Inductive Learning of Mutation Step-size in Evolutionary Parameter Optimization

Michèle Sebag
LMS – URA CNRS 317
Ecole Polytechnique, 91128 Palaiseau Cedex, France
Marc Schoenauer
CMAP – URA CNRS 756
Ecole Polytechnique, 91128 Palaiseau Cedex, France
Caroline Ravisé
LMS – URA CNRS 317
Ecole Polytechnique, 91128 Palaiseau Cedex, France

Abstract. The problem of setting the mutation step-size for real-coded evolutionary algorithms has received different answers: exogenous rules like the 1/5 rule, or endogenous factor like the self-adaptation of the step-size in the Gaussian mutation of modern Evolution Strategies. On the other hand, in the bitstring framework, the control of both crossover and mutation by means of Inductive Leaning has proven beneficial to evolution, mostly by recognizing – and forbidding – past errors (i.e. crossover or mutations leading to offspring that will not survive next selection step). This Inductive Learning-based control is transposed to the control of mutation step-size in evolutionary parameter optimization, and the resulting algorithm is experimentally compared to the self-adaptive step-size of Evolution Strategies.

1 Introduction

In the framework of stochastic optimization methods, Evolutionary Algorithms (EAs) have proven successful on many difficult problems in recent history. The general outlook of an EA is a crude mimic of the Darwinian evolution of a *population* (i.e. a P-tuple) of *individuals* (i.e. points of the search space). After a – generally – random initialization, the population undergoes a series of *generations*, whereas a generation can be described as:

- `selection` `(based on fitness values)`
  `of some individuals for reproduction;`
- `reproduction` `through evolution operators`
  `(e.g. crossover, mutation) to generate offspring`
- `replacement` `(based on fitness values)`
  `of some parents by some offspring`

The main branches of EAs – Genetic Algorithms (GAs) [11, 10], Evolution Strategies (ES) [24, 27] and Evolutionary Programming (EP) [9, 6] – can be distinguished by the way they perform selection and replacements: GAs use proportional selection and global replacements of parents by offspring; ES and EP do not use any selection, ES uses deterministic replacement and EP a stochastic tournament among parents and offspring. Moreover, GAs favor crossover as main evolution operator, whereas ES and EP use mostly mutation.

Another major historical difference lies in the search spaces these methods consider. Roughly, canonical GAs solve the problem at hand on a binary space by employing standard binary evolution operators. Several works have shown how much the feasibility of optimization depends on the particular binary mapping [3, 14]. EP and ES have developed algorithms suited to the "natural" formulation of the problem, i.e. they directly work in the real-valued search space in case of continuous parameter optimization.

However, as many real-world problems involve real-valued variables, real-parameter optimization has lately encountered an increasing interest among all branches of Evolutionary Computation.

This paper is concerned with mutation operators on $\mathbb{R}^n$. In practice, the mutation of a real-valued vector is often achieved through the addition of Gaussian noise. The main question remains to determine the desired variance of this Gaussian law, that is, the mutation step. This mutation step should ideally depend on the individual at hand (and on its proximity to the optima) and on the current state of evolution. Several heuristics have been proposed in the literature to control the mutation step, which are either deterministic [24], or based on evolution itself [27, 7]. This paper investigates another means based on Machine Learning (ML), for the control of the mutation step; it extends previous works [29, 23, 22], devoted to the control of crossover and mutation operators in binary spaces. The idea consists in observing evolution and learning online from these observations; from examples of trials and errors, learning extracts rules discriminating successful trials from errors. These rules can notably be used for "avoiding evolution to repeat its past errors" and filtering disruptive evolution events in the following generations.

This paper is organized as follows. The next section briefly describes the current trends for real-valued optimization in EAs, and summarizes the state of the art in mutation step-size control. Section 3 details how ML can be used to support the control of evolution and focuses on two questions: what can be learnt from observing evolution, and how to use this knowledge to provide evolution with relevant advises in the next generations. An ML-controlled scheme of mutation is applied to several test-problems of the literature; these results are compared to those of other evolutionary algorithms in section 4. The last section sketches several avenues for further research.

