situations, we will be more interested in the number of problems in which the strategy obtained better results than the others. CD outperformed other methods 69 times, followed by MX, FX.4, FX.6, FX.5 and ZZ. CD and MX seems to be the most generic strategies, and could work properly in most problems. ZZ is the only method able to always solve *els19*. However, it performs poorly over the rest of the instances. This is a good example of the no free lunch principle: a tradeoff exists between good specialized algorithms and not-so-good and generic ones.

An obvious question is whether variation in diversity is necessary. Let us consider the least defeated fixed setting, FX.4, and compare it against CD and MX. FX.4 outperforms other algorithms in 43 instances, while CD and MX does it 69 and 64 times, respectively. On the other hand, CD and MX are outperformed in 9 and 14 occasions, while FX.4 is defeated in 40. Results suggest that parameter control, when done in a careful manner, has clear advantages against a fixed setting, specially when the correct setting is unknown.

In order to understand how CD, MX and ZZ strategies work, we will develop the study of three representative instances. Here we are interested in the accuracy of the model, i.e., if observed diversity follows commanded diversity, and how the modifications in commanded diversity help to improve fitness.

Consider CD in *tai64c* (Figure 11). Observed diversity stays in the range $\pm 10\%$ in relation to commanded diversity. Diversity was well modeled, at least for the range $[0.6, 0.9]$, in which it was used. At the beginning of the execution, commanded diversity was decreased to concentrate the diverse initial population. Around generation 1800, the insufficient diversity produced the stagnation of the population, identified by a flat line of best fitness. This situation was early detected by the controller and the diversity was raised until a somewhat stable level close to 0.8, reaching a tradeoff between exploration and exploitation.

Now lets consider MX solving *ste36b* (Figure 12). Observed diversity remains closer to commanded diversity than CD. This is maybe due to the oscillation of commanded diversity: the fact that many levels of diversity are commanded in a short time decreases the probability that a single bad-modeled diversity is commanded, causing the good behavior of observed diversity. The multiple cycles of exploration-to-exploitation of MX are useful to escape from local optima. Note that the population started to converge around generation 4.700, from where probably it would not be able to escape. Raising diversity allows the population to "forget" that optimum to later find a better one.

Table 1: Mean percentual error and standard deviation regarding to the best known results

