The Baldwin Effect as an Optimization Strategy

Edgar Alfredo Duéñez Guzmán

July 30th, 2005

Contents

Dedicatory

This thesis is dedicated to three broad groups of people.

To my family for their support since the early stages of my life to present and future. Specially to my wife Claudia because if it were not by her I would have never done my Masters of Science. To my parents Margarita and Ernesto for the unconditional help they have always given me, and the impeccable formation I received from them. To my brothers Ernesto and Eduardo who made my life much better, each one in his very special and complementary way.

To my friends, who are as a second family to me, and who always were there when I was particularly down. To the Plano as a community, but very specially to Beta, Carlos, Eugenio, Inder, Limolín, Marte, Ponchito, Raúl, Saúl and Verónica (in strictly alphabetical order) for making my years in Guanajuato the happiest of my life.

To my professors, who build my temper and knowledge in a way I never thought possible. I will not mention anyone since I would surely omit very important people. My gratitude to them all.

Acknowledgments

I would like to thank Arturo Hernández Aguirre, my thesis advisor, for believing in me and helping me to finish my degree. Also, thanks to Mariano Rivera and Johan Van Horebeek for being a corner-stone in my late Bachelor's and early Masters' years.

Very specially, I want to thank the community CIMAT-FAMAT for being a shelter of knowledge and formation. They believed in me and allowed many activities on my behalf, always supporting and helpful.

Introduction

Biologically inspired models in computer science used for problem solving have resulted invaluable to the community. It has been almost half a century since the first attempt were made towards successful applications of these models to real world problems.

A model is by definition a simplification of reality, and it is usually the case that it can end in over-simplification of observed phenomenon. In evolutionary computation this might be the case since, from the point of view of biology, neo-Darwinism is a more complex model than any current evolutionary algorithm. This is also the case in many biologically inspired models as artificial neural networks, machine learning, automata theory, and more.

Making more and more complex models seems to be a trend of changing strength. While some researchers like more sophisticated methods for problem solving, others suggest that we should be trying to discover the *inners* of the current algorithms in order to set them on more formal foundations.

The main aim of this thesis is to present a biologically inspired, and to some extent, biologically accurate new trend in evolutionary computation by expressely trying to emulate the observed behavior known as the *Baldwin Effect*.

A number of researchers have observed (in both, evolutionary computation and evolutionary biology) a synergy between learning and evolution to a certain extent. This synergy is commonly (and mistakenly) known as the Baldwin Effect. While it is true that the Baldwin Effect explains this observed synergy, it is equally interested with the costs of learning over instinct. Concerning learning and instinct as a peculiar duality, the Baldwin Effect can be thought as the synergy, costs and trade-offs occurring between them.

Some experiments have been made by a handful of researchers, acquainted to some degree with both biology and computation, to study the Baldwin Effect in its complete form. The results were promising and inspired the author in further studying this phenomenon.

This thesis is organized as follows:

In the first Chapter we give a brief introduction to Optimization without wanting to make it the central point. The key terms are explored and an introduction to local search and constrained optimization will be given. These concepts will be used throughout the thesis, and it is recommended that the reader at least flips though them to be sure to understand the notation adopted and get used with names of

already known terms.

The second Chapter is devoted to evolutionary algorithms. There, we develop the basic definitions and algorithms. There is no attention given to results concerning proofs of convergence rate or underlying mechanisms for the algorithms, instead we try to develop the reader's intuition on the required steps to create and understand an evolutionary algorithm. Some of the main branches of this field are inspected, and a number of variants are discussed. The notions of evolutionary strategies and differential evolution will be the key for the presented experiments, and should be given special consideration.

In the next Chapter, we discuss about the Baldwin Effect. We are concentrated on a detailed explanation of the concepts and trends in this matter. We present the work of several other researchers in order to support our remarks, and give special attention to the Baldwin Effect as a whole. After developing the Baldwinian and Lamarckism concepts, we continue with a Section devoted to Baldwin Effect in computer science. There we present the more traditional works in this field, and give explanations of the observed behaviors.

The last Chapter is then filled with the central portion of this thesis. We present the term *Baldwinian Optimization*, which to the extent of our knowledge has never been used before. There we express the viability of using Baldwinian mechanisms to solve difficult constrained optimization problems, and also give the key ideas on how to adapt a Baldwinian version of virtually any population based algorithm. We also present a comparison between Baldwinian and non-Baldwinian versions of the same algorithm, and close with a small conclusion on the results obtained.

The conclusions on the work presented follow these Chapters. There we argue about the possibilities of the Baldwinian optimization as a research resource. We briefly argue that biologically inspired algorithms are more easily understood and adapted on the long run than other, more obscure, ones.

Chapter 1

Optimization

The body of mathematical results and numerical methods for finding and identifying the best candidate from a collection of alternatives without having to explicitly enumerate all possible alternatives is called *Optimization*. With the advent of the information era, the computational power have made the optimization task easier, but at the same time have brought a new range of questions concerning the efficiency and correctness of the algorithms used in optimization.

In this Chapter we provide the basis for global and constrained optimization. The aims of this Chapter are to develop the required definitions and to present a range of general-purpose techniques to attack an optimization problem.

1.1 Basic Concepts

The general optimization problem can be stated as follows. Given the pair $(S; f)$, where S is an arbitrary search space, and $f : S \to \mathbb{R}$ is a real-valued function to optimize. With the *optimum* of the problem, we mean either the maximum of the function or the minimum.

For purposes of this thesis, the optimization problem will always be regarded as a maximization problem. Observe that every minimization problem can be transformed into a maximization one by simply taking the problem as $(S; -f)$.

The value x^* is called the *optimum* (maximum) of the optimization problem $(S; f)$ if and only if it satisfies $f(x^*) \geq f(x)$ for every $x \in S$. When if along with the search space we have a *neighboring structure* $N : S \to 2^S$ defined on it, we can define the notion of *local optimum* as every value x^*_{local} satisfying $f(x^*_{local}) \geq f(x)$ for every $x \in N(x^*_{local})$. By notation we will define $X^* = \{x | x \text{ is an optimal solution of } (S; f)\}$, and $X^*_{local} = \{x | x \text{ is a local optimal solution of } (S, N; f)\}$.

Observe that the definition of a local optimum is dependent on the neighboring structure associated to the search space. With the appropriate neighboring structure, we can avoid local optimum solutions that are not global ones. We can also note that $X^* \subset X^*_{local}$ regardless of the neighboring structure N.

In general, we will have that $S \subset \mathbb{R}^m$, for continuous optimization, and $S \subset \mathbb{N}^m$,

for discrete optimization. We will call the elements $x \in S$ *solutions* of the optimization problem, as they represent the possible solution values of an optimization problem. Similarly, we will call $f(x)$, for $x \in S$, the *values* of a solution. By notation, $f(x^*)$ will be called the *optimum value* of the optimization problem.

In general, we can only tell if we are at a local optimum or not, since the notion of local optimum is based on a neighboring structure that is potentially very small compared to the size of the search space S. In order to be sure that we are in the global optimum, we have to enumerate all possible solutions and check if all of them are not greater than our proposed solution.

We also require a little more from the neighboring structure, as not every structure is useful. Given an optimization problem $(S, N; f)$, the neighboring structure is said to be *consistent* if for every pair of solutions $x, y \in S$, there exists a sequence (not necessarily finite) $\{z_i\}_{i\in\mathbb{Z}}$, such that $x = \lim_{i\to-\infty} z_i$ and $y = \lim_{i\to\infty} z_i$, and $z_{i+1} \in N(z_i)$ for every $i \in \mathbb{Z}$. If the sequence is finite, N is said to be *finitely consistent*.

This definition defines whether a neighboring structure can lead from one point in the search space to every other passing only through the neighbors (and the neighbors of the neighbors) of the points to be united. Observe that if S is finite, then every N is finitely consistent.

Let us now define a relation for neighboring. Given the relation $\sim\subset S \times S$, such that $x \sim y$ if and only if $x \in N(y)$, we can define certain desirable properties of the neighboring structure N. We will say that N is *coherent*, if and only if $\sim$ is *reflexive* (i.e. $x \sim x$) and *symmetric* (i.e. $x \sim y \Leftrightarrow y \sim x$).

The notion of consistency is used by many stochastic local search algorithms to assert global optimality, while the notion of coherency is mainly used for convenience.

There is also another definition that will prove useful in our study. We will say that the function f is *unimodal* in $T \subset S$ if and only if X^*_{local} of the reduced problem $(T, N|_T, f)$ has cardinality 1 (in other words, if there is only one local optimum in T) If the function f is not unimodal in T, then it is said to be *multi-modal*.

1.2 Local search

The first type of algorithms we might find in optimization history are the *local search* algorithms. This early attempt to solve optimization problems can be regarded as a function

$$a : S \to 2^S \text{ where } a(x) \in N(x) \text{ for each } x \in S \tag{1.1}$$

The algorithm can be either deterministic (i.e. a function as proposed above) or stochastic in which case we can generalize the above definition to be

$$a : S \times [0,1] \to 2^S \text{ where } a(x,r) \in N(x) \text{ for each } x \in S, r \in [0,1] \tag{1.2}$$

where the number r is considered to be the random portion of the algorithm.

The pseudo-code for the local search algorithm is given below to express the way in which the local search algorithm work.

Local search (stochastic)

```
i =initialSolution();
best =i;
iterations = 0;
while( depth-not-satisfied )
{
    count = 0;
    // Here starts the algorithm a.
    while( pivot-rule-not-satisfied )
    {
        j =next( N(i) );
        count + +;
        if( f(j) > f(best) )
            best = j;
    }
    // We think of best as the production
    //of the algorithm best = a(i)
    i = best;
    iterations + +;
}
```

In this pseudo-code we can observe a couple of conditions, the *depth condition* and the *pivot rule*. This pair of conditions determine the local search algorithm.

The pivot rule is the algorithm itself, and can be for instance *steepest ascent*, meaning that the whole neighborhood of the solution i is to be searched for the best solution available ($count = |N(i)|$). In the case of *greedy ascent*, we might use the pivot rule of stopping when the first better solution in the neighborhood is found ($count = |N(i)|$ or $best = i$). In practice, as the cardinality of the neighborhood $N(i)$ can be infinite, it is natural to consider only a random sample of size $n \ll |N(i)|$. This type of algorithms are deterministic in nature, but stochastic in behavior.

The depth condition is the termination criteria of the local search. It can range from the one-time local search (when $iterations = 1$), to the local optimality condition ($count = |N(i)|$ and $best = i$).

Another important remark is that stochastic local search algorithms will have a non-deterministic pivot rule. This means that they might accept a solution generated within the neighborhood based on a probabilistic condition. Algorithms like *simulated annealing* fall into this category, where a worst solution might be accepted with low probability.

1.3 Constrained Optimization

Most real world optimization problems are more complex than the problems presented in the last section. In particular, the solutions offered by the optimization process might not be applicable to real world after the over-simplification process of the model.

In order to overcome this problem, the notion of *constrained optimization* was born. It adds to the definition of an optimization problems the notion of *feasible region* and constraints that must be satisfied in order for the solution to be acceptable, but that are not objectives themselves.

1.3.1 Constrained optimization problem definition

A constrained optimization problem is a tuple $(S, N; f; g_1, g_2, \ldots, g_n; h_1, h_2, \ldots, h_m)$, where S is the arbitrary search space, $N : S \to 2^S$ is the neighboring structure, $f : S \to \mathbb{R}$ is the fitness function, $g_i : S \to \mathbb{R}$ which represent the inequality constraints, and $h_i : S \to \mathbb{R}$ which represent the equality constraints.

We call *feasible region* to the set

$$\mathcal{F} = \{x \in S | g_i(x) \leq 0 \forall 1 \leq i \leq n \text{ and } h_j(x) = 0 \forall 1 \leq j \leq m\} \tag{1.3}$$

and a solution x to the problem is *acceptable* if and only if $x \in \mathcal{F}$. When there is a solution x such that $g_i(x) = 0$, the constraint g_i is said to be *active* for x.

The constrained optimization problem is typically stated as

$$\text{optimize } f(x)$$

subject to

$$\begin{aligned} g_i(x) &\leq 0, i = 1, 2, \ldots, n \\ h_j(x) &= 0, j = 1, 2, \ldots, m \end{aligned}$$

and in both cases, equality and inequality constraints, can be linear or non-linear.

The *constrained optimum* is the value x^* such that is acceptable and the global optimum of the transformed problem $(\mathcal{F}, N|_{\mathcal{F}}; f|_{\mathcal{F}})$.

1.3.2 Techniques to handle constraints

In order to solve this type of optimization problems, researchers have developed a number of techniques. Most of them are variation of an already existing technique, or the transformation of the problem to a standard optimization problem that has its global optimum at the constrained optimum of the original problem.

In the following section we will examine many of this techniques.

1.3.2.1 Penalty functions

The first idea used to solve constrained optimization problems was to transform the problem to global optimization one over S, and applying a penalty in fitness to those solutions that lay outside the feasible region. Here we will examine two different techniques that use this idea as inspiration.

Total violation of constraints The first technique used to solve constrained optimization problems was the *total violation of constraints*. This technique consists of changing the fitness function to add a penalty based on constraint violation. Its general form allows a set of parameters to be adjusted for each constraint.

The problem is then transformed to $(S, N; f')$ where

$$\begin{aligned} f'(x) &= f(x) - \sum_{i=1}^{n} w_i g_i^+(x) - \sum_{j=1}^{m} w_{n+j} h_j(x) \qquad (1.4) \\ \text{with } g_i^+(x) &= \max\{0, g_i(x)\} \end{aligned}$$

where the numbers $w_k \in \mathbb{R}_+$ for each $1 \leq k \leq n+m$ represent the weights associated to that constraint function. These weights are not necessarily fixed during the whole optimization process. One my start with small weights in the first stages of the algorithms to then increase them to enforce the constraints later on.

Observe that depending upon the values of $\{w_i\}$, the global optimum of f' can be the constrained optimization. In general, when the weights approach infinity, the global optimum of the function f' approaches the constrained optimum of the function f.

There has been a number of attempts to set this parameters in a self-adapting way, but, because of the simplicity of this technique, they have not worked as expected.

Maximum violation of constraints As with the last technique, this is an early attempt to solve constrained problems. The basic idea behind *maximum violation of constraints* is to take the maximum value of violation of the individual as the penalty to the fitness function, instead of taking the sum of violations.

The problem is then transformed to $(S, N; f')$

$$\begin{aligned} f'(x) &= f(x) - \max_{0 \leq i \leq n} \{w_i g_i^+(x)\} - \max_{1 \leq j \leq m} \{w_{n+j} h_j(x)\} \qquad (1.5) \\ \text{with } g_i^+(x) &= \max\{0, g_i(x)\} \\ \text{and } h_j^+(x) &= |h_j(x)| \qquad (1.6) \end{aligned}$$

where the numbers $w_k \in \mathbb{R}_+$ for each $1 \leq k \leq n+m$ represent, as in the previous case, the weights associated to that constraint function. As before, the weights are not necessarily fixed during the whole optimization process. And yet again, when the weights are close to infinity, the global optimum of f' approaches the constrained optimum of f.