2 State of the art

This section describes the current trends in evolutionary optimization with respect to two search spaces: the first one, referred to as *phenotype space* is that of the fitness function; in that space, individuals are evaluated and selected. The second one, referred to as *genotype space* is the space the evolution operators apply to; in that space, individuals are represented and evolved.

Let us first discuss GA, EP, and ES, before detailing the control of mutation.

2.1 GAs and real coding

Genetic Algorithms [11, 10] have been designed to operate binary strings. Canonical GAs therefore used to tackle parameter optimization problems (the phenotype space is a subset of $\mathbb{R}^n$) through a binary representation of individuals (the genotype space is $\{0,1\}^k$). This is done by discretizing the domain of every real-valued parameters into 2^k intervals; a real-value is thereafter discretized, and encoded into a bitstring of length k using either the standard digit or the Gray coding [3]. However, beside the loss of information induced by the discretization step, Genetic Algorithms using binary coding and the corresponding standard binary operators violate some basic principles related to the so-called schemata analysis [20]. Most people working in the area of parameter optimization using Genetic Algorithms now rely on *real-encoded* GAs, where both the genotype space and the phenotype space are subsets of $\mathbb{R}^n$. This implies the definition of specific real-valued crossover and mutation operators.

The most popular real-valued crossover operators are based on a linear combination of the parents, ranking from the "Guaranteed average recombination" of Wright [31] to the general BLX-α recombination of Eshelman & Schaffer [5] through the flat recombination of Radcliffe [21] and Michalewicz's arithmetical recombination [12, 16]. A question that remains unresolved is to decide to what extent such operators actually are crossover operators (i.e. obey some equivalent of the bitstring schema theorem of Holland [11]), or can be viewed as yet another type of mutation operator (see [5] and [13] for more detailed discussions on this hot topic).

The most popular real-valued mutation operator is inspired from ES and EP, and is detailed below.

2.2 EP and ES

Since their inception, Evolution Strategies [24, 27] have been designed to handle real-valued parameters, and do work directly in the space of real-valued vectors. Similarly, Evolutionary Programming [9, 6], whose general approach is to directly evolve the *phenotypes* themselves, uses real-vector representations for real-valued parameters. Further, the original versions of both these algorithms used mutation as the only evolution operator.

Both methods use *Gaussian mutation* in which a random variable with normal (Gaussian) distribution and zero mean is added to the current value of each coordinate of the vector of real-valued parameters:

$$x \in \mathbb{R} \longrightarrow x + N(0, \sigma), \tag{1}$$

where σ is the standard deviation of the normally distributed $0-$mean random variable N. Parameter σ is referred to as the *mutation step size*: The larger σ, the more likely the occurence of large mutations. The expected value for the amplitude of mutation can therefore be controlled by adjusting σ. Eventually, a specific mutation step size σ_i is associated to every coordinate x_i of the individual at hand.

Note that recent trends in ES now incorporate some form of recombination (cross-over), including the above mentioned linear recombination [2]. Thus, apart from the selection/replacement scheme, the main difference remaining between GAs on the one hand, and ES and EP on the other hand, is the systematic use of mutation of the latter methods, compared to the historically parsimonious GA way of mutating.

2.3 Tuning the mutation step-size

Since its inception in the early days of Evolution Strategies [24, 27], many different methods have been proposed to control Gaussian mutation and tune the standard deviation σ.

An early approach to exogenous adaptation of σ (used at that time for one-parameter problems) is the well-known 1/5 rule of Rechenberg [24]. In this scheme, σ is modified according to the results obtained by mutation in the last few generations: if more than one-fifth of the mutations have been successful (i.e. led to an offspring more fit than the parent), increase σ, otherwise decrease σ. The schedules for increasing and decreasing σ are geometrical schedules; the factors suggested by Schwefel [27] are 1.1 and 0.9.