Instance	MX	CD	ZZ	FX.4	FX.5	FX.6	Best known
bur26a	0.07(0.05)	0.04(0.05)	0.09(0.03)	0.12(0.06)	0.06(0.04)	0.08(0.03)	5426670
bur26b	0.05(0.06)	0.07(0.08)	0.05(0.06)	0.21(0.11)	0.07(0.08)	0.03(0.04)	3817852
bur26g	0(0)	0(0.01)	0.01(0)	0.02(0.1)	0(0)	0.01(0)	10117172
bur26h	0(0)	0.04(0.15)	0(0)	0.17(0.28)	0(0)	0(0)	7098658
chr12a	0(0)	0(0)	0(0)	1.01(2.13)	0.27(1.03)	1.52(2.6)	9552
chr18b	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	1534
chr20c	5.92(4.78)	10.05(6.83)	2.26(2.93)	14.05(8.47)	12.18(11.73)	11.16(5.67)	14142
chr25a	12.8(3.81)	9.58(5.08)	24.81(4.97)	9.95(6.26)	10.22(5.29)	13.96(5.69)	3796
els19	0.24(0.37)	1.1(2.48)	0(0)	1.44(4.52)	3.22(6.31)	0.44(0.5)	17212548
esc32a	2.3(0.76)	5.38(2.3)	12.3(3.07)	1.53(1.53)	3.07(0.76)	6.15(1.53)	130
esc32b	3.57(4.76)	8.92(4.16)	13.09(1.78)	4.76(4.76)	2.97(4.16)	4.16(4.76)	168
esc64a	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	116
had12	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	1652
had20	0(0)	0(0)	0(0)	0.27(0.26)	0.23(0.26)	0.21(0.18)	6922
kra30a	1.93(0.6)	2.01(0.75)	3.26(0.62)	1.86(0.85)	1.58(1.03)	2.75(0.46)	88900
lipa20a	0.35(0.57)	0.19(0.32)	0.32(0.57)	0.13(0.32)	0.27(0.54)	0.27(0.48)	3683
lipa40b	5.79(8.47)	6.94(8.8)	17.24(3.98)	4.68(8.04)	2.37(5.88)	7.59(8.88)	476581
lipa60a	1.12(0.05)	0.97(0.04)	1.33(0.03)	1.21(0.04)	1.25(0.02)	1.3(0.02)	107218
lipa60b	19.1(5.29)	19.25(0.24)	21.77(0.2)	20.8(0.19)	21.21(0.23)	21.59(0.16)	2520135
nug15	0(0)	0(0)	0(0)	0.08(0)	0(0)	0(0)	1150
nug20	0.11(0.11)	0.03(0.07)	0.15(0.23)	0.07(0.03)	0.03(0.03)	0.07(0.07)	2570
nug30	0.86(0.42)	0.52(0.34)	2.23(0.4)	0.52(0.29)	1.3(0.57)	1.91(0.35)	6124
rou20	0.56(0.21)	0.45(0.24)	0.61(0.46)	0.33(0.33)	0.41(0.26)	0.47(0.31)	725522
scr20	0.31(0.38)	0.08(0.16)	0.13(0.23)	0.18(0.31)	0.1(0.22)	0.05(0.1)	110030
sko42	1.07(0.39)	0.94(0.3)	3.34(0.38)	0.93(0.35)	1.61(0.3)	2.44(0.27)	15812
sko64	1.43(0.24)	1.14(0.36)	5.02(0.43)	2.46(0.34)	3.49(0.28)	4.22(0.3)	48498
ste36a	2.28(1.07)	1.86(1.02)	7.78(1.55)	2.33(1.17)	2.62(0.82)	3.94(0.85)	9526
ste36b	2.25(1.84)	3.08(3.31)	7.07(1.6)	3.57(3.47)	3.19(2.71)	3.05(1.77)	15852
ste36c	1.97(0.92)	1.47(1.02)	3.47(0.84)	2.19(1.24)	1.8(1.16)	2.8(1.09)	8239110
tai20a	1.01(0.4)	0.89(0.22)	1.32(0.34)	0.84(0.25)	0.88(0.37)	0.8(0.49)	703482
tai20b	0.08(0.18)	0.3(0.22)	0.17(0.21)	0.21(0.23)	0.3(0.22)	0.17(0.21)	122455319
tai40a	3.52(0.38)	2.91(0.41)	5.26(0.35)	4.08(0.36)	4.65(0.36)	5.14(0.36)	3139370
tai40b	1.6(1.26)	2.11(1.93)	2.7(1.28)	2.04(1.81)	2.03(1.49)	1.83(1.24)	637250948
tai60a	5.68(0.46)	3.64(0.26)	7.17(0.31)	6.3(0.34)	6.72(0.16)	7.13(0.24)	7205962
tai60b	1.54(0.82)	1.35(0.74)	3.58(0.77)	5.7(3.4)	1.66(1.93)	1.91(0.83)	608215054
tai64c	0.1(0.11)	0.03(0.03)	0.1(0.11)	0.3(0.2)	0.28(0.19)	0.16(0.14)	1855928
tho40	1.55(0.56)	1.45(0.58)	4.75(0.66)	1.39(0.33)	2.68(0.37)	3.71(0.44)	240516
wil50	0.43(0.11)	0.36(0.18)	1.86(0.16)	0.86(0.21)	0.69(0.1)	1.22(0.11)	48816
avg.opt	4.82	4.6	4.29	4.5	4.61	3.87	
outp MX	–	8	2	2	1	1	
outp. CD	3	–	1	2	2	1	
outp. ZZ	21	21	–	18	21	16	
outp. FX.4	12	11	6	–	6	5	
outp. FX.5	12	12	4	9	–	14	
outp. FX.6	16	17	4	12	1	–	

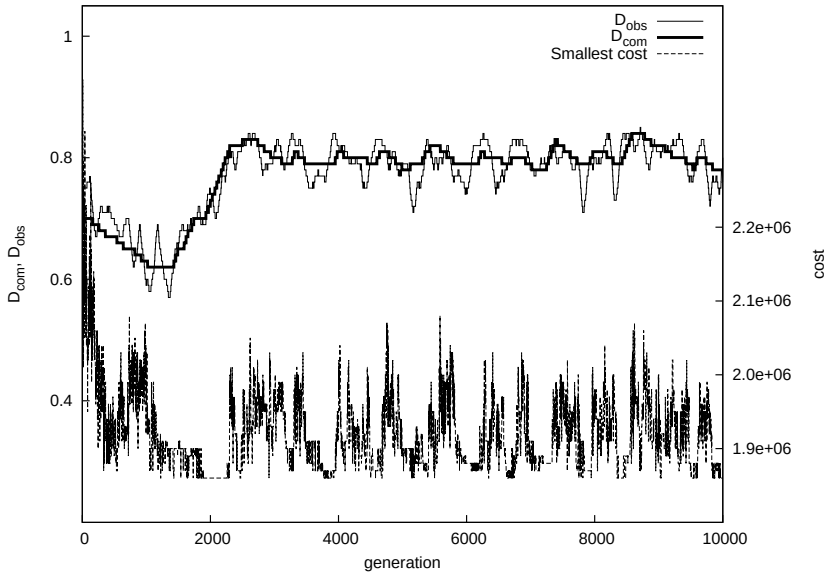

Figure 11: Plot of commanded diversity (D_{com}), running mean (100) of observed diversity (D_{obs}) and best cost (below) for *tai64c*, using CD