More penalty techniques We can see the last two techniques to handle constraints as a special case of a more general approach. The idea is to create a function to transform the violation value of each constraint to match the desired behavior. Hence, we will define two penalty functions ϕ and ψ taking values of the constraints g_i and h_j respectively to assign a penalty to the original function.

The problem is then transformed to $(S, N; f', \{G_i\}, \{H_j\})$ with

$$f'(x) = f(x) + \phi(g_1(x), g_2(x), \ldots, g_n(x)) + \psi(h_1(x), h_2(x), \ldots, h_m(x)) \tag{1.7}$$

with the only constraint that the functions ϕ and ψ should be non-negative, and be evaluated as 0 when $x \in \mathcal{F}$.

There is a wide range of selection for the functions ϕ and ψ, but they shall not be discussed here, as they are of secondary interest to the aims of this thesis.

1.3.2.2 Rules of feasibility

A more sophisticated approach to solving the constrained problem is the use of rules to decide when a solution is better than another one. The main advantage of these techniques is that they do not need to set parameters to balance the strength of the penalty. Instead, they use a set of rules to establish a natural order of fitness and violation of constraints.

These techniques are well-suited for evolutionary algorithms and other population based problem-solvers, as the comparison of two solutions is made based upon the established rules. The fitness function is then replaced by a binary function

$$b(x, y) = \begin{cases} -1 & \text{if } x \text{ is worst than } y \\ 1 & \text{if } x \text{ is better than } y \\ 0 & \text{if they are incomparables or the same} \end{cases}$$

Total violation rule The first approach on this group of techniques is very similar to the first approach on penalty functions. The binary comparison function uses the total sum of constraints in a similar way than in Equation (1.4). Let $\phi(x) = \sum_{i=1}^{n} w_i g_i^+(x)$, $\psi(x) = \sum_{j=1}^{m} w_{n+j} h_i^+(x)$, and $R(x) = \phi(x) + \psi(x)$, then the binary function can be regarded as

$$b(x, y) = \begin{cases} -1 & \text{if } R(x) > R(y) \text{ or, } R(x) = 0 = R(y) \text{ and } f(x) < f(y) \\ 1 & \text{if } R(x) < R(y) \text{ or, } R(x) = 0 = R(y) \text{ and } f(x) > f(y) \\ 0 & \text{if either } R(x) = R(y) \neq 0 \text{ or, } R(x) = R(y) = 0 \text{ and } f(x) = f(y) \end{cases}$$

This function can be interpreted as follows: x is better than y if and only if x violates less the constraints than y or, they are both feasibles but x has better fitness than y.

This technique can be generalized much like the penalty function techniques, but again, that generalization is out of the scope of this thesis and the exact generalization process is left to the reader.

Multi-objective rules Other, more recent type of rules, are concerned with the notion of multi-objective optimization. This is mainly due to the natural way in which we might transform the constrained optimization problem into a multi objective one, in which every constraint function is also an objective. For this to work, the constraint functions must be transformed to g_i^+ and h_j^+ as before.

Once this is done, the solution to the multi-objective optimization problem defined by the tuple $(S, N; f, \{g_i^+\}, \{h_j^+\})$, contains the solution to the constrained optimization problem $(S, N; f; \{g_i\}; \{h_j\})$.

Before we can define the binary function we need to develop several concepts from the theory of multi-objective optimization.

Given two vectors $\vec{x}, \vec{y} \in \mathbb{R}^k$ $\vec{x}$ is said to Pareto-dominate $\vec{y}$ if and only if, $x_i \leq y_i$ for every $i = 1, 2, \ldots, k$, and $x_j < y_j$ for at least one $j = 1, 2, \ldots, k$. The notation for dominance is $\vec{x} \succeq \vec{y}$ which is read $\vec{x}$ dominates $\vec{y}$. This definition gives us a possibility to compare two multi-objective solutions, in the sense that if $\vec{x} \succeq \vec{y}$, then solution $\vec{x}$ is considered better than solution $\vec{y}$.

When we have a set of solutions (vectors) $X = \{\vec{x}_i\}$, we can define the *Pareto levels* in a recursive manner

$$\begin{aligned} PL(0) &= \{\vec{x} | \forall \vec{y} \in X, \vec{y} \not\succeq \vec{x}\} \\ PL(i+1) &= \{\vec{x} | \forall \vec{y} \in X \setminus \bigcup_{i=1}^{k} PL(i), \vec{y} \not\succeq \vec{x}\} \end{aligned} \tag{1.8}$$

The zero-Pareto level has a special name, it is called the *Pareto front*. For convenience, we will define the function $level(\vec{x}, X)$ as the Pareto level of the vector $\vec{x}$ in the set of solutions X.

Before we can define the multi-objective rules, the following notation will be used in the definitions of the binary comparison functions. Let us define the set

$$\begin{aligned} R &= \{r(x) | x \in X\} \\ \text{where } r(x) &= (g_1^+(x), g_n^+(x), \ldots, g_n^+(x), h_1^+(x), h_2^+(x), \ldots, h_m^+(x)) \end{aligned}$$

representing all the constraint values of a set of solutions $X \subset S$. Observe that $r(x) = \vec{0}$ means that $x \in \mathcal{F}$.

Pareto-rank We are, now, ready to define one of the binaries functions, describing what is known as *Pareto-rank rules*. We define the binary comparison function as

$$b(x,y) = \begin{cases} -1 & \begin{cases} \text{if} & level(r(x), R) > level(r(y), R) \\ \text{or} & level(r(x), R) = 0 = level(r(y), R) \text{ and } f(x) < f(y) \end{cases} \\ 1 & \begin{cases} \text{if} & level(r(x), R) < level(r(y), R) \\ \text{or} & level(r(x), R) = 0 = level(r(y), R) \text{ and } f(x) > f(y) \end{cases} \\ 0 & \text{if } level(r(x), R) = level(r(y), R) \neq 0 \end{cases}$$

for any two values $x, y \in X$. The condition $level(x, R(X)) = level(y, R(X))$ and $r(y) \succeq r(x)$, is not required as one solution cannot dominate any other one of the same Pareto level. Observe that, although R depends on X, this dependence is not made clear for clarity in the formulas.

Feasibility and dominance Another, widely used multi-objective rules is the known as *feasibility and dominance*. The binary comparison function can be described as

$$b(x,y) = \begin{cases} -1 & \begin{cases} \text{if} & r(y) \succeq r(x) \\ \text{or} & r(x) \neq \vec{0} \text{ and } r(y) = \vec{0} \\ \text{or} & r(x) = \vec{0} = r(y) \text{ and } f(x) < f(y) \end{cases} \\ 1 & \begin{cases} \text{if} & r(x) \succeq r(y) \\ \text{or} & r(y) \neq \vec{0} \text{ and } r(x) = \vec{0} \\ \text{or} & r(x) = \vec{0} = r(y) \text{ and } f(y) < f(x) \end{cases} \\ 0 & \text{otherwise} \end{cases}$$

for any two values $x, y \in X$. This function can be interpreted as, from two feasible solutions the best is the one with best fitness function, from two non-feasible solutions take the one that Pareto-dominates, if one is feasible and the other is not take the feasible.

The biggest draw-backs of this rules are that it might be very difficult to find the feasible region in the first place, and that the Pareto dominance decreases in *intensity*[1] with increasing dimensionality.

1.3.3 Stochastic Ranking

The rules as a strategy for constrained optimization are good way to solve a problem, however, due to the problems just mentioned, many researchers in constrained optimization are searching for new techniques that can solve problems more efficiently and in a better way than with the previous techniques.

One of the better attempts to solve these intrinsic problems was made by Runarsson [17] when he proposed the *stochastic ranking*. The main idea behind stochastic ranking is based on a parameter used by the traditional penalty function approach. His notation, however, is a little different from our own, but for clarity, his notation will be used for the rest of this section.

The penalty function approach is

$$f'(x) = f(x) + r_g \phi(g_1(x), g_2(x), \ldots, g_n(x)) \tag{1.9}$$

where

$$\phi(g_1(x), g_2(x), \ldots, g_n(x)) = \sum_{i=1}^{n} (\max\{0, g_i(x)\})^2$$

[1] The probability than one random vector dominates another random one decreases exponentially –as 2^{-d}– with the dimension.

or any other penalty function. The value r_g may be variable over the generation number g.

Runarsson notes that, while this approach works quite well with some problems, it is in general very sensitive to the value of r_g as said in Section 1.3.2.1. If r_g is too small, a non-feasible solution may not be penalized enough, and if it is too big, there will be no room in the optimization process to improve the solution once they are in the feasible region. This is specially true if the feasible region is not connected, and the exploration brought the search in one portion of the feasible region that does not contain the constrained optimum of the problem.

The optimal setting for the values r_g is problem dependent and an optimization problem in it own. As an alternative to this issue, the stochastic ranking defines a way to simulate a dynamic adaptation of the parameters r_g.

1.3.3.1 Constraint handling

For any given penalty coefficient $r_g > 0$ let the ranking of λ individuals be

$$f'(x_1) \leq f'(x_2) \leq \ldots \leq f'(x_\lambda)$$

where f' is the transformation of the fitness function given by Equation (1.9). We will use an abbreviation of Equation (1.9) to simplify notation, and let $f'(x_i) = f'_i = f_i + r_g\phi_i = f(x_i) + r_g\phi(x_i)$.

If we examine two adjacent individuals in the order induced by r_g in function f', we can observe that

$$f_i + r_g\phi_i \leq f_{i+1} + r_g\phi_{i+1}$$

for every $i = 1, 2, \ldots, \lambda - 1$.

We define the *critical penalty coefficient* $\breve{r}_i$ for the adjacent pair i and $i+1$, as

$$\breve{r}_i = (f_{i+1} - f_i)/(\phi_i - \phi_{i+1})$$

where it is assumed that $\phi_i \neq \phi_{i+1}$. Note that if we have r_g fixed, then there are three cases for the inequality to hold.

1. $f_i < f_{i+1}$ and $\phi_i \geq \phi_{i+1}$: The comparison is said to be *dominated by fitness function* and $0 < r_g \leq \breve{r}_i$, meaning that the ordering in fitness function is what is deciding the ordering in f'.

2. $f_i \geq f_{i+1}$ and $\phi_i \leq \phi_{i+1}$: The comparison is said to be *dominated by penalty function* and $0 < \breve{r}_i < r_g$, meaning that the ordering in penalty function is what is deciding the ordering in f'.

3. $f_i < f_{i+1}$ and $\phi_i < \phi_{i+1}$: The comparison is said to be *non-dominated* and $\breve{r}_i < 0$, meaning that the ordering in f' is not decided neither by f nor by ϕ.

Observe that the last possible case $f_i \geq f_{i+1}$ and $\phi_i \geq \phi_{i+1}$ is not necessary, because it contradicts the assumption that $f'_i \leq f'_{i+1}$. The non-dominated case is also one in which the value of r_g has no relevance. Its value is critical, however, when comparing in the first two cases, as the value of $\breve{r}_i$ acts as a threshold to decide whether a solution x_i is better or not than a solution x_{i+1}. For example, if we increase the value of r_g in the first case to be higher than $\breve{r}_i$, then the solution x_i will pass from being better, to being worse than x_{i+1}. For the entire population, the chosen value of $r_g$will determine the fraction of individuals ranked only according to the penalty function, and the one ranked by fitness function.

Observe that not every possible value for r_g can influence this selection. There are upper $\overline{r_g}$ and lower $\underline{r_g}$ bounds such that, if $r_g < \underline{r_g}$, then every comparison among solutions will be based upon fitness function[2], and if $r_g > \overline{r_g}$, then every comparison among solutions will be based upon penalty function[3]. Observe that the values of $\overline{r_g}$ and $\underline{r_g}$ are dependant on the current solutions $x_i, i = 1, 2, \ldots, \lambda$.

It has been discussed previously that neither of those cases will lead to the optimal constrained solution. In this sense, the optimal value for r_g must lay in the range from $\underline{r_g}$ to $\overline{r_g}$, so that the comparison among solutions will be balanced between penalty and fitness function.

1.3.3.2 The Stochastic ranking algorithm

The stochastic ranking is concerned with the simulation of maintaining the value r_g in the range $r_g \in [\underline{r_g}, \overline{r_g}]$. Stochastic ranking uses a probability p_f of using only the fitness function for comparisons in ranking individuals in the infeasible region of the search space.

The ranking is achieved by a bubble-sort-like procedure with an stochastic comparing operator. Th procedure is halted when no change in the rank ordering occurs within a complete sweep. This stochastic ranking procedure can be used as the selection operator of any evolutionary algorithm in which the selection is a sorting of the individuals according to a certain order, and then keeping the best individuals for the next generation. This will be explained in detail in Chapter 2.

Stochastic ranking procedure

```
for( j = 1 to λ )
    I_j = j;
for( i = 1 to N )
{
    for( j = 1 to λ − 1 )
    {
        if( φ(I_j) = φ(I_{j+1}) = 0 or rand()< p_f )
```

[2]Called *under-penalization*

[3]Called *over-penalization*

```
            {
                if( f(I_j) > f(I_{j+1}) )
                    swap( I_j, I_{j+1} );
            }
            else
                if( φ(I_j) > φ(I_{j+1}) )
                    swap( I_j, I_{j+1} );
        }
        if( no-swap-performed )
            i = N; //break the for
    }
```

Observe from this procedure, that the algorithm is performing at most N sweeps through the whole population. When $p_f = 0$, the ranking is over-penalized, and when $p_f = 1$, the ranking is under-penalized, so it is a good idea to take values for p_f that are neither close to 0 nor to 1.

Runarsson [17] notes that if the number N of sweeps the algorithm performs tends to infinity, then the ranking will be determined as follows, if $p_f > 1/2$ then the ranking will be under-penalized, and if $p_f < 1/2$ then the ranking will be over-penalized. This can be regarded as increasing N is effectively the same as varying p_f. By this reason, he decided to set $N = \lambda$, and modify p_f to control the performance of the algorithm.

The result of stochastic ranking in the well known benchmark are given in the appendix, with exception of the function $g02$ since the values obtained in this thesis are much better than the reported by Runarsson.

Chapter 2

Evolutionary Algorithms

The origins of evolutionary computation can be traced back to the late 1950's, however, the new-born field remained relatively unknown to the scientific community for almost three decades, mainly due to the lack of computational power in the early stages of evolutionary computation. With the works of Holland [11], Rechenberg [16], Schwefel [18] and Fogel [8], the evolutionary computation started to grow, and we currently observe a steady increase in the number of publications and conferences in the field.

The most significant advantage of using evolutionary algorithms over other optimization techniques lies in the great adaptability and flexibility of the evolutionary search, along with the robust performance and global search characteristics [1]. In fact, evolutionary computation should be regarded as a general adaptable concept for problem solving, specially well suited for difficult optimization problems, rather than a collection of related and ready-to-use algorithms.