In the early real-valued EP, the standard deviation of mutation was determined on the basis of the fitness of the individual at hand, in such way that most fit individuals undergo small Gaussian mutations, i.e. with small standard deviations [6].

Nevertheless, both ES and EP now use the *self-adaptive mutation* [27]: The description of an individual $\mathbf{x}$ includes both the n *object variables* (its coordinates $(x_1, \ldots, x_n)$) and the corresponding standard deviations $(\sigma_1, \ldots, \sigma_n)$[1]. When mutating $\mathbf{x}$, the step sizes σ_i first undergo mutation; the coordinates x_i are modified afterwards using the new values of the deviations σ_i as in equation 1. The standard deviations themselves can therefore get adjusted "for free" along

[1] The more general version of adaptive mutation actually uses the complete covariance matrix of normal distribution in n dimensions. But the simplified version using only the standard deviations is the most widely studied and used.

evolution [27, 2].

Early works in EP used Gaussian mutation described in equation (1) to evolve the standard deviations σ_i [7]. But there is now an almost total agreement [26], with the noticeable exception of noisy domains [1], on using a log-normal mutation for mutating the standard deviations σ_i:

$$\sigma_i \longrightarrow \sigma_i \exp(\tau'.N(0,1) + \tau.N_i(0,1)). \tag{2}$$

where τ and τ' are robust exogenous parameters for which Schwefel [27] suggests the values $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$. The log-normal mutation preserves positive values for the standard deviations, and each parameter is modified by a random geometric factor symmetric around the mean neutral value of 1.

Of evidence, the successes of evolutionary optimization make it a good candidate for adjusting the parameters of evolution itself, as done in the self-adaptation method described above. However, comparative experiments showed that, whenever an optimization problem could be tackled by other (deterministic) methods, these deterministic methods were faster than evolutionary optimization by one or several orders of magnitude.

The question then becomes: how could a (deterministic) method handle the optimization of the parameters of evolution? As far as we know, the only possibility studied so far was that of the 1/5 rule [24], developped by careful analysis of two simplifed models (the sphere and the corridor). However, its generalization to other situations is not clear [8]. Moreover, this deterministic method is limited in that it can only globally handle the vector of standard deviations (σ_i), since it simply counts the number of successes and failures (offspring more or less fit than the parents): counting does not supply enough information to increase the standard deviation σ_i for some coordinate i while decreasing it for some other coordinates.

It is suggested that Machine Learning could bring some answers for adjusting the evolution parameters, through a more thorough analysis of the successes and failures of mutation depending on the actual vectors σ_i.

3 Inductive learning-based control of evolution

The use of Machine Learning for evolution control relies on the following conjectures:

- Evolution is made of events and one can categorize these events into successes and failures.
- The history of past generations contains useful information as to the trials which should — or should not — take place in the next generations.

One may doubt the collection of past successful events to give (reliable) hints as to further successful events: any strategy could be misleading on a predetermined problem.

In opposition, there is no contest that the collection of past errors gives sound – though partial – information as to give undesirable further events: whenever the fitness landscape does not change along evolution, it is a pure waste of time to repeat trials which showed errors in the past.

A strategy of control would then be to explicitly store the history of evolution, and use this memory to avoid the repetition of past errors. Note this strategy is ensured to be safe, i.e. it incurs no chance of misleading evolution.

Building explicitly and exploiting the whole history of evolution is intractable. Machine Learning, and more precisely induction from examples [17, 18], allows one to overcome this potential limitation, through summarizing examples of successes and errors into rules. Further, these rules can be used to estimate whether any unseen trial will likely be an error or a success. Several strategies of control relying on this estimate are thereafter possible; some have proven successful in the context of canonical bitstring GAs, on several well-studied binary problems [29, 23, 22].

This paper is concerned with extending this ML-based approach of the control of evolution, from binary to real-valued search spaces, and from GAs to ESs. We first briefly describe how to learn from and about evolution, and how to use on-line the rules extracted by induction. A hybrid scheme coupling evolution and induction is described last.