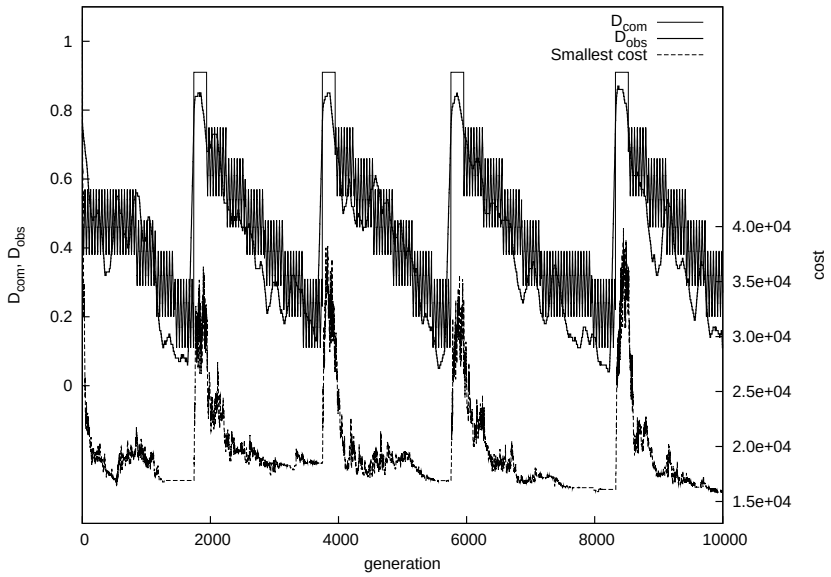

Figure 12: Plot of commanded diversity (D_{com}), running mean (100) of observed diversity (D_{obs}) and best cost (below) for *ste36b*, using MX

Figure 13 shows ZZ solving *els19*. In this case the model tends to produce lower diversities than commanded, but this seems not to be important, since ZZ has solved this instance in every run.

Best improvements were produced in generations 650, 1930 and 8267 (where the oscillation was reset). In all cases improvements are found when diversity is close to 0.7, thus it seems to simply be matter of keeping the right diversity. Actually, executions with fixed diversity in 0.7 (not shown here) produced almost the same results.

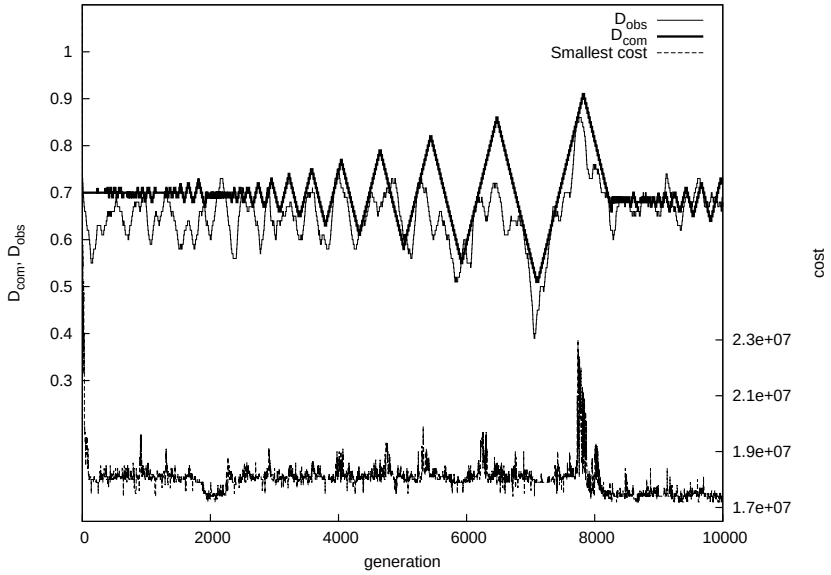

Figure 13: Plot of commanded diversity (D_{com}), running mean (100) of observed diversity (D_{obs}) and best cost (below) for *els19*, using ZZ

Perhaps the most intriguing question is why neither CD nor MX did reach this value. Figure 14 shows plots of *els19* being solved by CD and MX.