2.1 Definition of an Evolutionary Algorithm

Given an optimization problem $(S; f)$, defined as in Section 1.1, with a search space S, and a function $f : S \to \mathbb{R}$, an evolutionary algorithm is a tuple

$$EA(\Omega, k, \Pi_k, \tau; \Psi, \Phi, \sigma; O) \tag{2.1}$$

where, Ω is the *search space* of the algorithm, $\Pi_k = \Omega^k$ is the set of all possible *populations of size* k and $\tau : \Omega \to S$ is a function mapping the search space of the optimization problem to the search space of the evolutionary algorithm; $\Psi = (\psi_1, \psi_2, \ldots, \psi_n)$, where $\psi_i : \Pi_k \times [0,1] \to \Pi_k$ for every $1 \leq i \leq n$, and represent the *mutation operators*; $\Phi = (\phi_1, \phi_2, \ldots, \phi_m)$, where $\phi_i : \Pi_k \times [0,1] \to \Pi_k$ for every $1 \leq i \leq m$, and represent the *crossover operators*; $\sigma : \Pi_k \times \Pi_k \times \mathbb{R}^k \times \mathbb{R}^k \times [0,1] \to \Pi_k$, and represent the *selection operator*; and $O : \Pi_k \times [0,1] \to \Pi_k$ represents the *order* of the operators.

By notation, let $K = \{1, 2, \ldots, k\}$. We will call ψ_i *mutation functions*, and ϕ_i *crossover functions*. Also, we call *populations* to the elements of Π_k; they will usually

be represented by $P_i = (P_{i,1}, P_{i,2}, \ldots, P_{i,k})$. For the sake of clarity, we will define $\Psi(P_i, r) = \psi_n \circ \ldots \circ \psi_2 \circ \psi_1(P_i, r)$, and $\Phi(P_i, r) = \phi_m \circ \ldots \circ \phi_2 \circ \phi_1(P_i, r)$ to assume the same r will be used in every internal function. This r represents the random number generated to make the operators *non-deterministic*. It is not hard to see that one random number is enough to create an arbitrary amount of random data.

Some times it will be useful to apply the operators directly to individuals (i.e. elements of populations) instead of populations.

In the case of mutation, we will *overload*[1] the ψ_j functions to the functions $\psi_j : \Omega \times [0,1] \to \Omega$, and assume that, if $P_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,k})$, then

$$\psi_j(P_i, r) = (\psi_j(p_{i,1}, r), \psi_j(p_{i,2}, r), \ldots, \psi_j(p_{i,k}, r)) \tag{2.2}$$

As for the crossover operators, we will usually require a more complex mechanism to overload the functions. Lets assume that the set of integers $R = \{r_1, r_2, \ldots, r_n\}$ is such that we can redefine the crossover operators as $\phi_j : \Omega^{r_j} \times [0,1] \to \Omega$, and assume we have a function $s_j : [0,1] \to K^{r_j}$. This function will obtain a vector containing the indexes of r_j individuals from the population P_i to be crossed by the new ϕ_j function. In this sense, obtaining k uniform random numbers v_i from r—one for each new individual in the population—, the crossover function will be given by

$$\begin{aligned} \overrightarrow{x_{j,u}} &= s_j(v_u) \text{ for } 1 \leq u \leq k \\ \text{Let } q_{j,u,t} &= p_{(x_{j,u})_t} \forall 1 \leq t \leq r_j \\ \phi_j(P_i, r) &= (\phi_j(q_{j,1,1}, q_{j,1,2}, \ldots, q_{j,1,r_j}; r), \ldots, \phi_j(q_{j,k,1}, q_{j,k,2}, \ldots, q_{j,k,r_j}; r)) \end{aligned} \tag{2.3}$$

Observe that $\overrightarrow{x_{j,u}}$ is a vector with r_j elements, and that each element $(x_{j,u})_t$ of the vector is a number between 1 and k, so they can serve as indexes for individuals in the population.

The function O is usually defined as

$$O(P_i, r) = \Psi \circ \Phi(P_i, r) \tag{2.4}$$

where $P_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,k})$, and $p_{i,j} \in \Omega$ for every $j \in K$.

In a more general setting, the operators may be applied to populations with a size other than k, but the generalization of the definition of an evolutionary algorithm as stated before is simple and is left to the reader.

The general sketch for the evolutionary algorithm is

Evolutionary Algorithm

```
initialize-population P_0;
Let i = 0;
while( termination-criteria-is-not-met )
```

[1] As in programming, two functions with the same name, but with different kind (number of type) of arguments. In general, it is clear from context whether we are referring to one or another.

```
{
    P_f = O( P_i, rand() );
    F_i = computeFitness(P_i);
    F_f = computeFitness(P_f);
    P_{i+1} = σ( P_f, P_i, F_f, F_i, rand() );
    i = i + 1;
}
```

Each of the loop's cycles are called *generations*, and the termination criteria could be that a certain number of generations have passed, or that a certain amount of fitness function evaluations have been reached, or a more sophisticated test such as a population convergence rate or a generational difference threshold has been met, etc.

Given a population $P = (p_1, p_2, \ldots, p_k)$, the fitness is usually computed as $F = (f \circ \tau(p_1), f \circ \tau(p_2), \ldots, f \circ \tau(p_k))$, where $f \circ \tau(p_i)$ is called the fitness of individual p_i.

The majority of current implementation of evolutionary algorithms descend from three related but independently developed approaches: *Genetic Algorithms*, *Evolutionary Programming* and *Evolutionary Strategies*.

Evolutionary programming was originally offered as an attempt to create artificial intelligence. The approach was to create *finite state machines* (FSM) to predict events based upon former observations. A FSM is an abstract machine which transforms a sequence of input symbols into a sequence of output symbols. The transformation depends on a finite set of states and a finite set of transition rules.

The other two main evolutionary algorithms are more popularly used to optimization and will be given greater attention.

2.2 Genetic Algorithms

Genetic algorithms (GA) were invented by Holland [11] in the 1960's, and were developed by Holland, his students and his colleagues at the university of Michigan for over a decade. Holland's goal, in contrast to that of evolutionary strategies and evolutionary programming, was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop a theory that could aid to import those mechanisms to computer systems.

What Holland developed was a method to move a population of chromosomes[2] to a new population by using an artificial implementation of natural selection together with the genetic-inspired operators of crossover, mutation and inversion. In this mechanism, we have another selection operator to decide which individuals are going to be selected for reproduction. This and the other operators will be analyzed later in greater detail.

[2]In its simplest form this chromosomes are strings of bits.

In the last several years there has been widespread interaction among researchers studying various evolutionary computation methods, and the boundaries between GA, evolutionary strategies, evolutionary computation, and other evolutionary approaches have broken down to some extent.

Nowadays, researchers often use the term genetic algorithm to refer to something quite different from Holland's original conception. In general terms, GAs are the more flexible evolutionary computation algorithms in terms of the available operators and representations.

2.2.1 The Simple Genetic Algorithm

The traditional GA, also known as *Simple Genetic Algorithm* (SGA) is detailed as follows. Using the notation for evolutionary algorithms, we define the simple genetic algorithm as $SGA(p_c, p_m) = EA(\Omega, k, \Pi_k, \tau; \Psi, \Phi, \sigma; O)$, where $\Omega = \mathbb{Z}_2^l$, and the function τ is problem dependent.

It only contains one mutation ($m = 1$) function which, given an individual $p \in \Omega$, and getting random numbers $s \in \{0, 1\}$ and $t \in \{1, 2, \ldots, l\}$ from r,

$$\psi(p, r) = \begin{cases} p & \text{if } s = 0 \\ (p_1, p_2, \ldots, p_{t-1}, 1 - p_t, p_{t+1}, \ldots, p_l) & \text{if } s = 1 \end{cases} \tag{2.5}$$

where the probability of $s = 1$ being known as the *mutation probability* p_m, which is usually set to $1/l$. On the other hand, t is expected to be uniform. We can see a schematic representation in Figure 2.1, where we can observe the mutation spot, and that position is flipped in the individual as a result of the mutation.

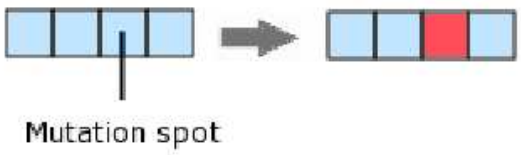

Figure 2.1: The schematic view of the simple mutation operator.

It contains also only one crossover function ($n = 1$) in its crossover operator which first selects the parents with what is called *fitness proportion* or *roulette wheel*. The amount of parents is always 2, which means $r_1 = 2$. The fitness proportional is the function which, given the population $P = (p_1, p_2, \ldots, p_k)$

$$\begin{aligned} s_1(r) &= (x_1, x_2) \\ \text{such that } P(x_1 = i) &= \frac{f(\tau(p_i))}{\sum_{j=1}^{k} f(\tau(p_j))} \\ \text{and } P(x_2 = i) &= \frac{f(\tau(p_i))}{\sum_{j=1}^{k} f(\tau(p_j))} \end{aligned} \tag{2.6}$$

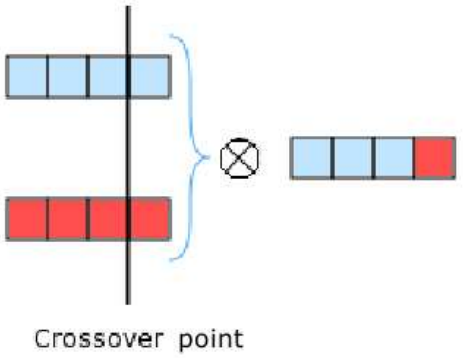

Figure 2.2: The schematic view of the one-point crossover operator.

which can be interpreted as one individual having a probability proportional to that individual's fitness of being selected in the current population. The crossing function is then defined as follows

$$\phi(p_{x_1}, p_{x_2}, r) = \begin{cases} (p_{x_1,1}, p_{x_1,2}, \ldots, p_{x_1,t-1}, p_{x_2,t}, \ldots, p_{x_2,l}) & \text{if } s = 1 \\ p_{x_1} & \text{if } s = 0 \end{cases} \tag{2.7}$$

with $t \in \{2, 3, \ldots, l\}$ being a random number obtained (from r) with uniform probability, and $s \in \{0, 1\}$ is a random number which probability of being 1 is equal to a constant known as the *crossover probability* p_c which is usually set to 0.7, and $(x_1, x_2) = s_1(r)$. The schematic representation of this operator is in Figure 2.2, where we can observe the crossover point, and the resulting individual.

This crossover function is known as *one-point* crossover, because it is equivalent to taking one crossover spot (i.e. the number t) and taking the first t *genes* from the first parent and the rest from the second to create a new individual.

2.2.2 More operators and codings

There are a number of operators for crossing and mutation other than the reviewed in the last section. There are also some coding possibilities for the genotype, instead of the usual $\mathbb{Z}_2^l$. We can even use different cardinalities for every gene, i.e. $\Omega = \mathbb{Z}_{i_1} \times \mathbb{Z}_{i_2} \times \ldots \times \mathbb{Z}_{i_l}$, where $i_j \in \mathbb{N}$ and $1 \leq j \leq l$.

There is also a possibility of using data structures in the place of genes. When a GA has data structures as genes, and operators to act on them are provided, the evolutionary algorithm resulting from it is known as *Genetic Programming* [12].

Inversion operator There is a biologically inspired mutation operator that we will review. It is called *inversion* mutator, and, given the random numbers $s \in \{0, 1\}$, as in the simple mutation, $1 \leq t \leq l - 1$, and $t + 1 \leq u \leq l$ uniform numbers obtained

from r, it can be viewed as the function

$$\psi(p,r) = \begin{cases} (p_1, p_2, \ldots, p_{t-1}, p_{u-1}, \ldots, p_{t+1}, p_t, p_u, \ldots, p_l) & \text{if } s = 1 \\ p & \text{if } s = 0 \end{cases}$$

It could be used to preserve some qualities of the genotype that other mutation operators would destroy, as the sum of the 1's in the genome, or the genes itself, but to change the order[3].

Shuffle operator Another useful mutation operator that preserves the genes in the individual is the *shuffle* operator. It consists of choosing a permutation of size l. This operator assumes an uniform type of genes in each position, i.e. $\Omega = A^l$, where A is the set of possible genes. This operator can be mathematically expressed by

$$\psi(p,r) = \begin{cases} (p_{\alpha(1)}, p_{\alpha(2)}, \ldots, p_{\alpha(l)}) & \text{if } s = 1 \\ p & \text{if } s = 0 \end{cases}$$

where $s \in \{0,1\}$ as usual representing the mutation probability, and the function $\alpha : \{1,2,\ldots,l\} \to \{1,2,\ldots,l\}$ a permutation (i.e. 1–1 and onto) obtained from r.

Two-point crossover There is another widely used crossover operator for GAs, and is known as *two-point* crossover, because it resembles the one-point crossover, but with two crossover spots. Formally, given the random numbers $s \in \{0,1\}$ as in the one-point crossover, $1 \leq t \leq l-1$, and $t+1 \leq u \leq l$ uniform numbers obtained from r, it can be viewed as

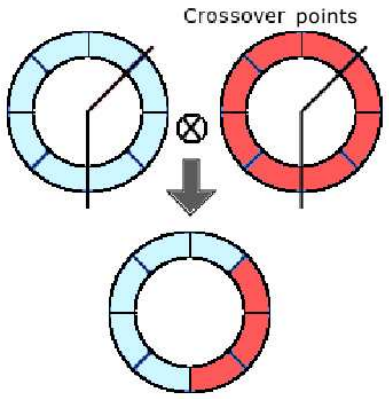

Figure 2.3: The schematic view of the two-point crossover operator. Observe that the genotype is viewed as if it were a ring.

$$\phi(p_{x_1}, p_{x_2}, r) = \begin{cases} (p_{x_1,1}, \ldots, p_{x_1,t-1}, p_{x_2,t}, \ldots, p_{x_2,u-1}, p_{x_1,u}, \ldots, p_{x_1,l}) & \text{if } s = 1 \\ p_{x_1} & \text{if } s = 0 \end{cases}$$

[3]Useful for solving problems as the *traveling salesman problem* (TSP).

This operator has a fame of being better than the classical one-point crosser, and also, it is easy to see that it generalizes it. But there is an even more renown crossover operator.

Uniform crossover The *uniform* crossover is the crossover operator that better preserves diversity in the population. It is a generalization of the one and two-point crossover operators. As its predecessors, it requires a set of random numbers, the first of which is exactly the same as before, $s \in \{0, 1\}$, while the others vary a little; obtain $t_1, t_2, \ldots, t_l$, where $t_i \in \{0, 1\}$ for every $1 \leq i \leq l$, with uniform probability. The function of this operator can then be viewed as

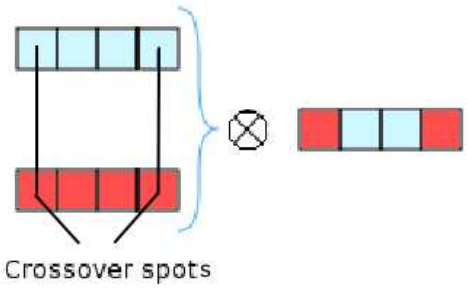

Figure 2.4: The schematic representation of the uniform crossover operator. Note that at every crossover spot, the offspring has the genes of the second parent, while it has the genes of the first elsewhere.

$$\begin{aligned} \phi(p_{x_1}, p_{x_2}, r) &= (q_1, q_2, \ldots, q_l) \\ \text{where } q_i &= \begin{cases} p_{x_1,i} & \text{if } t_i = 1 \\ p_{x_2,i} & \text{if } t_i = 0 \end{cases} \end{aligned} \tag{2.8}$$

This operator is schematically presented in Figure 2.4.