3.1 Learning from evolution

In this paper, the events of evolution are defined as the evolution operators that give birth to offspring. An event is categorized as being an error or a success depending on whether the offspring reach lower or higer fitness than their parents[2]. This definition is somewhat short-sighted since actually reaching the optimum is the result of a whole "dynasty". Further research will be concerned with evaluating the success or failure of an event from its descendants, in the line of Davis [4].

This study is restricted to mutation operators for two reasons. First, previous experiments in binary search spaces unexpectedly showed that mutation control is far more efficient than crossover control, in spite of the relative rarity of mutations compared to crossovers [23, 22]. Second, mutation traditionally plays the primary role in the evolutionary framework as far as real-valued individuals are considered.

A mutation event is completely described by the parent it applies onto, represented as the real-valued vector (x_i), and the quantity added to the parent in order to get the offspring, represented as a real-valued vector (δ_i).

Examples of mutation events together with the corresponding category, are represented in Table 1.

[2] Another category of events was introduced in [22]: that of inactive events, giving rise to offspring equally as fit as parents. However, in contrast with binary search spaces, there are almost no inactive events in real-valued search spaces.

Table 1 : Induction from examples of mutation. Rule R is a generalization of examples E_3 and E_4 not contradicting examples E_1 and E_2.

	Individual				*Mutation*				*Class*
E_1	3.21	4.48	-5.56	0.26	.22	0.18	-0.35	0.63	*success*
E_2	6.41	-1.33	-0.3	0.61	.14	0.04	1.02	0.18	*success*
E_3	7.52	-2.51	1.4	0.33	-.32	-0.27	1.15	0.56	*error*
E_4	7.23	0.58	0.55	0.39	-.08	0.33	2.01	0.69	*error*
R	$(x_1 > 6.41)$	$\star$	$(x_3 > -.3)$	$\star$	$(\delta_1 < .14)$	$\star$	$(\delta_3 > 1.02)$	$\star$	*error*

From such examples, induction constructs discriminant rules, that is, conjunction of conditions of the kind *attribute* A_i *lies in* V_j, where V_j is a subset of the domain of *attribute* V_i, such that these conditions are satisfied by examples which all belong to the same category. See [17, 18, 15] for a comprehensive introduction to inductive learning from examples.

In our toy example, rule R states that for any individual such that $x_1 > 6.41$ and $x_3 > -.3$, augmenting the first component by a small or negative amount δ_1 ($\delta_1 < .14$) and augmenting the third component by a large amount ($\delta_3 > 1.02$) will be an *error*, i.e. lead to an offspring less fit than the parent.

Roughly, induction either proceeds in a top-down or in a bottom-up manner. Top-down induction repeatedly selects the attribute (here, x_i or δ_j) that is most informative regarding the category of the example, e.g. in the sense of the quantity of information [19]. It thereby builds a decision tree whose nodes are attributes, branches are conditions on the father node attribute, and leaves are categories of examples.

Bottom-up induction repeatedly considers one example Ex at a time, and determines the conjunction(s) of conditions which are satisfied by Ex and by examples belonging the same category as Ex only.

For the sake of convenience, we used a home-made learner named *DIVS*, which is detailed in [28], though any other learner could probably have been used. *DIVS* is a bottom-up algorithm the complexity of which is linear in the number of attributes and quadratic in the number of examples.

The parameters of most learners, among which *DIVS*belongs, control:

- the generality of the rules, that is the fraction of the example space they cover. The degree of generality commands both the applicability and the reliability of the rules: intuitively, general rules are not as reliable as specific rules; but they apply much more often.
- the significance and consistency of the rules, that is, respectively the number of examples they explain, and the fraction of examples in the other categories they cover. The degrees of significance and consistency altogether control the number of rules extracted from examples.

The precise tuning of these parameters depends on the further use of the rules.