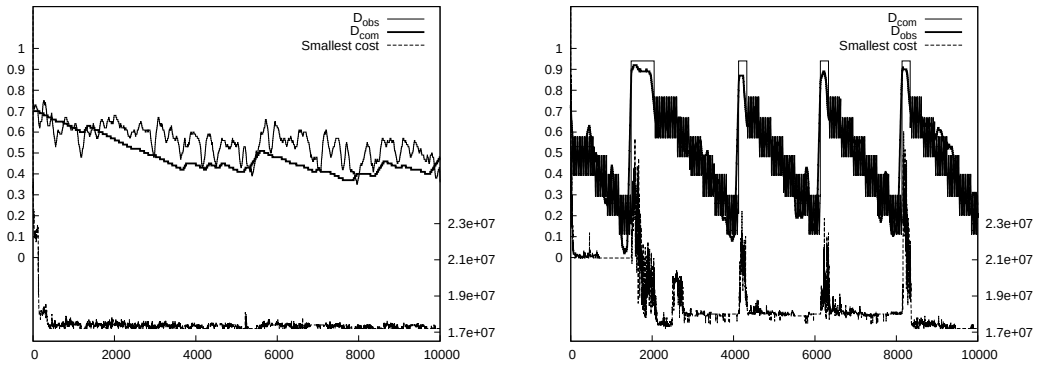

Figure 14: Plot of commanded diversity (D_{com}), running mean (100) of observed diversity (D_{obs}) and best cost (below) for *els19*, using CD (left) and MX (right)

CD systematically decreases diversity below 0.7, causing stagnation in a local optima. This is probably due to the selection scheme used: *els19* has big values of cost ($\sim 10^7$), so roulette wheel is not always able to keep the best value from one generation to the next, since a small difference in fitness values produces virtually no difference in the probability of being chosen. This produces the illusion that the population is spread, reason why CD decreases its diversity value. On the other hand, the problem with MX is that most of the time diversity is either above or below 0.7, and the population does not have the time to find the global optimum. Note that when diversity remains longer at this value, lower cost are produced (generations 2300 and 8400 in the figure, when using MX).

5.1 Learning time

The main drawback of this approach is the time consecrated to the Learning phase, which is victim of the "curse of dimensionality", given the exponential relationship of the number of parameters controlled and the number of examples to be collected. The number of generations required by the Learning phase is given by the following expression.

$$G_i + e \cdot \prod_{j=1}^{d} (p_j \cdot f) + G_r \tag{2}$$

where G_i and G_r are the number of generations during the Ignoring and Refinement subphases, p_j is the number of fuzzy partitions of the parameter j, f is the fineness, and e is the number of examples taken in each cell of the training grid. The table 2 shows the number and percentage of generations dedicated to each phase, using the setting described above.

Table 2: Distribution of generations by phase and subphase

Phase	Subphase	Generations	Percentage
Learning	Ignore	2.000	9%
	Example gathering	8.640	39%
	Refinement	1.600	7%
Control		10.000	45%

Following subsections discuss some extensions and modifications in order to improve the methodology, specially trying to reduce Learning time.

5.2 Learning tree

The main idea here is to quickly identify useless settings in order to prune the search space of parameters. Figure 15 presents a 2-parameter search space. The strong line represents the ideal CachedDiv that we are trying to find. The idea is to divide the domain of each parameter in a number of intervals (2 in the example), and take a sample of both diversity and quality in each area (dots in the figure). The average values give a rough approach to compare areas and decide which one should be further explored. The comparison is based on these two criteria, and only the areas that are Pareto-dominant are "opened" (B and C, in the figure). The process is repeated several times, opening sub-areas with different compromises of EEB. At the end, only the gray zones

should be strongly explored. To test this approach we have developed a method able to explore a n-dimensional search space, dividing each dimension into s intervals every time.