Tournament Aside from crossover and mutation operators, there are many selection operators. Maybe the best known is the *tournament* selection, and its variations. In simple words, it takes a set of individuals at random (usually with uniform probability), and selects the fittest one of them to be part of the next generation. The most used type of tournament is the *binary tournament*, where we are to select a pair of individuals in each step, and then select the best one. Formally, we can define the n-tournament as, getting, as usual from the random number r, uniform random integers $i_{1,1}, i_{1,2}, \ldots, i_{1,n}; i_{2,1}, \ldots, i_{2,n}; i_{k,1}, \ldots, i_{k,n}$, the selection operator would be

$$\begin{aligned} \sigma(P, Q, F_P, F_Q, r) &= (b_1, b_2, \ldots, b_k) \\ \text{and } b_a &= \arg\max_{0 \leq j \leq n} \{f \circ \tau(q_{i_{a,j}})\} \end{aligned}$$

Observe that this selection mechanism ignores the previous generation P and is only concerned with the fitness of the newly generated population Q. This is the usual form of the selection operators in newer genetic algorithms.

One of the main advantages of this selection mechanism is that we don't need to evaluate the fitness of the individuals directly if we have a less-expensive mechanism to decide whether one individual is better than the other.

For example, if we want to solve the problem of controlling a system without making it crash, and the individuals represent the actions to take, we only require to know if one individual is able to maintain the system working for more time than the other, instead of knowing exactly how much time they can both keep it working.

The main disadvantage of them is that the best solution found so far could be lost (i.e. not selected). In order to avoid the lost of the best individual during selection, the operator can be changed to include a number of the best individuals of the previous generation automatically into the next one. This type of selection mechanisms are known as *elitist selection*. The elitism can be of one or two individuals or even the whole population.

Challenge (Probabilistic Tournament) There is a variation of the tournament, less used in the literature, which instead of always selecting the best out of the set of selected individuals, selects the best only with a certain probability. This mechanism is sometimes referred to as *challenge selection* or *probabilistic tournament*.

The selection pressure is a measure of the probability of selecting individuals with low fitness. A high selection pressure gives small or zero probability of selecting the worst individual. The tournament is a good example of a high pressure selection mechanism, while the roulette wheel is the classic example of a middle pressure selection. In the challenge the selection pressure is relaxed compared to the normal tournament, but preserves the good qualities of the tournament over the roulette wheel.

2.3 Evolutionary Strategies

The *evolutionary strategies* (ES) were developed in Germany in the 1960s [16, 18] to solve difficult hydrodynamical problems. It simulates the evolution at an individual level, and as a result, the crossover operator is considered secondary.

The main ideas behind evolutionary strategies are a self-adapting mutation on the individuals, along with a deterministic and extinctive selection[4]. ESs are also under the influence of the neo-Darwinism used in many evolutionary algorithms, and in particular in GAs. The uses and roles are, though, substantially different in ESs than in GAs [4], and we will discuss a little about this differences.

[4] The best individuals are to form the next generation, in consequence, the worst individuals will never be selected.

To begin with, evolutionary strategies are more concerned with phenotype as there is no coding from genotype to phenotype. Also, the crossover is as important to GAs as the mutation is important to ESs. The GA's search progresses through recombination of genes in good individuals, while the search progresses in ES's via the mutation of promising individuals.

The order of the operators is also changed, and the next generation's population is selected after evaluating the offsprings of the last generation, in contrast to the GA's way, in which the selection process is carried away to create the offsprings. This obeys to a philosophical remark. As mutation is viewed as the main operator, mutation is constructing the actual solutions, and its effect should not be disrupted crossing over. The good solutions are thought to come from prior good solutions via mutation. After this, the crossover can try to improve the exploration, but without loosing any mutated individual.

2.3.1 The $ES(1+1)$

The first evolutionary strategy ever made was the $ES(1+1)$, in which only one offspring was generated from one single parent. Needless to say there was no crossover operator in this early version of the ESs. Traditionally, $\Omega = \mathbb{R}^l$, and although we can think of other type of codings, apparently it is part of the definition of a ES to be real coded. This simplifies the function τ in the sense it is simply the identity function. We will use the notation $p = (x_1, x_2, \ldots, x_l)$ for the individual.

The first mutation operator used was simply to add a normal value to every x_i. Formally, this operator can be thought of as obtaining normal values $s_i \sim N(0,1)$ for $1 \leq i \leq l$, and then the mutation function is

$$\psi(p, r) = p + (s_1, s_2, \ldots, s_l) \tag{2.9}$$

This operator offers the advantage of no extra parameters to adapt, but unfortunately has proven insufficient to solve many problems. This si mainly due to the inability of the mutation operator to adapt to a rescaling of the function. It is obviously not the same task to optimize the function $f(\vec{x}) = \prod_{i=1}^{k} x_i$ as it is to optimize $f(\vec{x}) = \prod_{i=1}^{k} 10^9 x_i$, although conceptually the problems are of the same difficulty.

For this reason, a more complex operator was developed.

The $1/5$-rule The first attempt to create a self-adapting mutation was the so-called $1/5$-rule. The idea behind this is to have a control value representing the intensity of mutation to apply. The value of $l_2 = 1$, and by simplicity, we use l instead of l_2. The individual is then defined as

$$p = (x_1, x_2, \ldots, x_l; \sigma)$$

where σ is the intensity of mutation. Then, a new individual is constructed by adding a normal value with the parameter σ as standard deviation. The operator can be

viewed as, obtaining normal values $s_i \sim N(0, \sigma)$ with $1 \leq i \leq l$, and the function is

$$\psi(p, r) = p + (s_1, s_2, \ldots, s_l)$$

This operator would not be very different from the one in (2.9) if the value of σ were fixed. This value, however, is not fixed, but it is updated every certain number of generation (usually 20) as follows

$$\sigma = \begin{cases} 0.82\sigma & \text{if } e < 1/5 \\ 1.22\sigma & \text{if } e > 1/5 \\ \sigma & \text{otherwise} \end{cases}$$

where e is the number of successful offsprings in the last (20) generations. By the number of successful offspring individuals we mean the number of individuals that improved their parent.

As we can see, if the individual is trapped in a particularly difficult local optimum, the number of successful offsprings will very likely be less than 1/5 thus decreasing even more the value of σ and consequently making more and more difficult to escape this local optimum.

This is the main reason why the generalization of the $ES(1+1)$ was developed.

2.3.2 $ES(\mu, \lambda)$ and $ES(\mu + \lambda)$

The basic scheme of the generic ES is, following the formal notation, defined by $ES(\mu + \lambda) = EA(\Omega, k, \Pi_k, \tau; \Psi, \Phi, \sigma; O)$ or $ES(\mu, \lambda) = EA(\Omega, k, \Pi_k, \tau; \Psi, \Phi, \sigma; O)$. The difference between them is in the selection operator, μ represents the number of parents in the population, while λ is the number of offsprings that the parents will have. In $ES(\mu + \lambda)$, the parents are to be compared with their offspring during selection to decide what is going to be the next generation, while in $ES(\mu, \lambda)$, the best μ offspring will completely replace the parents population as the next generation ($\mu \leq \lambda$).

$ES(\mu, \lambda)$ can be seen as the non-elitist version of $ES(\mu+\lambda)$, which has *full elitism*[5].

The most important idea behind the new operators of the more sophisticated ESs is to add a number of new values to the individuals, and use those values to direct the mutation and the search itself. In this sense, the individuals consist of an objective portion (namely, the values of x_i) and a control portion. This is effectively the same as changing $\Omega = \mathbb{R}^{l_1} \times \mathbb{R}^{l_2}$ instead of the usual $\Omega = \mathbb{R}^l$. We will use the notation

$$p = (x_1, x_2, \ldots, x_{l_1}; c_1, c_2, \ldots, c_{l_2}) \tag{2.10}$$

and we will use $\vec{x} = (x_1, x_2, \ldots, x_{l_1})$ to refer to the objective part, and $\vec{c} = (c_1 c_2, \ldots, c_{l_2})$ to refer to the control portion of the individual. For clarity, we will still use the number l, but we will set it to $l = l_1 + l_2$.

[5]We mean by full elitism the behavior of a selection operator in which the only way for an individual to be part of the next generation is by being better (in fitness) and replacing one of the last generation.

Observe that we can define the function $\tau(p) = \vec{x}$, as the control values are not part of the optimization process.

In these methods a deterministic rule—as the 1/5-rule—, is no longer used. Instead, we let the control parameters to self-adapt, and add those parameters for each objective value.

The control parameters are also subject to mutation and recombination, which will allow evolution to select the best values of the parameters by itself. It is expected that those individuals with good control values will end up having a good fitness value, and in the long run, will give birth to better individuals.

2.3.3 More operators

The obvious introduction of crossover operators surges from the availability of many individuals in the population. In ESs there are two types of crossover: *sexual* and *panmitic*. In the sexual crossover, the offspring is generated by exactly two parents, and in the panmitic crossover, we select one individual to play the role of one parent, and for every objective and control value we choose another random (with replacement) parent. In the formal notation, the sexual crossover has values $r_i = 2$, while in the panmitic version, $r_i = l + 1$.

The panmitic version of the crossover operators creates more diversity in the population, but slows down convergence. It is normally used in very difficult problems.

Discrete crossover The first crossover operator used in ESs was the *discrete* crossover. It consists of interchanging values from the parents to create the offspring. This is very similar to the uniform crossover of the GAs. The formal function is as follows

$$\begin{aligned} \phi(p, p', r) &= (q_1, q_2, \ldots, q_l) \\ \text{where } q_i &= \begin{cases} p_i & \text{if } s_i = 1 \\ p'_i & \text{if } s_i = 0 \end{cases} \end{aligned} \tag{2.11}$$

where $s_i \in \{0, 1\}$ is an uniform random number for $1 \leq i \leq l$. The panmitic version of this operator can be defined as

$$\begin{aligned} \phi(p', p_1, p_2, \ldots p_l, r) &= (q_1, q_2, \ldots, q_l) \\ \text{where } q_i &= \begin{cases} p'_i & \text{if } s_i = 1 \\ p_{i,i} & \text{if } s_i = 0 \end{cases} \end{aligned} \tag{2.12}$$

This crossover is the easiest to compute from all, but it is also the one with the worst diversity. Observe that no new value is generated as we only generate a new individual with values already in the population. For this reason, even more sophisticated operators were created.

Intermediate crossover The next used crossover operator is called *intermediate* crossover, and was proposed, as its name implies, to make an offspring at the average of two parents. The formal function of this operator requires no random numbers (except for the selected parents), and is

$$\phi(p, p', r) = (\frac{p_1 + p'_1}{2}, \frac{p_2 + p'_2}{2}, \ldots, \frac{p_l + p'_l}{2}) \tag{2.13}$$

and its panmitic version is

$$\phi(p', p_1, p_2, \ldots p_l, r) = (\frac{p'_1 + p_{1,1}}{2}, \frac{p'_2 + p_{2,2}}{2}, \ldots, \frac{p'_l + p_{l,l}}{2}) \tag{2.14}$$

Observe that this crossover does create new values for the individual. By always averaging two parents (in its sexual form), it tends to make the population converge easily. By generalizing this idea of the average, new operators were proposed.

Generalized intermediate crossover There also exists a generalized version of the intermediate crossover, to allow a weighted average of the two parents. This is known as the *generalized intermediate* crossover. The formal version requires an uniform random number $\eta \in [0, 1]$, and the function is

$$\phi(p, p', r) = (\eta p_1 + (1 - \eta)p'_1, \eta p_2 + (1 - \eta)p'_2, \ldots, \eta p_l + (1 - \eta)p'_l) \tag{2.15}$$

and the panmitic version is

$$\phi(p', p_1, p_2, \ldots, p_l, r) = (\eta p'_1 + (1-\eta)p_{1,1}, \eta p'_2 + (1-\eta)p_{2,2}, \ldots, \eta p'_l + (1-\eta)p_{l,l}) \tag{2.16}$$

Observe that this crossover has the possibility of generating new individuals along the line segment joining the two parents (in the sexual version). This notion can be even more general, as we are still confining the search for offsprings to a relatively small space.

Generalized crossover The last crossover to discuss here is called *generalized* crossover, and creates offsprings on the hyper-cuboid with corners on the parents. That is, instead of using the same value as the weighted average of the parents, a random value $\eta_i \in [0, 1]$ is created for each value $1 \leq i \leq k$, and the weighted average is created for each value. The formal function is

$$\phi(p, p', r) = (\eta_1 p_1 + (1 - \eta_1)p'_1, \eta_2 p_2 + (1 - \eta_2)p'_2, \ldots, \eta_l p_l + (1 - \eta_l)p'_l) \tag{2.17}$$

and its panmitic version is

$$\phi(p', p_1, p_2, \ldots, p_l, r) = (\eta_1 p'_1 + (1 - \eta_1)p_{1,1}, \eta_2 p'_2 + (1 - \eta_2)p_{2,2}, \ldots, \eta_l p'_l + (1 - \eta_l)p_{l,l}) \tag{2.18}$$

An important remark is that, unlike the GA's crossover operators, these operators can be applied to either only the objective values ($\vec{x}$) or to the control values ($\vec{c}$), thus increasing the matching possibilities to create a complete crossover operator.

In general, it is used the generalized intermediate, or the generalized crossover on the objective values, and discrete on the control values, but other combinations are equally possible.

Control mutation The natural way to extend the individuals is to add a control parameter for each objective parameter to optimize. In this sense, the mutation will be controlled by these parameters. In this case, $l_1 = l_2$, and $\Omega = \mathbb{R}^{l_1} \times \mathbb{R}_+^{l_1}$, and thus

$$p = (\vec{x}; \vec{\sigma}) = (x_1, x_2, \ldots, x_{l_1}; \sigma_1, \sigma_2, \ldots, \sigma_{l_1}) \tag{2.19}$$

Observe the difference against (2.10), in which only one control value was used. As stated before, the control values are not to be changed by a deterministic rule, but by another mechanism.

The control mutator function can be defined with $l_1 + 1$ standard normal values $t', t_i \sim N(0,1)$, and l_1 normal values $s_i \sim N(0, \sigma_i \exp(\tau'' t' + \tau' t_i))$, for every $1 \leq i \leq l_1$. The function is then defined as

$$\psi(p,r) = (\vec{x} + (s_1, s_2, \ldots, s_{l_1}); \sigma_1 \exp(\tau'' t' + \tau' t_1), \sigma_2 \exp(\tau'' t' + \tau' t_2), \ldots, \sigma_{l_1} \exp(\tau'' t' + \tau' t_{l_1})) \tag{2.20}$$

where $\tau' = \frac{1}{4\sqrt[4]{k_1}}$ and $\tau'' \frac{1}{\sqrt{2k_1}}$. These values are parameters to compensate the high dimensionality of some problems, and are functionally equivalent to the *learning factor* used in artificial neural networks. These constants are usually referred to as τ and τ' instead of τ' and τ'', however, due to the existence of the mapping τ in the definition of the EA, we opted to avoid the ambiguity by using an extra prime in the constants.