3.2 Knowledge-controlled evolution

The category of a further mutation event can be estimated from the rule(s) this event satisfies[3]. This estimate can accommodate several control strategies:

- Effecting only events estimated as successful. In that perspective, general rules should be preferred. Otherwise, this control would favor mutation events "close" to past successful mutation events. And this strategy would likely break the balance between the exploration and exploitation tasks achieved by evolutionary optimization [10].
- Rejecting events estimated as errors. In that perspective, specific rules should inversely be preferred, in order to ensure that only mutation events close to past error events are rejected. When the specificity of rules increases, they are satisfied by fewer and fewer examples; other examples are thus unclassified, and considered admissible. In the limit, this strategy corresponds to the pure avoidance of past errors.

Only the second strategy of control will be investigated in this paper. The degree of specificity is adjusted so as to ensure that only one-half of the mutation events will be rejected; (other events are either estimated as *successes*, or, more frequently, satisfy no rules and are unclassified). This way, only mutations most "similar" to past unsuccessful mutations are filtered.

It should be noted that this strategy does not break the balance between exploration and exploitation. Rather, the rules delineate regions where exploration or exploitation have led to bad or null results. This allows one to bias both exploration and exploitation toward other regions.

For these first experiments, however, only the vector δ_i is stored into the description of a mutation event, because most of the evolution time is spent with a nearly homogeneous population (the description of the parent thereby gives no usable information). This entails two consequences: examples may be inconsistent, i.e. a vector δ_i could be considered successful when added to a given parent, and unsuccessful when applied to another parent. However, this has no important consequence on the learning task as *DIVS*can deal with a limited amount of inconsistency. But this also means that the category associated to a given vector δ_i might change when the population moves toward regions of better fitness.

This in turn implies that the lifetime of the rules should be limited: rules must be periodically updated in order to follow the evolution of the population and remain relevant.

3.3 Integrating evolution and induction

The scheme integrating evolution and induction can be described as follows:

[3] In case it satisfies several rules concluding to different categories, the actual category of the event is determined by a majority vote.

- Examples needed for induction are gleened from evolution. One therefore gathers a set of examples, the size of which is set by the user (by default the size P of the population). Special attention is paid to representing both categories of events in this set of examples. Typically, if only bad events are gathered, induction will learn a single rule (*all is error*), which leads to a poor estimate of the event categories.
- Every L generations, these examples are considered. If both successes and errors are represented, rules are learnt. Otherwise the rule set is set to the empty set.
- The specificity of the rules is tuned so as 50% of P randomly generated mutation events are admissible (i.e. not classified errors). *DIVS*, in contrast with most other learners, allows to meet this constraint *a posteriori*, at classification time (see [28] for details).
- During the next L generations, mutations events are filtered depending on the rule set (if it is not empty): vectors δ_i estimated *error* are rejected.

The cost of ML-control includes the building and the use of the above estimate:

- The cost of induction, which is in $N \times P^2$, where N denotes the dimension of the search space and P the size of the evolutionary population; and this costs intervenes every L generations.
- The cost of using the rules, which is equally in $N \times P^2$, and is the main factor of complexity of this approach, as every mutation requires to be classified admissible before being allowed to apply.

4 First experimental results

This section presents preliminary validation experiments of the above scheme on two well-studied benchmark functions.

4.1 The sphere problem

The first and simplest possible test case is the *sphere* problem. On the one hand, it can be thought of as the real-valued equivalent of the "Count the bits" binary problem. On the other hand, it is used in Evolution Strategies as a fundamental test problem: Theoretical results prove that ESs using the self-adaptation mechanism do achieve a log-linear progress rate [25].

The problem is to find the minimum of the quadratic function f defined on $[0, 1]^n$ by $f(\mathbf{x}) = \sum_{i=0}^{n} x_i^2$.

Experimental settings Three algorithms are compared on the sphere problem:

- Standard ES with self-adaptation mechanism based on the log-normal update of the standard deviations, as described in section 2.3

- The same algorithm (ES with self-adaptation), but with the addition of the ML-control described in section 3.3: only mutations classified admissible are actually performed.