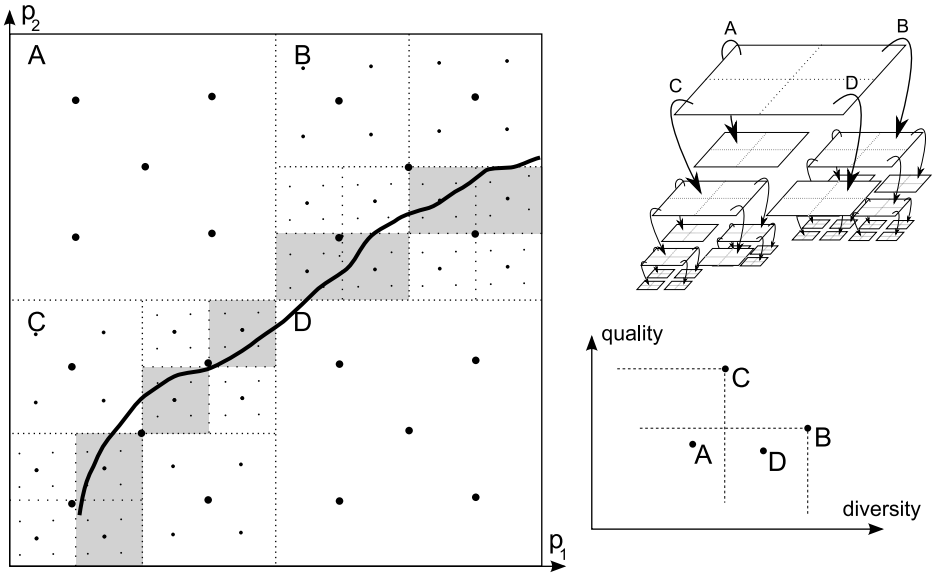

Figure 15: Expected operation of learning tree. Search space of parameters is explored according to its Pareto-dominance. In the first level zones B and C are opened because they are Pareto-dominant over A and D

The problem with this approach is the speed of he EA to find a solution: regardless of the parameter settings, diversity and quality increase quickly, so examples in a small area dominate the rest of the points, avoiding the exploration of other segments of CachedDiv. In practice, this method could be useful for exploring spaces that do not evolve (or do it slowly). This problem could be overcame by comparing several parallel executions of the algorithms, one in each opened zone, using a method like the one proposed by Yuan and Gallagher (2007).

5.3 Automatic fuzzy partition

The placement of fuzzy partitions plays a crucial role in the quality of the model. Consider the function being modeled in Figure 16 with two different partitions. The strong line in Figure 16.a is the function to be modeled, and the dashed one is the model, obtained by linear regression in each Fuzzy partition, regularly placed (shown below). Figure 16.b shows the modeling of the same function, this time with Fuzzy partitions placed in a smarter way. Note that important improvements of the model can be obtained by slightly moving partitions. Implementing a self-organizing controller has two advantages: (1) improve the accuracy of models, and (2) eliminate the parameters associated to the number of partitions to apply to each parameter.

Several constructive self-organizing methods have been proposed to define fuzzy partitions in Mamdani-type controllers (Riid and Rüstern, 2004; Piegat, 2001). We have experimented a method inspired by these previous works, making the necessary changes to use it with Takagi-Sugeno Controllers. The motivation is twofold: on one hand, an automatic setting of fuzzy partitions eliminates

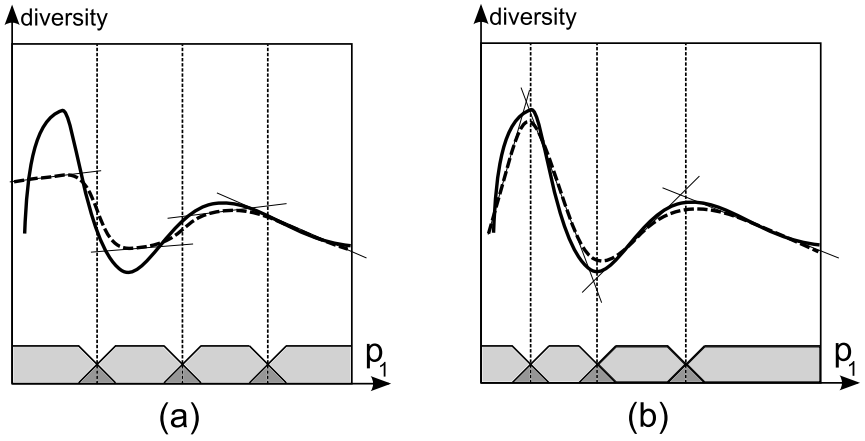

Figure 16: Effect of fuzzy partitions placement when modeling a function

the need of setting the parameter corresponding to the number of fuzzy partitions (p_j in equation 2); on the other hand, a better partitioning could reduce the number of fuzzy partitions, thus decreasing the number of generations during example gathering subphase.

The process is shown in Figure 17. Strong lines represent the function to model, dashed lines represent the model and thin lines represent the linear approximation of each fuzzy partition. At the bottom of each plot, the error of the model and the partition scheme is shown.