Observe that the values of the σ's are updated before the objective values, and also, observe that only one random value is generated to be multiplied by τ'', while new random numbers are generated for every value to be multiplied by τ'.

Correlated mutation Another type of mutation proposed by Schwefel was the *correlated* mutation, which main objective was to perform mutations in directions not aligned with the coordinate axis. By performing a rotation in space, we allow the mutations to align with more general search directions, and make the optimization process faster.

Schwefel observed that, in general, the path of one individual and its offspring is roughly perpendicular to the optimal step (i.e. the vector joining the present individual to the optimal one). By this reason, a better direction can be used to allow a faster convergence ratio. A natural way to do this was to use the correlation matrix of the successful offsprings to choose a direction. It has been proved, however, that the same effect can be achieved by using a series of canonical rotation angles.

A correlated mutation is achieved by rotating a non-correlated mutation by an angle θ over one hyper-plane. The total number of angles required to define every possible rotation in an l_1-dimensional space is $\begin{pmatrix} l_1 \\ 2 \end{pmatrix} = l_1(l_1 - 1)/2$. We can, then, define $\Omega = \mathbb{R}^{l_1} \times \mathbb{R}_+^{l_1} \times (-\pi, \pi]^{l_1(l_1-1)/2}$, which sets the individuals as

$$p = (\vec{x}, \vec{\sigma}, \vec{\theta}) = (x_1, \ldots, x_{l_1}; \sigma_1, \ldots, \sigma_{l_1}, \theta_1, \ldots, \theta_{l_1(l_1-1)/2}) \tag{2.21}$$

where $\vec{c} = (\vec{\sigma}, \vec{\theta})$, and $l_2 = l_1 + l_1(l_1 - 1)/2$.

This mutation operator is very similar to the control mutation, except that the θ's are updated before the objective values. That is, getting $l_1(l_1-1)/2$ standard normal values $\alpha_i \sim N(0,1)$, and l_1 more normal values $\gamma_i \sim N(0, C(\sigma, \hat{\theta}))$, the formal operator can be regarded as

$$\psi(p,r) = (\vec{x} + (\gamma_1, \ldots, \gamma_{l_1}); \sigma_1 \exp(\tau'' t' + \tau' t_1), \ldots, \sigma_{l_1} \exp(\tau'' t' + \tau' t_{l_1}); \hat{\theta}) \quad (2.22)$$

where $\beta \approx 0.0873$, $\hat{\theta} = \vec{\theta} + \beta(\alpha_1, \alpha_2, \ldots, \alpha_{l_1(l_1-1)/2})$, and $C(\sigma, \hat{\theta})$ is the covariance matrix. And one way to obtain this covariance directions is given in the next algorithm

Covariance directions

```
for( i = 1 to l_1 )
    Δx_i = σ_i exp(τ''t' + τ't_i) s_i;
for( m = l_1(l_1 − 1)/2 to 1 )
{
    (i, j) = indexOf(m); //Get the indexes that θ_m affects.
    Δx_i = Δx_i cos θ̂_m − Δx_j sin θ̂_m;
    Δx_j = Δx_i sin θ̂_m + Δx_j cos θ̂_m;
}
for( i = 1 to l_1 )
    x_i = x_i + Δx_i;
```

As we can see, the directions are given in inverse order. This is due to the canonical transformation in Euler's rotations in a k_1-dimensional space, as the rotations end up representing the product of the rotation matrices with rotation angle $\hat{\theta}_m$.

2.3.4 A simple evolutionary strategy for constrained optimization

In this section we will give an example of a simple evolutionary strategy to solve constrained optimization problems using rules to rank individuals.

The ES used is a $ES(70+130)$, with control individuals as in Equation 2.10, using intermediate generalized crossover—Equation(2.15)— on objective values and discrete crossover —Equation (2.11)— on control values. The mutation used is the standard for control individuals as in Equation (2.20).

The binary comparison function used to sort the individuals for selection is the total violation rule explained in Section 1.3.2.2.

This ES is used for comparison with the Baldwinian algorithms explained in Chapter 4.

2.4 Memetic Algorithms

Another type of evolutionary algorithms are known as *memetic algorithms* (MA). They can be thought of as *hybrid* algorithms as they incorporate a local search in their search process [7].

2.4.1 Definition of a Meme

The concept of a *meme* was first introduced by Dawkins [6], where he proposes a social equivalent to the gene as a basic unit for inheritance. According to Dawkins, ideas evolve in culture much like organisms evolve in biological evolution. The basic unit of cultural transmission is then called a meme.

Examples of memes are spoken sentences, written sentences, live music, recorded music, theater, cinema and many more. They are the means by which we express our ideas, while the ideas themselves can be regarded as the phenotype of the meme.

2.4.1.1 Memes and Lamarckism

Dawkins suggested that memes evolve by Lamarckian mechanisms. However, it is possible that memes are a type of Darwinian evolution [20]. When a human brain receives a meme, the meme slowly matures into an idea. Eventually the *host* person can decide to communicate his idea to another person.

This process seem to be less Lamarckian than originally thought, as the changed meme itself (genotype) is not transmitted, but the idea (phenotype) instead. If the meme were changed by an individual, it is not tractable to recognize the meme, but perhaps the similarities that the idea (phenotype) has with the original meme; also, if the meme itself changed, instead of just its representation, it would mean that a reverse engineering process actually occurred in the host brain. Besides, the new host receives the idea, but the meme that *colonizes* this new host is different from the actual idea he received, as the idea was transformed by the previous person.

This might point to an internal evolution where the received meme interacts with many other memes in the host brain giving birth to new memes with crossover and mutation. The transmitted memes are also selected from a pool of memes inside the host brain. These mechanisms tend to point to a Darwinian model of memes.

Memes, though, are generally regarded a Lamarckian, and the definition of a memetic algorithm states this clearly. This discussion will be useful, nevertheless, in Chapter 3, when we will try to create a new algorithm based on the idea of non-Lamarckian local searches.

2.4.2 Definition of a memetic algorithm

From the point of view of the study of adaptive systems, it is the idea of memes as agents that can transform an individual what is of major interest. We can consider the addition of a learning phase to the evolutionary cycle as a form of meme–gene

interaction. This interaction can aid evolution considering the genes to be *plastic* and allowing them to be guided by the learning mechanism.

The basic idea behind MAs is to have at least one local search mutation operator among its operators (an in the evolutionary algorithm). This local search operator is usually applied after the crossover and mutation operators have been applied.

The result of the local search replaces (Lamarckian) the individual if the found solution is better than the initial one. In this sense, if we have a local search algorithm $a : \Omega \to \Omega$ that takes initial points and returns the result of the local search, the memetic learning can be viewed as

$$\psi_{memetic}(p, r) = \begin{cases} p & \text{if } f \circ \tau(a(p)) > f \circ \tau(p) \\ a(p) & \text{if } f \circ \tau(a(p)) \leq f \circ \tau(p) \end{cases}$$

A more rigorous definition of a local search algorithm can be found in Section 1.2. In general, the only thing that makes a MA different from other EAs is the inclusion of this other algorithm. The local search is used to smooth the fitness landscape as we are now searching with evolution not on the normal search space, but on the set of local optimum solutions.

Within a memetic algorithm, we can consider the local search stage to occur as an improvement within the evolutionary cycle, and we should consider if whether the changes made to the individual should be kept or whether the improvement is only to affect the fitness associated with it.

This idea is precisely the motivation of this thesis, and will be dedicated a Chapter on its own. In short, the decision of whether the change is made to the individual (a Lamarckian behavior) or to the fitness (a Darwinian behavior) is what makes the difference between the memetic algorithms and the Baldwinian algorithms.

All this might make more sense if we think of meme evolution as a Darwinian mechanism instead of a Lamarckian one. Turney [20] gives reasons why memes are not necessarily Lamarckian, as well as reasons why memes could be Baldwinian. This discussion might be relevant to decide whether the name memetic algorithm is a misnomer or not, but is not of direct interest to this thesis.

2.5 Differential Evolution

One of the most recent and famous evolutionary algorithms in the literature is the *differential evolution* (DE). Created by Price and Storn [15], the DE is a little different from traditional evolutionary algorithms in the sense that it has only one operator to perform all the searching process. It is, in contrast to genetic algorithms and evolutionary strategies, not based on recombination and mutation to perform the search, but on a more mathematical than biological operator that gives his name to the algorithm.

The basic idea behind DE is to take the difference of two randomly chosen vectors in the population and make a weighted sum of this difference with another randomly

chosen vector and compare it with the original one to place a new individual for the next generation. If this new individual turns out to be better than the individual in the current position, then the old individual is replaced by the new one.

Because no crossover is performed, DE is highly susceptible to parallelization. It is also fast and efficient for global optimization, and it also has a small number of parameters, which have, in great measure, won for itself most of its fame.

2.5.1 The DE_1 algorithm

The formal specification of the differential evolution can be regarded as $DE\_1(F) = EA(\Omega, k, \Pi_k, \tau; \Psi, \Phi, \sigma; O)$, where $\Omega = \mathbb{R}^l$, the function τ is the identity, and Φ is also the identity (i.e. no crossover), thus $O = \Psi$, and the selection mechanism is as follows

$$\begin{aligned} \sigma(P, Q, r) &= (b_1, b_2, \ldots, b_l) \\ \text{where } b_i &= \arg\max\{f \circ \tau(p_i), f \circ \tau(q_i)\} \end{aligned}$$

i.e. it compares only the individuals at corresponding positions in the current population P and the newly generated one Q.

The individuals are vectors defined by

$$p = (x_1, x_2, \ldots, x_l)$$

The only mutation operator is, as described above, what gives its name to the differential evolution, and the classical one is defined next. Getting random integer numbers $s_1, s_2, s_3 \in \{1, 2, \ldots, k\}$ without replacement from r, the *difference* operator can be defined as

$$\psi(p, r) = p_{s_1} + F(p_{s_2} - p_{s_3}) \tag{2.23}$$

Observe that the only thing that matters about the parameter p is its position in the current population P, as it is not used to decide the new vector generated by ψ.

Once we have generated the population P_f from the population P_i, we can proceed to selection, and then to the iterative step in the evolutionary algorithm.

The parameter F controls the strength of the difference operator. It is usually close to 1, but depend on the size k of the population. If the population size is small, a $F = 1$ should be used, if the population size is large, a $F \leq 0.9$ should work fine.

This is due to the special behavior of the difference operator. Mathematically, if the population is near convergence, it is expected that the operator will create small changes[6], on the other hand, if we take one good solution with a bad one for the difference, the rough direction of the difference will be towards the optimum (or completely away from it if it has the opposite sign), this is why it manages to find optimal solution while searching.

This method has the risk, however, of premature convergence, and as it does not have a mechanism to avoid it, several runs might be necessary to achieve the actual optimum.

[6]The difference among two vectors will be small if the individuals are close enough.

The selection pressure of this EA is also interesting to analyze. As one individual is only compared to a newly generated one in the selection process, it is fairly easy that the worst individual will survive. In fact, when in the middle of the process, the worst individual so far might very well survive for several generations if is has a bit of luck. This might suggest that the selection pressure of the selection operator is weak.

On the other hand, however, once a cluster of the population starts to converge, the probability of having new individuals generated near that cluster increases very quickly, thus creating a cycle in which more and more individuals are dragged to this zone. In consequence, the selection pressure for individuals far from this cluster increases almost exponentially.

In conclusion, differential evolution seems to have, implicitly, a self-adaptive selection pressure, starting weak and maintaining so for several generations, and then abruptly starting to grow to the point in which no new solutions out of the (sub)optimal cluster are tolerated by the selection operator.

2.5.2 The DE_2 algorithm

The second variation known as $DE\_2(\lambda, F)$ is somehow based on *particle swarm optimization* as it uses the current best found solution to direct the search. Formally, this difference operator can be regarded as

$$\begin{aligned} \psi(p, r) &= p_{s_1} + \lambda(p_b - p_{s_1}) + F(p_{s_2} - p_{s_3}) \\ \text{where } p_b &= \arg\max_{1 \leq i \leq k} \{f \circ \tau(p_i)\} \end{aligned} \tag{2.24}$$

and the variable λ is a control value used to control the greediness towards the best solution so far. It should be small normally, unless the global optimum is relatively easy to find.

2.5.3 More operators

As is usually the case with evolutionary algorithms, there is a number of other operators used to improve the performance of the DE algorithm.

Here we will only discuss the *pseudo-crossover* performed to increase the diversity in the population. When this operator is working, it is used over one of the difference operators explained in Equation (2.23) and in (2.24). This operator requires another variable, CR, representing the crossover rate. It is usually set to a high value (near 1), except for easy optimization problems.

Suppose the function ψ' is defined as either (2.23) or (2.24), and obtain two random integers $d \in \{1, 2, \ldots, l\}$ and L such that $P(L \geq v) = (CR)^{v-1}$, $v > 0$. The new mutation (pseudo-crossover) operator is defined as

$$\begin{aligned} \psi(p, r) &= (v_1, v_2, \ldots, v_l) \\ \text{where } v_j &= \begin{cases} \psi'(p, r)_j & \text{if } d \equiv d, d+1, \ldots, d+L-1 (\text{mod } l) \\ p_j & \text{otherwise} \end{cases} \end{aligned}$$

where $\psi'(p,r)_j$ is the j-th value of the vector $\psi'(p,r)$.

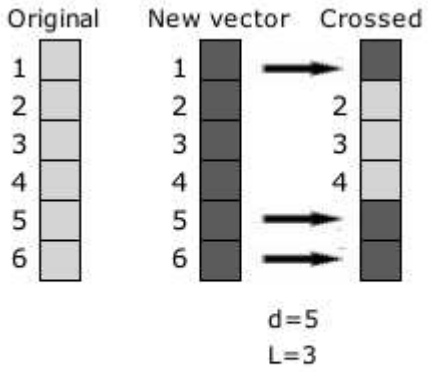

Figure 2.5: The schematic view of the pseudo-crossover operator for differential evolution. We can observe that the crossed vector has 3 values of the original vector, and 3 from the new one.

The sketch of this operator can be observed in Figure 2.5. The individual depicted there has length $l = 6$, the values used for the pseudo-crossover are $d = 5$ and $L = 3$, and then the new individual shares three values with the original one, and three with the new one, beginning at d and circling around in a modular fashion. This operator resembles the two-point crossover of GAs.

2.5.4 Differential evolution for constrained optimization

In this section we will give an example of a simple evolutionary strategy to solve constrained optimization problems using rules to rank individuals. As in Section 2.3.4, we will adapt the DE_1 to solve a benchmark of constrained optimization problems.

The DE picked uses the pseudo-crossover operator mentioned above, and it is then stated as a $DE\_1(0.9, 0.9)$, with normal parameters. The binary comparison function used to accept individuals in selection is the total violation rule explained in Section 1.3.2.2.

This DE is used for comparison with the Baldwinian algorithms in Chapter 4.