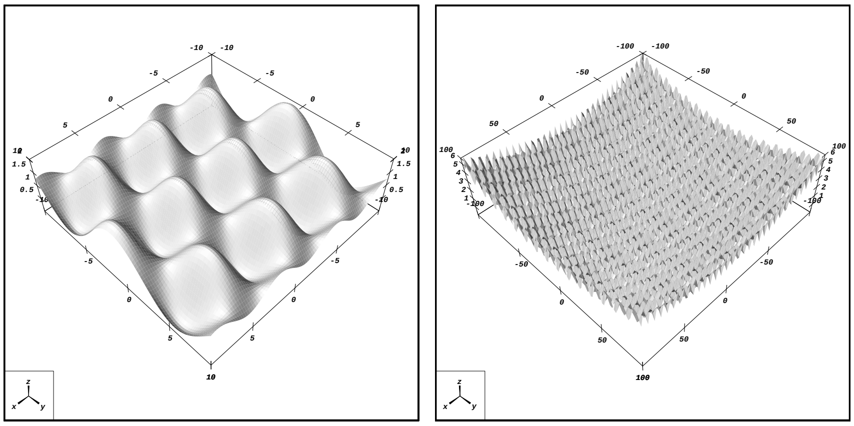

Fig. 1. *Fitness landscape for function* g. *(a) Zoom around the optimum, on sub-domain* $[-10, 10] \times [-10, 10]$. *(b) On the larger sub-domain* $[-100, 100] \times [-100, 100]$.

- A degraded ES algorithm, where the standard deviations of the mutations do not use self-adaptation: they are kept constant all along evolution.

The dimension of the search space n is set to 10. The population size is set to 10, each parent generating 6 offspring by mutation only, and the replacement step selects the best 10 individuals out of the 60 offspring (this is termed $(10, 60)-ES$ in ES terminology). The algorithms stop when 50 generations show no improvement of the overall best fitness.

The values of the exogenous parameters for the log-normal update of standard deviations are those described in section 2.3. All results are averaged over 25 independent runs.

Results The performances of the three algorithms are almost indistinguishable. However, positive conclusions can be drawn from such tight results:

- The addition of ML-control does not improve on the self-adaptation mechanism: this was predictable, as the self-adaptation mechanism is known to be optimal on the sphere problem.
- The addition of ML-control does not degrade the optimal self adaptation mechanism: it was argued in section 3 that the proposed control – avoid past errors – cannot be misleading in a fixed fitness landscape. Hence, at least on the sphere problem, it is not.

– Substituting ML-control to self-adaptation did not make significant difference: This is the first sign that indeed ML-control can be beneficial to evolution in the framework of parameter optimization, as it is able to provide similar performances than the optimal self-adaptation.

4.2 Generalized Griewank function

The second problem we consider here is to minimize the Griewank function (currently used for multi-modal optimization, both evolutionary and deterministic [30]) defined by:

$$g(\mathbf{x}) = \frac{1}{4000}\sum_{i=0}^{n} x_i^2 - \prod_{i=0}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1,$$

where variables x_i belong to $[-600, 600]$. This function has a global minimum at point $(0, \ldots, 0)$, and several local optima: see Figure 1-a for a closed view around the optimum, and Figure 1-b for a larger view. Notice that the z-scale has increased by a factor 10 between both figures, and that a view of the fitness landscape on the whole $[-600, 600]$ domain would only show a smooth bowl-like surface.

Experimental settings The aim is here to try to demonstrate that our approach can improve the overall results of evolutionary algorithms. So only the first two algorithms described in the above section 4.1 are compared (i.e. standard ES, and standard ES plus ML-control).

The dimension of the search space is set to 30, and both algorithms, use the $(100, 600) - ES$ scheme with log-normal self-adaptation of the standard deviations described in section 2.3. All results are averages over 25 independent runs.