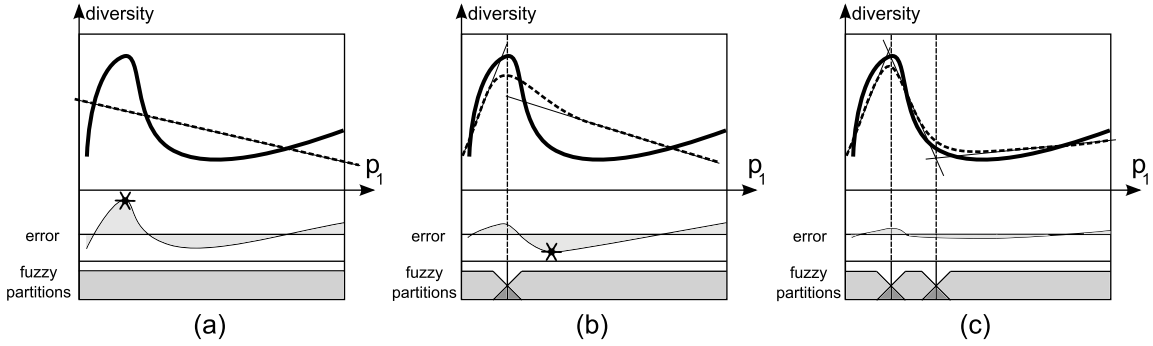

Figure 17: Self organizing fuzzy model. Example of Fuzzy partitions placement, guided by error

At the beginning (a), a single partition, that covers all the domain of parameters, models the function as a linear function. The first cut point is defined by looking at the error function: the peak point on the biggest error area is selected to be cut (marked with an asterisk). The function is modeled again (b), this time by considering the cut. A new cutting point is defined and the model is built again (c).

In order to prevent over-training of the model, the gathered data is divided into two sets, the first one, of size $\frac{2}{3}$ of the examples, is used for training, and the remaining third, to control generality. Cuts are placed in the higher error zone, considering all parameters. If a new model is less accurate than the previous one, the last cut is removed from the model and put in a tabu list, until another successful cut is found. The process goes on until the model has reached a fraction of the first model, or until there are not points left for placing new cuts.

Compared to previous existing methods, we have modified a small but crucial detail. Given that the consequents of rules (outputs) in Mamdani-type controllers represent fixed levels of the output variable, maximum error points correspond typically to the center of a fuzzy rule. On the contrary, when using Takagi-Sugeno FLCs, maximum error points correspond to the *edges* of fuzzy partitions, and no to their centers.

This method eliminates the need to define a number of fuzzy partitions and the manual bounding of the domain of parameters (note that in Figure 8 the domain of the mutation exch-mut operator is restricted to $[0, 0.6]$).

In our problem, the effect of parameters over both diversity and fitness present often a simple shape, so the accuracy has not been improved very much. However, this approach represents an interesting advance for our controller, since it allows us to get rid of some parameters, providing a more autonomous algorithm.

5.4 Confidence interval for CachedDiv

As the search goes on, individuals in the EA cover different areas of the search space of the problem. The fitness landscape may vary in different zones of the search space, thus the effect of parameters can change in different stages of the search. In order to update the model, we have tried to blend refinement subphase of learning and control phase. The idea is to constantly refine the model, in particular the values of parameters that produce the most demanded diversity levels. This could have the additional benefit of reducing the number of generations required for the initial (pure) learning phase, because a strong learning would be not needed anymore: a rough outline would be enough to start, and the real CachedDiv would be discovered on-the-fly.

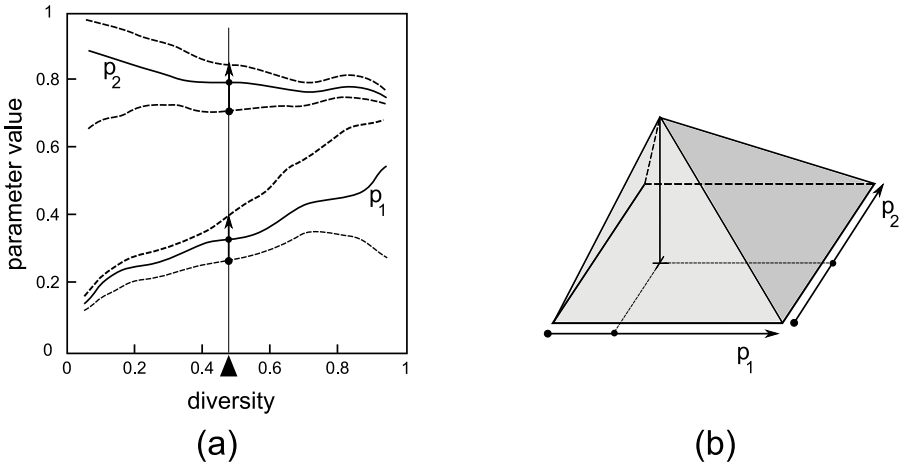

Figure 18: Confidence interval shown in cachedDiv table (a). Triangular distribution of parameter setting, based on length of confidence intervals (b)

A confidence interval is defined for each parameter and for all levels of diversity. The thickness of the interval depends on the accuracy of observed diversity with respect to commanded diversity. It is thiner when the values of parameters that produces a given diversity is well identified and wider when several values can produce this diversity. Figure 18.a shows cachedDiv with the confidence

interval. The thickness of each diversity level is obtained by looking at Diversity FLC. A threshold around a given diversity is set, for instance, to $\pm 2\%$, and assigned as lower and upper bounds of cachedDiv.