Chapter 3

The Baldwin Effect

Many researcher have drawn analogies between learning and evolution as two *intelligent* processes, one taking place during the lifetime of an organism, and the other taking place over the evolutionary history of life on Earth. We tend to regard the evolutionary process as adaptive and intelligent in the sense that individuals are (sub)optimal solutions to the problem of staying alive. In this sense, there is an optimization process undergoing evolution. The question remains, though, as if learning can have an impact at all in the evolutionary mechanisms in nature, and if so, to what extent.

Since the moment in 1987 that Hinton and Nowlan [10] published their classic paper, a large number of researcher have worked in experiments concerning the *Baldwin Effect* in evolutionary computation[14, 2]. Many of them have also observed the synergic effect that learning[1] can have in the evolutionary mechanisms when there is an evolving population of individuals. This synergy is what is usually called the Baldwin Effect. In general, there seems to exist a misunderstanding of the real aspects behind this effect, and, apparently, the researchers have left aside another equally important aspect of it.

At a first approach, we can think of the whole Baldwin Effect as a two-sided coin. In one face, one has the observed behavior that lifetime learning can, under certain circumstances, accelerate the evolutionary process in a population. In the other one, we must take into account that it is costly for an individual to learn.

In this line, there is indeed a synergy effect that can occur during evolution with individuals that are able to learn, but there is also a cost associated with that learning ability. The Baldwin Effect is concerned with both aspects.

This chapter is mainly concerned with the understanding of the Baldwin Effect as a biological mechanism that may or may not be present in nature, but that can be of use for the evolutionary computation community as a new search strategy. It is also the aim to demystify the relation between Lamarckism and Baldwinism in a system, and the possible uses that both may have in optimization problems.

[1] Actually, phenotypic plasticity, but we will talk about it later in this chapter.

3.1 Basic Concepts

In order to fully understand the Baldwin Effect, a number of concepts must be developed in advance. The Baldwin Effect is a misnomer because it was discovered independently by Baldwin, Morgan and Osborn (1896), and also because it is not a single effect, but rather a cluster of effects or observations.

It is relatively well know the difference between the *genotype* and the *phenotype*. The genotype stands for the internal heritable material of an individual, it codes the final utter aspects of the individual in a persistent and unchangeable[2] way. It is typically represented by the organism's DNA. It obtains its name from the *genes*, which are considered the atoms of inheritance. On the other hand, the phenotype is the physical realization of an organism's genotype. It refers to every represented aspect that was implicit in the genetic code, and was developed as part of the individual. It includes from the body composition to the behavioral traits, and the abilities to adapt any of these based on an inherited characteristic. They can be viewed as the observable aspects of the organism's genotype. It obtains its name from the Greek word *phainein*, which means *to show*.

The key term in the Baldwin Effect is known as *phenotypic plasticity*, which can be regarded as the ability of an organism to adapt to its environment due to the features of the phenotype. There are many examples of phenotypic plasticity in nature, most of which have a direct relation with the organism's body in its environment; for instance the ability of the skin to tan when exposed to the Sun, or to form callus when constantly abraded, or many conditioned behaviors acquired by association[3].

Another concept is the notion of *lifetime learning*, which is the set of learning that happens during the lifetime of an individual. It is only concerned with the learning made by a single individual and not with the macroscopic population level of learning in which the evolution may fall into. The impact of lifetime learning on evolution is only one example of the Baldwin Effect; in its most general sense, it deals with the impact of phenotypic plasticity as a whole, on the evolution of a species.

In contrast to the phenotypic plasticity, we call *phenotypic rigidity* the inability of an individual to adapt to a new problem. This inability, contrary to what the intuition dictates us, may be an advantage over more plastic individuals. We will explore this in more detail.

3.1.1 Benefits of phenotypic rigidity

Phenotypic rigidity can be advantageous to an organism in many situations. A *hard-coded* behavior is potentially less hazardous to an individual than a plastic one. For example, learning requires experimentation, and in the case of a potentially fatal

[2]Not quite unchangeable since the individual can mutate, but in general terms it is not susceptible to changes.

[3]Like the famous Ivan Pavlov's experiments on conditioned response on dogs.

behaviors[4], instinct will certainly have an advantage over learning, because an individual will be born with a natural avoidance behavior instead of with trial-and-error learning ability. Another example could be the time required to form a callus which could be used in some other activities if the organism were born with a thick skin[5].

In general, an individual with an instinctive behavior, will require much less energy and will save time. The behaviors are ready for him to use at birth-time. In contrast, plasticity offers the possibility to adapt, but the cost of developing the required behavior, has potentially fatal consequences.

3.1.2 Benefits of phenotypic plasticity

In contrast, phenotypic plasticity enables an organism to explore new possibilities of potentially better behaviors. This may be a great advantage in changing environments or in environments that abruptly changed and are to remain so. The specialization is an observed characteristic of phenotypic rigidity, but can lead to a disaster when taken to the limit[6]. If the rigidity will not allow an individual to adapt to an already changed environment, then, clearly the plasticity will bestow the individual that has it with an evolutionary advantage over those who does not have it.

In general terms, the phenotypic plasticity smooths the fitness landscape enabling the organism to explore neighboring areas of the phenotype space, and thus allowing the individual to have an effective fitness of a local maximum in this space. If a certain continuity in the mapping from genotype to phenotype is assumed, a (potentially) worst genotype would have a better fitness through plasticity than a better genotype.

Behaviors tend to be more plastic than physical structures. The process of learning a behavior represents appropriate changes in the nervous system, and it is in general true that the nervous system of an organism is more flexible than many other physical structures.

3.1.3 Lamarckism and Baldwin Effect

The Lamarckian hypothesis states that the traits acquired during an organism lifetime can be transmitted via inheritance to the organism's offsprings. This hypothesis is generally interpreted as referring to acquired physical traits[7], but something learned during lifetime can also be considered an acquired trait.

To put it in simple terms, Lamarck says that the son of an athlete is more likely to be a good athlete, and the son of a scientist tends to be more intelligent. Thus, a Lamarckian view would hold that learned knowledge can (and will) guide evolution by directly passing the knowledge to the next generation. However, due to overwhelming evidence against it, the Lamarckian hypothesis has been rejected by virtually all

[4] Like learning not to eat a poisonous fruit.

[5] For example the elephant.

[6] As is the case with the Koala, whose diet is confined to a single dish: the eucalyptus' leaves

[7] Such as physical defects due to environmental toxins

biologists. Lamarckism requires an inverse mapping from phenotype and environment to genotype, and this mapping is biologically implausible [14, 20].

It would seem that the rejection of the Lamarckian hypothesis leaves out the question of if learning has any impact on evolution, but the answer seems to be that learning can indeed have a significant effect, though in a less direct way than Lamarck suggested. The Baldwin Effect is purely Darwinian (in contrast to Lamarckism) and it does not involve any reverse mapping.

Suppose the typical example of Lamarckism, with a short-necked animal that learns to stretch its neck to reach leaves on a tall tree. Lamarck believed that the animal's offsprings would inherit slightly longer necks than they would otherwise have had. It requires a mechanism for modifying the parent's genes based on the habit of stretching its neck.

The Baldwin Effect has observable consequences that are similar to Lamarckian evolution. Baldwin would have pointed that if stretching their necks helps towards their survival, then the organisms that are more able to learn to stretch their necks will have the most offspring, thus effectively increasing the frequency of the genes responsible for learning. In this sense, if the environment remains relatively fixed, so that the best thing to learn remain constant, this can lead, via selection, to a population of animals very good at stretching their necks.

There can be advantages, however, in being born with a longer neck. And it is believed that if given enough time, the evolution process will be able to evolve longer necks in the population, which will lead in its turn, to a genetical encoding of longer necks.

One may view this process as if the Baldwin Effect were Lamarckian in its results, but not Lamarckian in its mechanism. Given a desirable trait, the Baldwin Effect only provides the required time (via acquiring the trait due to phenotypic plasticity) for the trait to appear in the population's genes (via the evolutionary process).

3.1.4 The Darwinian mechanism

The evolutionary biologist G. G. Simpson, studied the conjectures made by Baldwin [19] and pointed out that it is not clear how the necessary correlation between phenotypic plasticity and genetic variation can take place. We mean by correlation the requirement that genetic variations happen to occur and produce the same adaptation that was previously learned. This kind of correlation would be easy understood if genetic variations were *directed* towards some particular outcome rather than at random. But randomness is central in modern evolution theory, especially concerning genetic variation, and a specific correlation would mean a Lamarckian mechanism for evolution.

It seems that Baldwin was assuming that, given the laws of probability, correlation between phenotypic adaptations and random genetic variation will happen, especially if the phenotypic adaptations keep the lineage alive long enough for these variation to occur. It does not point, however, to a specific correlation among them. Simpson

agreed that this was possible in principle, but remains unknown if it is an important force in evolution.

While it appears that we are at a dead end, it may not be the case, as the answer to that question may be found in the work of Waddington [22], who proposed a mechanism called *genetic assimilation*. This mechanism is concerned with the inheritance of acquired traits, but tries to explain the underlying process from a slightly different point of view. It states that some sudden and potentially deadly changes in an environment would require phenotypic adaptation that are not necessary in normal environments. If organisms are subject to such changes, they can sometimes adapt during its lifetime because of their inherent plasticity, thereby acquiring new physical or behavioral traits. If the genes of these traits are already in the population, but are *dormant*[8], they can fairly quickly be expressed in the changed environments, and as in the Baldwin Effect case, especially if the acquired traits prevent the individuals from dying.

Waddington even demonstrated that it has happened in several experiments on fruit flies. It suffers, however, of the same skeptical point of view offered by Simpson: there is no final proof that this effect is indeed an important force in evolution. However, although the genetic assimilation is better known in the evolutionary biology community than the Baldwin Effect is, the later has been recently picked up by evolutionary computing researchers mainly because of the experiment made by Hinton and Nowlan, and because it has proven useful in several research areas.

3.2 Baldwin Effect and Computer Science

There is a common feeling to think that learning is always good, at least that is what our nature tends to tell us. As we have observed before, this may not always be the case, and this might be particularly true when confronted to the world of computers, when CPU time and memory requirements are crucial in the analysis of a new algorithm. Evolution is constantly selecting the best balance between learning and instinct, and this balance is usually not fixed during all the optimization process. It varies dramatically when species are confronted with an abrupt change in their environment and also when the environment has achieved an *epistatic* state[9].

There is a number of interesting experiments applying the Baldwin Effect to evolutionary computing on various settings, mainly dedicated to observe the interactions between learning and instinct. Peter Turney [20] presented a list of observations, based on the fundamental insight that there are trade-offs between learning and instinct[10], and are reproduced in Table 3.1.

[8] Here we say that a gene is dormant if it is not usually expressed in the population's phenotype, in contrast to expressed if the trait it codes actually appears in the population.

[9] Roughly speaking, an state in which there are no more sudden changes.

[10] We have been using learning as a form of phenotypic plasticity and instinct as phenotypic rigidity, the generalization to other kinds of phenotypic behaviors is fairly straightforward and is left to the reader.

	dimension of trade-off	phenotypic rigidity (instinct)	phenotypic plasticity (learning)
1	time scale of environmental change	relatively static	relatively dynamic
2	variance, reliability	low variance, high reliability	high variance, low reliability
3	energy, CPU consumption	low energy, low CPU	high energy, high CPU
4	length of learning period	short learning period	long learning period
5	global versus local search	more global search	more local search
6	adaptability	brittle	adaptive
7	fitness landscape	rugged	smooth
8	reinforcement learning versus supervised learning	reinforcement learning	supervised learning
9	bias direction	string bias; direction of bias crucial to success	weak bias; direction of bias not as important
10	global goals versus local goals	emphasis on global goals	emphasis on local goals

Table 3.1: Reproduction of *tradeoffs in evolution between phenotypic rigidity and phenotypic plasticity [20]*

According to Turney [20], the course of the balance is not the main concern of the Baldwin Effect, but the fact that there are trade-offs. In this sense, we try to examine the trade-offs offered by Turney in order to clarify the possible applications of the Baldwin Effect.

Time scale of environmental change. Evolution and learning operate at different time scales. In a dynamic environment, evolution cannot adapt fast enough, so learning is better. In a static environment, evolution can adapt, so learning is a waste of time.

Variance and reliability. Learning is heavily based on experience and requires that the right kind of experience is present in order to acquire the desired learned behavior. This makes learning more stochastic than instinct. Learning increments the variation in the population (different stimuli lead to different behaviors) which can aid evolution.

Energy and CPU consumption. Any individual must expend energy in order to learn. The local search associated with learning consumes evaluation of the fitness function (CPU time), and less resources are left for evolution.

Length of learning period Once an individual is born, it must dedicate some time to learn a trait, if it is instinctive, it is available at birth-time. Shorter learning times are usually preferred by evolution.

Global versus local search. Evolution performs a global search, while individuals perform a local search (in phenotypic space). This trade-off varies greatly depending on the stage of the evolution and the current population.

Adaptability. Learning is more able to adapt to a variation in the environment while instinct tends to be brittle.

Fitness landscape. Learning, as discussed before, smooths the fitness landscape effectively removing rugged areas in the phenotypic search space. It is only advantageous if the landscape was not already smooth in which case it is less useful.

Reinforcement versus supervised learning. An evolutionary algorithm is a type of reinforcement learning for high fitness areas of the search space. In terms of feedback from the environment, it is situated somewhere between unsupervised and supervised learning. Supervised learning obtains more feedback from the environment and is more alike to the local search performed by learning as phenotypic plasticity.

Bias direction. The bias is a term widely used in machine learning, but has recently attracted the attention of the constrained optimization community. The bias direction has two components, the direction and the strength. If the

direction is wrong to a certain problem, the strength will either allow or restrict the exploration process of the algorithm, and learning is better suited. If the direction is correct, an strong bias (instinct) will be better suited for the problem.

Global versus local goals. Evolution and learning have different goals. Evolution seeks to maximize fitness while individuals have more immediate goals. Learning is used by individuals to help them achieve their immediate goals in a better way. It is usually said in Game Theory that every individual must pursue its own (simple) goals for the global (more complex) goals to be fulfilled, and in this sense, yet again we get a synergy from learning to evolution.

As explained before, the trade-offs shown here are not exhaustive and, as Turney himself says, there may be some overlap in the terms. The list will tend to grow as new aspects of the Baldwin Effect are known, and new applications are found for it.

3.2.1 Hinton and Nowlan's experiment

Some recent work in Genetic Algorithms has been directed towards the analysis of the benefits of phenotypic plasticity, phenotypic rigidity and the plasticity of learning. Perhaps the first attempt made in this direction was performed by Hinton and Nowlan [10] as stated at the beginning of this chapter.

Their observations seem to imply that learning can facilitate evolution but these learned behaviors will eventually be replaced by instinctive behaviors if the environment remains constant during a relatively long time. An extremely simple neural-network[11] learning algorithm was created to model learning in a population. Every individual in the population codifies a candidate for solution to the neural network, thus a genetic algorithm played the role of evolution on the population of evolving individuals with learning capabilities.

In this simplified model, every individual consists of 20 potential connections among neurons. A connection can have one of three values: *present*, *absent*, and *learnable*; which are coded as 1, 0 and ? respectively, where each question mark can be set during learning to either 0 or 1. Then, the representation is a string of 20 values, so an individual is represented by $a_1 a_2 \ldots a_{20}$ where $a_i \in \{0, 1, ?\}$ for each $i \in \{1, 2, \ldots, 20\}$. There is only one correct setting of the neural network's connections (which, by simplicity is *all present*[12]), and no other setting confers any fitness to the individual. We will say that a connection is *fixed* if it is either 0 or 1, and that it is *not fixed* if it has a question mark.