Results Figure 2 shows plots of the average best fitness vs the number of function evaluations for the standard ES and three trials of the ML-controlled step-size, for the different frequencies of learning: the number of generations L between two successive learning phases (see section 3.3) is set respectively to 3, 5 and 10.

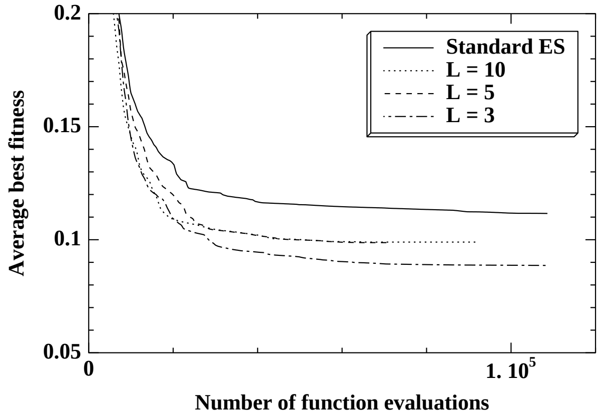

Figure 2: *Results for Griewank function.*

The first conclusion is that, on function g, the addition of ML control to standard ES seems to improve the overall performance: as can be seen on figure 2, the average values of best fitness along evolution are more than 20% lower for the case $L = 3$ than for the standard ES (and the best result obtained in the case $L = 3$ is 0.052317, compared to 0.069351 for the standard ES).

Furthermore, the frequency of learning is important: if too many generations elapse between two successive learning phases, the rules tend to become less accurate with respect to the current population, leading to degraded results. Notice however that this phenomenon did not occur on the sphere problem: due to the highly smooth and isotropic sphere surface, when mutation steps are good on that surface, they stay good for quite a long period of time. Nevertheless, further experiments will involve learning at every generation, in spite of the extra cost this will require.

However, as discussed in section 3.3, the main cost does not come from the learning phase, but from the classification phase, which is totally independent of L. But it should be noted that this cost is the main limitation to the use of the ML-control: for both problems described above, it increases the total computation time by one order of magnitude. Hence, ML-control should be used only on very expensive fitness functions, for which the eta cost of learning and classification (independent of the fitness) would be relatively small.

5 Discussion and perspectives

These preliminary results show that ML-control can adjust the mutation step size as well or better than self-adaptation, which is regarded as the optimal method for that task in evolutionary computation.

The strengths and weaknesses of ML-based control and self adaptation are quite different:

– Self-adaptation proceeds by "stochastic recommendations": it selects the

best mutation step size. Furthermore, these recommendations apply on one single individual, and are based on the fitness of this individual.
– In opposition, ML-based control proceeds by "deterministic inhibitions": it detects regions of *actual* mutation to be forbidden. These inhibitions apply on the whole population (or regions of the individual space in case the description of mutation events includes the description of the parent), and are based on the recent history of the evolution.

The advantage of ML-based control is twofold: suitable recommendations are much less numerous than suitable inhibitions, and therefore an inhibition-based control strategy is less likely misleading. Second, it offers a deterministic alternative to evolutionary self-adaptation, and will hopefully demonstrate improvement of the overall computational time of evolutionary algorithms — as soon as the role and potentialities of ML will be better understood and adjusted.

In particular, further research is concerned with learning the *direction* of mutation. A straightforward extension of the presented work consists in learning from both the parent and the direction of mutation. Rules can thereby determine which directions are to be preferred or avoided when starting from a given region. This strategy is to be compared to the general form of self-adaptive mutation in ES, based on the full covariance matrix (which in n dimension includes the n standard deviations as well as the $n(n-1)$ covariance factors). Note that, in the case where the fitness is differentiable, this strategy is nothing but building and using its gradient!

A comprehensive study will compare the performances of standard and generalized self-adaptation, the ML-based control of mutation amplitude and direction, and also the arithmetical crossover [5], which can also be viewed as a population-driven mutation operator.