In order to explore the search space of parameters during the run, parameters instantiation is done using a triangular distribution, using the value in cachedDiv as the mode, and those corresponding to lower and upper bounds as limits. Figure 18.b shows the probability density function for two parameters, using the intervals corresponding to a given diversity (small triangle at the bottom of Figure 18.a).

This approach provides a better model of the parameter search space. However, the cost of this information was too high: the fact that parameter values were instantiated in a range makes even harder to obtain the diversity requested by the controller. This alternative could be used if the goal is to understand the effect of parameters over the EA.

6. Conclusions

Excluding evolutionary strategies, parameter control in evolutionary algorithms is restricted to experts. Actually, a general user of EA fixes the parameters either by hand or by a long series of experiments, that are often incomplete and lead to a suboptimal parameterization.

In this paper we propose a method that automatically creates an abstraction of the parameters of the EA. All parameters are handled in the same way, and their control is wrapped by a single parameter, that adjusts the balance between exploration and exploitation. The importance of this abstraction relies on a human factor: it is indeed much easier to deal with a single and intuitive parameter than with many ill-known ones. Our goal was to create a method that would work with any parameter in an abstract way, easy to implement, and that helps the user to save time in the process of becoming familiar with the parameters of the algorithm. As far as we know, no prior effort was made to create such general abstraction of parameters in EAs.

We distinguish here two ways for achieving parameter control. The most used one consists in setting up a function that computes the parameter values, according to some measures of performance. A different approach, used in this work, relies on the idea of discovering the effect of parameters over performances by building a model based on a set of examples obtained from the same algorithm.

Our method is divided in two main phases. The first one is dedicated to the understanding of how the algorithm works, and corresponds to the automation of the user's work when he tries to establish the values of parameters. The second phase corresponds to the real execution of the algorithm, based on the knowledge obtained before.

Two criteria are considered for each parameter, that are common to any EA. The first one is the quality of the solutions, which is measured as the mean fitness of the individuals. The second one is population diversity. We aimed at maximizing both criteria in order to keep a compromise between these two goals and identify the parameter values corresponding to the Pareto front. Since the shape of the functions that relate parameter values with both diversity and quality are unknown, we used fuzzy logic controllers to model these functions.

The resulting Pareto front corresponds to a set of points that represent different compromises between exploration and exploitation (EEB).

During the first phase, we analyze the main problems that we found when gathering the examples, namely dimensionality, inertia and noise. Some mechanisms were proposed to mitigate them.

The learning process is presented as a black box to the user, who only needs to implement two simple methods to integrate the controller into his algorithm. Nevertheless, a more inquiring user may benefit from the acquired knowledge in order to understand the internal behavior of the algorithm.

During the second phase, several strategies for diversity variation were compared. We may distinguish between two extreme behaviors, from strategies that look for an ideal level of EEB, to those which promote a continuous oscillation between exploration and exploitation. An experimental comparison of these strategies has been presented. Two strategies MX and CD, stood out in the comparison. MX is based on oscillation, and CD on maintaining a stable level of EEB. Although both obtained similar results, CD a appeared more interesting due to its simplicity.

Our approach has been tested over 38 different instances of QAP, using an EA with 3 operators, whose application rates were controlled. We also have proposed a new diversity measure for permutation encodings, used in QAP.

The main drawback of our method is the amount of time required to gather the examples to build the model. In our experiments, 55% of the generations were dedicated to this task. However, it must be noted that this method replaces the work of the user when he is trying to obtain an appropriate parameter setting. Anyway, several extensions of the method, aimed at reducing this time, were outlined and could guide future work on this subject.

Other future directions may include a method to identify parameters that does not have a preponderant effect over the EA performance, in order to eliminate them from the controlling scheme and lighten the learning phase. Parallelization of the computation, or store results from one run to the next could be also considered to decrease the learning time.