The problem to be solved is to find this single correct set of connections. Is will not be possible for those networks that have incorrect fixed connections to find the

[11] Which is actually transparent to the process, so no prior knowledge about artificial neural networks is required to understand it.

[12] This means all the connections present, or, as an individual, a chromosome consisting of 20 ones.

solution, but those networks that have correct values in all fixed places, have the opportunity to learn the correct setting. In this experiment, the simplest learning method was used: *random guessing*. On each trial, an individual guesses 0 or 1 at random (uniform) on each question mark it possesses.

This problem is, by design, a *needle in a haystack* search problem, since there is only one correct setting out of the 2^{20} possibilities. The fitness landscape for this problem is schematically represented in Figure 3.1—the single spike represents the single correct connection setting. Introducing the ability to learn, as expected by the

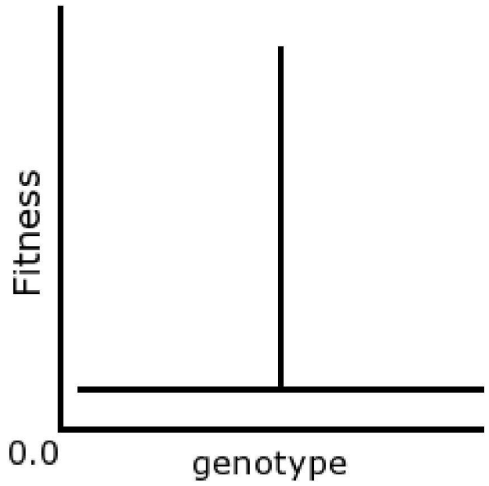

Figure 3.1: Schematic view of the fitness landscape for Hinton and Nowlan's search problem. All genotypes have fitness 0 except for the correct one with fitness 1.

Baldwin Effect, the landscape it smoother, and now we observe in Figure 3.2 a zone of increased fitness, meaning that there are individuals that can learn the correct setting and have a reward of fitness (inversely proportional to the number of trials). This zone includes individuals with only correct fixed positions and question marks. Once the individual is inside this zone, evolution makes it possible to climb to the peak.

The initial population consisted of 1000 individuals, each consisting of 20 genes, generated at random, with each gene having probability 0.25 of being 0, probability 0.25 of being 1, and probability 0.5 of question mark. At each generation, each individual was given 1000 learning trials. On each learning trial, the individual tried a random combination of settings for the question marks.

The fitness was calculated by the following formula,

$$Fitness = 1 + \frac{19(1000 - i)}{1000} \tag{3.1}$$

where i stands for the number of trial in which the individual guessed the correct setting of connections. The fitness is an inverse function of the number of trials needed

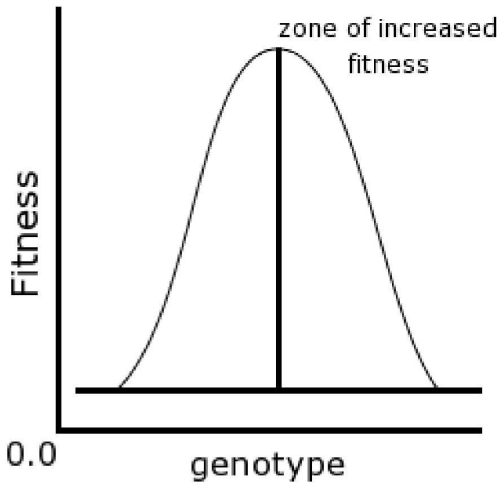

Figure 3.2: Schematic fitness landscape after learning.
The search problem is smoother with a zone of increased fitness containing individual able to learn the correct connection settings.

by an individual to find the correct solution. With this function, an individual with all its positions fixed and equal to 1, would get the maximum fitness value of 20, while an individual that was never able to correctly guess the solution or that has at least one wrong fixed position would get the minimum fitness value of 1.

In this experiment we can observe the trade-off of the Baldwin Effect as many question marks mean that, on average, many guesses are needed to arrive to the solution, but the more fixed positions, the more likely it is that at least one value is wrong thus effectively killing the individual. This trade-off depicts the one existing between efficiency and plasticity in a very straightforward way.

In expectation, an individual has half of its positions fixed in the initial population. The expected number of individuals in the initial population that have no wrong fixed position is about one (the 2^{10} possible values for half fixed positions are about 1000). In the ending, it is expected that at least one individual will be able to learn the correct settings, but this is no surprise because $1000 * 1000 = 10^6 \sim 2^{20}$, so this experiment could be considered invalid because of this analysis, however, it is an example of a simple experiment and the ability of the Baldwin Effect to smooth the fitness landscape, as it was stated by Mitchell [14] that the mean fitness was not observed to improve over generations in the case of pure evolution.

Hinton and Nowlan's genetic algorithm used to solve this problem was very similar to the simple genetic algorithm discussed in Section 2.2. The selection mechanism was by roulette wheel, with replacement. They used one-point crossover and simple mutation; also, the chromosome of the individual was obviously not affected by learning that took place during its lifetime. Originally, they let the algorithm run for 50 generations. They observed that 0 genes were rapidly eliminated from the population

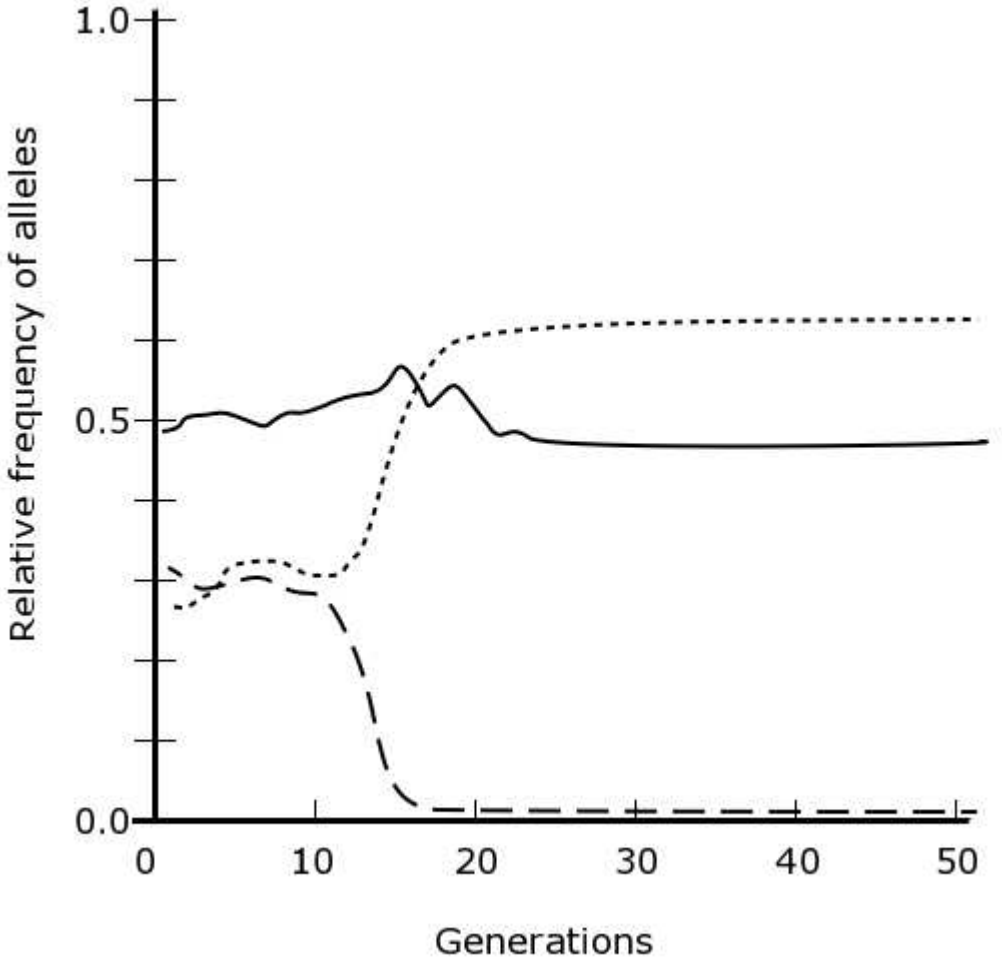

Figure 3.3: Relative frequencies of 1's (dotted), 0's (dashed) and undecided (solid) alleles in the population plotted over 50 generations.

and that the frequency of 1's increased accordingly. In Figure 3.3 we show the relative frequencies of correct (ones), incorrect (zeros) and undecided (question marks) alleles in the population plotted over 50 generations.

3.2.1.1 Harvey's experiment

The main concern resulting from the plot is why did the frequency of undecided alleles stays so high. With the frequency of question marks stable at 45%, and the frequency of 1's stable at 55%, an average individual with 20 genes would have eleven 1's and nine ?'s. A more detailed study of this experiment was performed by Harvey [9], and Belew [3], and according to them, the expected fitness of such an individual is roughly 11.6. Also, they performed an statistical analysis of the expected fitness of the algorithm if only evolution was allowed to search (i.e. not learning), and resulted at 1.009.

This points clearly to the first aspect of the Baldwin Effect, in which learning aided evolution to improve the expected fitness from 1.009 to 11.6, but this experiment, as it was made, did not say much about the evolution's preference of instinct over learning on the long term. To answer this question, Harvey [9] reproduced and augmented the original experiment in order to address the so-called *Puzzle of the persistent question marks*. In his work, he ran the model for 500 generations, and he observed that the frequency of question marks indeed decreased in time towards 0%. However, it did not matter how many generation he ran the model, that percentage never reached zero.

The reason seems to be the *genetic drift*, due to random mutation in the population. Mutation exerts a constant pressure that maintains a certain frequency of undecided alleles in the population, and eventually, the population will achieve an equilibrium state where the pressure of genetic drift balances with the selection pressure that favors instinct.

3.2.2 Turney's experiments

We will analyze now a model that is a bit closer to a more complete Baldwinian scenario. In his paper, Turney [21] used the Baldwin Effect as a method to shift the bias in a machine learning problem. His experiment is also simple as he argues that a more complex experiment would only obscure the role of Baldwinism in the optimization process. His work is of interest to us since he introduces a new type of coding for learning in the genotype. In order to understand his work, we will have to develop a few concepts.

3.2.2.1 Definition and types of bias

Excluding the input data, every factor that influences the selection of one particular concept (in machine learning) constitute the *bias* of a learning algorithm. Bias includes such factors as the language in which the learner expresses its concepts, the

algorithm used to search the space, and the criterion for deciding whether a concept is compatible with the training data.

As we saw at the beginning of Section 3.2, the bias consists of two factors: *direction* and *strength*. A correct bias is one that allows the concept learner to select the target concept. The correctness of the direction is thus measured by the performance of the learned concept on a test data. A strong bias is one that focuses the concept learner on a relatively small number of concepts.

3.2.2.2 Shift of bias

A growing body of research in machine learning is concerned with algorithms that shift bias as they acquire more experience. Shift of bias performs two levels of search, one through concept space and one through bias space.

We have seen that a strong bias is somewhat analogous to an instinctive behavior, while a weak bias is to a learned behavior. The cost of having a strong bias is that the bias can be incorrect, and the disadvantage of having a weak bias is a poor performance or efficiency on the long term. Unless we have high confidence that the bias is correct, it is in general risky to have a strong bias. All of this is in accordance to the Baldwin Effect, so it seems reasonable to incorporate it as a bias shifter.

3.2.2.3 The Baldwinian model

Turney generalized the experiment of Hinton and Nowlan and adapted it to the machine learning problem as a shift of bias problem. It may not be clear that a shift of bias was used in Hinton and Nowlan's experiment, but we might see the amount of question marks as the strength of the bias. Having many undecided alleles would result in a weak bias, while having just a few would result in a stronger bias. The plot of question marks' frequencies in the population can be regarded as the population's trajectory of search in bias space. For this new experiment, this distinction is made explicitly, and might be clarified better with the experiment itself.

Let us consider the example of concept learning. Suppose the examples to classify are all five-dimensional Boolean vectors $\vec{x} \in \{0,1\}^5$, and that they may belong to one of two classes $\{0,1\}$. By simplicity, let us call this space $T = \{0,1\}^5$. In this sense, the search space of concepts is the space of functions $F = \{f | f : T \to \{0,1\}\}$, mapping vectors to classes. To simplify the notation, we see that it is possible to identify each concept (i.e. each function in F) with its truth table. The truth table lists all of the $2^5 = 32$ possible vectors in lexicographical order, and the value of the function for each vector. As the vectors are in lexicographical order[13], we can implicitly assume the vectors in the truth table, and compactly write the associations of the function as a 32-*bit string*, with the i-th position in the string corresponding to the class of the i-th 5 -bit vector.

[13] Actually, any order may work.

For example, the function that maps every vector $\vec{x} \in T$ to 1, would be coded as the bit string consisting of 32 ones, and conversely; the binary string given by $\overline{11101111111101111111110111111111}$ represents the function that maps the vectors $\overline{00011}$, $\overline{01100}$ and $\overline{10110}$ [14] to the class 0, and the rest to the class 1. In this way, we have a total of $2^{32} = 4294967296$ possible functions, and thus, the amount of possible solutions to the classifier problem are also 2^{32}.

Suppose that one particular *target* concept is what we want to find, as was the case with Hinton and Nowlan's neural network. To facilitate comparison with Harvey [9], we will suppose that the target function is the function that classifies every vector to the class 1 (i.e., $f(\vec{x}) = 1$ for each $\vec{x} \in T$). We assume, also, the standard supervised learning paradigm, with a training phase followed by a testing phase.

During training, the learner is taught the class of each of the 32 possible input vectors. To make the problem interesting, we will assume there is a certain probability p that the learner is taught the wrong class. During test, the learner must guess the class of the supplied input vector. Again, there is a probability that the test is mistaken about the correct class for an input vector. That is, the probability p is the level of noise in the classifier.

We will use the next notation,

$$\begin{aligned} \text{target} = (t_1, t_2, \ldots, t_{32}) &= \vec{t} \\ \text{train} = (\alpha_1, \alpha_2, \ldots, \alpha_{32}) &= \vec{\alpha} \\ \text{test} = (\beta_1, \beta_2, \ldots, \beta_{32}) &= \vec{\beta} \end{aligned}$$

where $t_i, \alpha_i, \beta_i \in \{0, 1\}$. We generate $\vec{\alpha}$ and $\vec{\beta}$ from $\vec{t}$ by randomly flipping bits in $\vec{t}$ with probability p. The probability that the class of a training example or a testing example matches the target is $1-p$, but the probability that the class of the training example matches the class of the testing example is $1 - 2p + 2p^2$. Namely,

$$\begin{aligned} P(\alpha_i = t_i) &= 1 - p \\ P(\beta_i = t_i) &= 1 - p \\ P(\alpha_i = \beta_i) &= 1 - 2p + 2p^2 \end{aligned}$$

and we observe that either $\alpha_i = \beta_i = t_i$, with probability $(1-p)^2$, or $\alpha_i = \beta_i \neq t_i$, with probability p^2, which yields $(1-p)^2 + p^2 = 1 - 2p + 2p^2$. This model is very common in statistics, and can be thought as the observed class ($\vec{\alpha}$ or $\vec{\beta}$) being composed of a signal ($\vec{t}$) plus some random noise (p).