Another perspective is concerned with building an rule-based model of the fitness landscape from the current population; induction is used here to discriminate better from less fit individuals in the population. This rule-based model could in turn be used to directly derive next population. The main difficulty remains to characterize the regions which are yet unexplored. From rules characterizing both promising regions and “terra incognita”, evolution could then sample the desired amount of followers and explorers in these regions to build next population.

References

1. P.J. Angeline. The effects of noise on self-adaptive evolutionary optimization. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the 5^{th} Annual Conference on Evolutionary Programming*, pages 433–439. MIT Press, 1996.
2. T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
3. R. A. Caruna and J. D. Schaffer. Representation and hidden bias : Gray vs binary coding for genetic algorithms. In *Proceedings of ICML-88, International Conference on Machine Learning*. Morgan Kaufmann, 1988.

4. L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the* 3^{rd} *International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.
5. L. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202, Los Altos, CA, 1993. Morgan Kaufmann.
6. D. B. Fogel. An analysis of evolutionary programming. In D. B. Fogel and W. Atmar, editors, *Proceedings of the* 1^{st} *Annual Conference on Evolutionary Programming*, pages 43–51. Evolutionary Programming Society, 1992.
7. D. B. Fogel, L. J. Fogel, W. Atmar, and G. B. Fogel. Hierarchic methods of evolutionary programming. In D. B. Fogel and W. Atmar, editors, *Proceedings of the* 1^{st} *Annual Conference on Evolutionary Programming*, pages 175–182, La Jolla, CA, 1992. Evolutionary Programming Society.
8. D. B. Fogel and A. Ghozeil. Using fitness distributions to design more efficient evolutionary computations. In T. Fukuda, editor, *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pages 11–19. IEEE, 1996.
9. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. New York: John Wiley, 1966.
10. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, 1989.
11. J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.
12. C. Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of 4th International Conference on Genetic Algorithms*, pages 31–36. Morgan Kaufmann, July 1991.
13. T. Jones. Crossover, macromutation and population-based search. In L. J. Eshelman, editor, *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.
14. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.
15. Y. Kodratoff. *Introduction to Machine Learning*. Pitman Publishing, London, 1988.
16. Z. Michalewicz. *Genetic Algorithms+Data Structures=Evolution Programs.* Springer Verlag, New-York, 1996. 3rd edition.
17. R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : an artificial intelligence approach*, volume 1, pages 83–134. Morgan Kaufmann, 1983.
18. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
19. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
20. N. J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–20, 1991.
21. N. J. Radcliffe. Forma analysis and random respectful recombination. In R. K. Belew and L. B. Booker, editors, *Proceedings of the* 4^{th} *International Conference on Genetic Algorithms*, pages 222–229. Morgan Kaufmann, 1991.
22. C. Ravisé and M. Sebag. An advanced evolution should not repeat its past errors. In L. Saitta, editor, *Proceedings of the* 13^{th} *International Conference on Machine Learning*, pages 400–408, 1996.

23. C. Ravisé, M. Sebag, and M. Schoenauer. An induction-based control for genetic algorithms. In J.-M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, pages 100–119. Springer-Verlag, 1996.
24. I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
25. G. Rudolph. Convergence of non-elitist strategies. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano, editors, *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pages 63–66. IEEE Press, 1994.
26. N. Saravanan, D. B. Fogel, and K. M. Nelson. A comparison of methods for self-adaptation in evolutionary algorithms. *Biosystems*, 36:157–166, 1995.
27. H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2^{nd} edition.
28. M. Sebag. Delaying the choice of bias: A disjunctive version space approach. In L. Saitta, editor, *Proceedings of the 13^{th} International Conference on Machine Learning*, pages 444–452. Morgan Kaufmann, 1996.
29. M. Sebag and M. Schoenauer. Controlling crossover through inductive learning. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, pages 209–218. Springer-Verlag, LNCS 866, 1994.
30. A. Törn and A. Zilinskas. *Global Optimization*. Springer Verlag, New-York, 1989.
31. A. Wright. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann, 1991.