Future work could also include an investigation of more strategies, trying to identify the simpler and most effective ones. We also want to search for other problems using different operators and parameters in order to validate the generality of our method. One may explore other s parameters such as local search operators, discrete ones, selection pressure, or population size. Of course, since our approach is based on the measurement of diversity and quality, parameters that affect fitness function could not be controlled by our method.

References

H. G. Beyer and H. P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing, Springer*, 1(1):3–52, 2002.

J. J. Buckley. Sugeno type controllers are universal controllers. *Fuzzy Sets and Systems, Elsevier*, 53(3):299–303, 1993.

R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *Journal of Global Optimization, Springer*, 10:391–403, 1997. URL `http://www.seas.upenn.edu/qaplib/`.

P. J. Costa-Branco and J. A. Dente. Noise effects in fuzzy modelling systems. *Computational Intelligence and Applications, World Scientific and Engineering Society Press*, pages 103–108, 1999.

A. E. Eiben, E. Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Proceedings of Parallel Problem Solving from Nature (PPSN)*, volume 3242 of *LNCS*, pages 41–50. Springer, 2004.

A. E. Eiben, M. Horvath, W. Kowalczyk, , and M. C. Schut. Reinforcement learning for online control of evolutionary algorithms. In Brueckner, Hassas, Jelasity, and Yamins, editors, *Proceedings of the 4th International Workshop on Engineering Self-Organizing Applications (ESOA)*, volume 4335 of *LNAI*, pages 151–160. Springer, 2006.

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. (2007), 2007.

G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 258–265. Morgan Kaufmann, 1999.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. ISBN 0262082136.

C. Igel and M. Kreutz. Operator adaptation in structure optimization of neural networks. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, page 1094. Morgan Kaufmann, July 2001. ISBN 1-55860-774-9.

E. Kee, S. Airey, and W. Cyre. An adaptive genetic algorithm. In H. Beyer, E. Cantu-Paz, D. Goldberg, Parmee, L Spector, and D. Whitley, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 391–397. Morgan Kaufmann, 2001.

T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica, Blackwell Publishing*, 25:53–76, 1957.

A. Kulkarni. *Fuzzy Logic Fundamentals*, chapter 3, pages 61–103. Prentice Hall PTR, 2001.

J. Lis. Parallel genetic algorithm with dynamic control parameter. In *Proceedings of IEEE International Conference on Evolutionary Computation (CEC), IEEE Press*, pages 324–329, 1996.

F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

F. G. Lobo and D. E. Goldberg. Decision making in a hybrid genetic algorithm. In *Proc. of IEEE Intl. Conference on Evolutionary Computation (CEC), IEEE Press*, pages 122–125, 1997.

J. Maturana and F. Saubion. On the design of adaptive control strategies for evolutionary algorithms. In *Proceedings of 8th International Conference on Artificial Evolution. LNCS 4926, Springer*, 2007a.

J. Maturana and F. Saubion. Towards a generic control strategy for EAs: an adaptive fuzzy-learning approach. In *IEEE Congress on Evolutionary Computation (CEC), IEEE Press*, 2007b.

Z. Michalewicz. *Genetics Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the TSP. In *Proceedings of the 2nd Int.Conf.on Genetic Algorithms and their applications, Lawrence Erlbaum Associates*, USA, 1987. ISBN 0-8058-0158-8.

V. Pareto. *Cours d'économie politique*. Université de Lusanne, 1896.

A. Piegat. *Fuzzy Modeling and Control*. Springer-Verlag, 2001.

I. Rechenberg. *Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, GER, 1973.

A. Riid and E. Rüstern. Heuro-fuzzy extraction of interpretable fuzzy rules from data. *Proc.IEEE International Conference on Systems, Man And Cybernetics*, 3:2266–2271, 2004.

T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:116–132, 1985.

D. Thierens. *Parameter Setting in Evolutionary Algorithms*, chapter Adaptive Strategies for Operator Allocation, pages 77–90. Volume 54 of Lobo et al. (2007), 2007.

S. Tsutsui, Y. Fujimoto, and A. Ghosh. Forking gas: Gas with search space division schemes. *Evolutionary Computation, MIT Press*, 5(1):61–80, 1997.

L. X. Wang and J. M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1414–1427, 1992.

J. Whitacre, T. Pham, and R. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1345–1352. ACM, 2006.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, IEEE Press*, 1:67–82, 1997.

Y. Y. Wong, K. H. Lee, K. S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing, Springer*, 7(8):506–515, 2003.

B. Yuan and M. Gallagher. *Parameter Setting in Evolutionary Algorithms*, chapter Combining Meta-EAs and racing for Difficult, pages 121–142. Volume 54 of Lobo et al. (2007), 2007.