3.2.2.4 The algorithm

We will use a genetic algorithm to solve this example problem. Each genotype consists of 64 genes, 32 of which determine the bias direction, and 32 that determine the bias

[14] i.e. the 3rd, 12th and 22nd in the string.

strength. The bias direction genes are either 0 or 1, and represent the class to be proposed for that entry. The bias strength genes are real values in the interval $[0, 1]$, each one coded with 8 bits, as illustrated next

$$\begin{aligned} \text{genotype } G &= (D; S) \\ \text{bias direction } D &= (d_1, d_2, \ldots, d_{32}) \\ \text{bias strength } S &= (s_1, s_2, \ldots, s_{32}) \end{aligned}$$

where $d_i \in \{0, 1\}$, and $0 \leq s_i \leq 1$, and the use of the strength is as follows: if the i-th bias strength gene has a value s_i, then there is a probability s_i that the individual will guess d_i, similarly, there is a probability $1 - s_i$ that the individual will guess α_i.

The guess vector is expressed as

$$\begin{aligned} \text{guess } \vec{g} &= (g_1, g_2, \ldots, g_{32}) \\ P(g_i = d_i | d_i \neq \alpha_i) &= s_i \\ P(g_i = \alpha_i | d_i \neq \alpha_i) &= 1 - s_i \\ P(g_i = d_i = \alpha_i | d_i = \alpha_i) &= 1 \end{aligned}$$

which can be interpreted as, if the bias is weak (s_i close to zero), then the individual will guess based on what it observed in the training data (i.e. it guesses α_i); if the bias is strong (s_i close to one), the individual ignores the training and relies on instinct (i.e. it guesses d_i).

Turney [21] points out that his simplified model does not describe the learning mechanism. He also states that the level of abstraction used in this experiment is one in which the mechanism is not important, and in a more complex problem, the genotype could encode for example the architecture of a neural network, and a learning algorithm as back propagation could be used as a learn method.

In this experiment we could see a number of features of the Baldwin Effect, for instance, if an individual relies entirely on instinct (for each $i \in \{1, 2, \ldots, 32\}, s_i = 1$), and its instinct is correct, (for each $i \in \{1, 2, \ldots, 32\}, d_i = t_i$), then the probability that it will correctly classify all the 32 input vectors in the testing phase is $(1 - p)^{32}$; while if an individual relies entirely on learning (for each $i \in \{1, 2, \ldots, 32\}, s_i = 0$), then the probability that it will correctly classify all testing vectors is $(1 - 2p + 2p^2)^{32}$. Observe that with increasing noise level (p), the correct instinct has an advantage over pure learning. This is due to a small catch in the phrasing, as we require the instinct to be correct in advance.

For convenience, the fitness of the individuals will range from 0 to 1. As with Hinton and Nowlan, we will require the individuals to correctly guess the class of all 32 testing examples. We assign a fitness score of 0 when the guess does not perfectly match the testing data, and a score of 1 when the match is perfect[15].

[15]In contrast to Turney, who assign $(1 - p)^{-}32$ by an unknown reason.

In order to better understand what is going on in a run of the algorithm, we will measure the bias correctness and the bias strength as follows

$$\begin{aligned} \text{bias correctness} &= \frac{1}{32}\sum^{3} 2_{i=1}\,[d_i = t_i] \\ \text{bias strength} &= \frac{1}{32}\sum^{3} 2_{i=1} s_i \end{aligned}$$

where the bias correctness is represented by the frequency with which the bias direction matches the target, and bias strength is the average of the strengths s_i.

We can view the genotype in Hinton and Nowlan as a special case of Turney's genotype:

$$\begin{aligned} 0 &\Leftrightarrow d_i = 0, s_i = 1 \\ 1 &\Leftrightarrow d_i = 1, s_i = 1 \\ ? &\Leftrightarrow s_1 = 0, d_i \in \{0, 1\} \end{aligned}$$

In Hinton and Nowlan's genotype, the only way to increase bias strength is to change one or more question marks to a fixed number (either 0 or 1), and conversely to decrease it. In Turney's genotype, we can alter the bias strength without changing bias direction.

The Baldwin Effect predicts that, initially, when the bias correctness is low, selection pressure will favor weak bias. Later, when bias correctness is improving, selection pressure will favor a stronger bias.

3.2.2.5 Experiments

The algorithm was set to a genetic algorithm, with population of 1000, with a crossover probability of 0.6 and a mutation rate of 0.001. The algorithm was left to run for 10000 generations. Various parameters of p were used in the experiments, and in general, the behavior can be observed in Figure 3.4.

In each experiment, Turney plotted the average bias correctness in the population, bias strength, and fitness as a function of the generation number. He used a logarithmic scale in the generations to allow an improved visibility of the features of the Baldwin Effect, since the first aspect of the effect (selection for learning) tends to take place quite rapidly in the early generations, while the second aspect (selection for instinct) tends to take place much more slowly.

We can see this behavior in Figure 3.3, were, on the long run, we should expect the question marks to approach zero. The logarithmic scale was used to be able to see both behaviors in the same figure.

Turney performed a number of experiments modifying the bias strength in an external way, and allowing the evolution to adapt with those strength paths by itself (i.e. no Baldwin Effect was allowed). He concluded that, compared against a constant and a linear increment bias strength, the Baldwin Effect performed better. This points

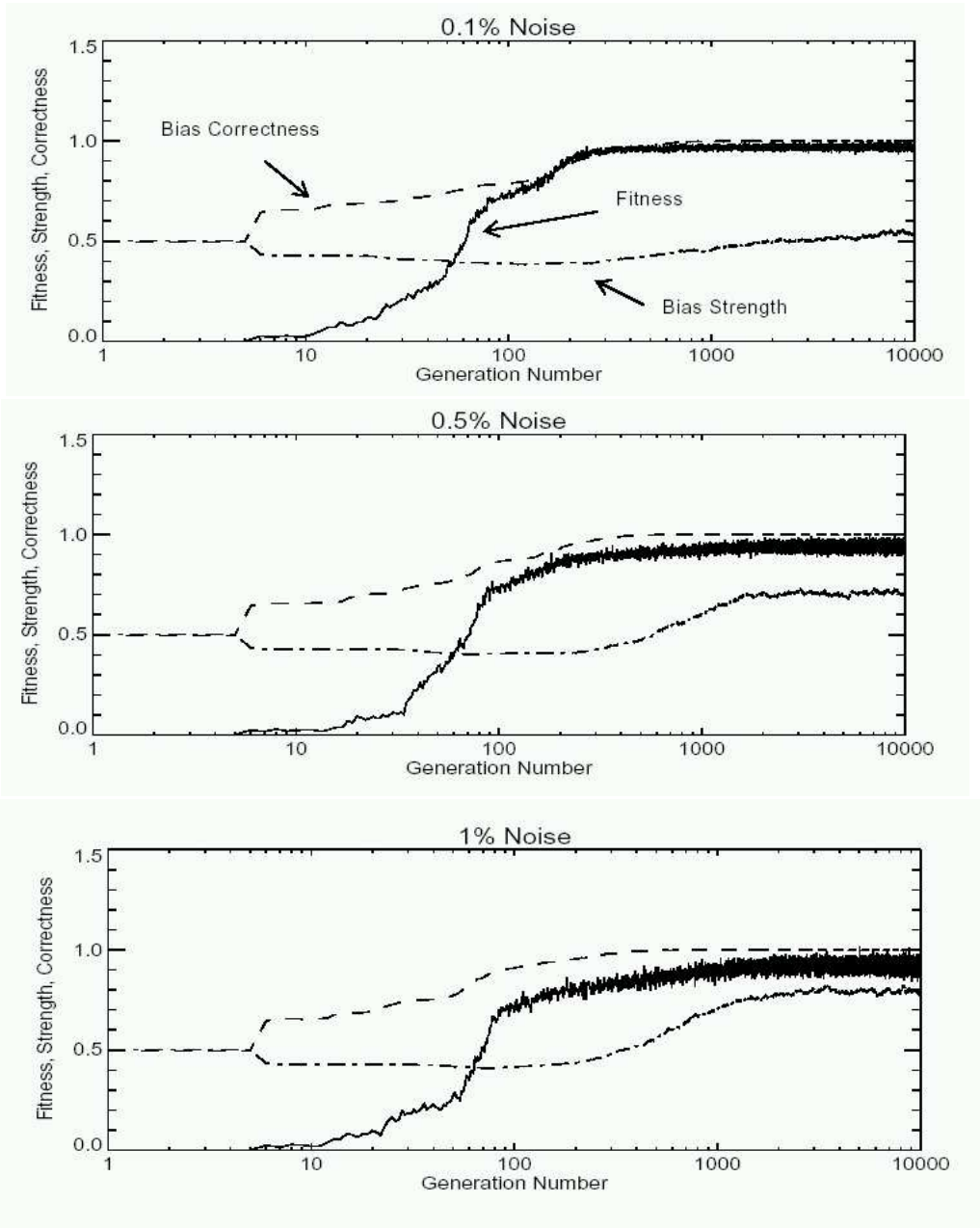

Figure 3.4: The average fitness, bias strength, and bias correctness of a population of 1000 individuals, plotted for generations 1 to 10000, with three noise levels.

to the quality of the path traversed in bias space by the Baldwin Effect. It is not demonstrated though, that the Baldwinian path is optimal, but at least it is a good one.

We present here a number of reproductions of the graphs obtained by Turney. The original plots were made for three p parameter values, however, for the sake of clarity, we will only present here the plots for $p = 0.5\%$. The rest of the figures are very similar.

Skewed strength Turney tested the robustness of the phenomena observed in the experiment. He deliberately skewed the first generation by assigning a random individual generator which favors a strong bias. The bias genes were generated so that there was a probability of 75% that $0.9 \leq s_i \leq 1$, a probability of 25% that $0.5 \leq s_i < 0.9$, and a probability of 5% that $0 \leq s_i < 0.5$.

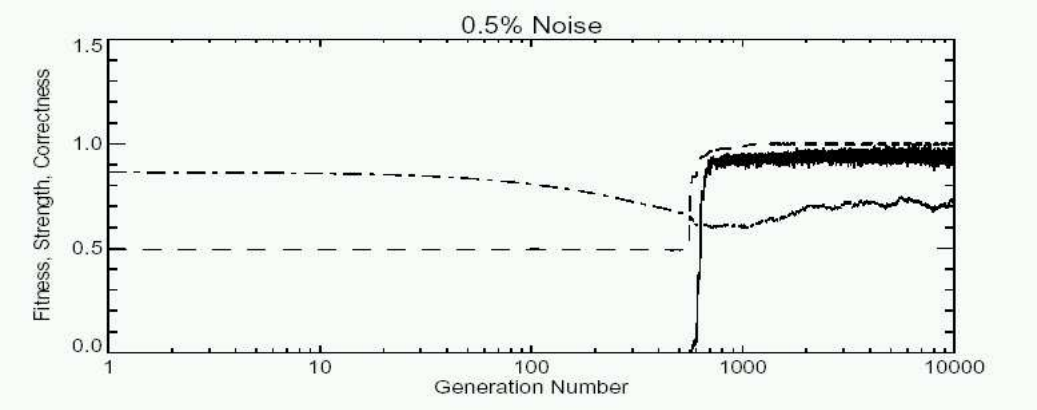

Figure 3.5: Experiment result for $p = 0.5$. The population is skewed towards stronger bias.

In Figure 3.5 we can see the results for the experiment. There are a number of remarks that can be done:

1. The population eventually settled into (approximately) the same equilibrium state that was observed in the first experiment.

2. The skewed bias strength slowed down the creation of the first individual with non zero fitness.

3. Once this individual is created, there is little difference among the experiments.

4. During the time for which all individuals have zero fitness, genetic drift pushes bias strength towards 0.5.

5. After the first *non-zero* individual is created, the strength still decreased for a small number of generations.

This observations are expected from the Baldwin Effect, as a result of the observations made on Table 3.1.

Forced bias strength Another experiment made, was the one concerning forced bias strength trajectories. The idea was to create a fixed trajectory on bias space by *a priori* setting the bias strength of the individuals as a function of the generation number.

The Baldwin Effect explained fairly well the behavior of the model created by Turney. However, it seems fair to compare it to some other trajectories forced upon the bias strength. In general, a non-Baldwinian algorithm will have a strength of zero. Turney considers some other possible trajectories to compare to.

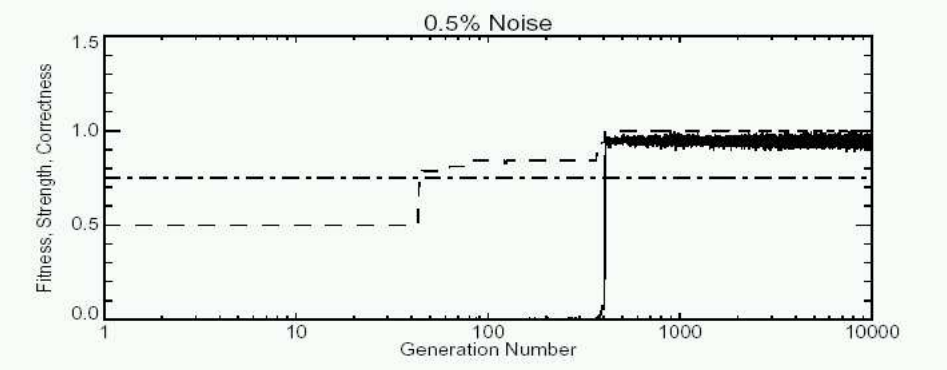

Figure 3.6: Bias strength fixed at 0.75.

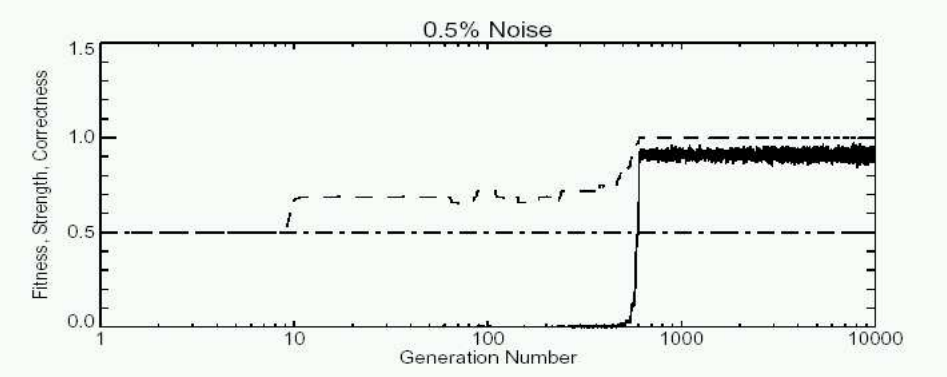

Figure 3.7: Bias strength fixed at 0.5.

By directly manipulating the bias strength, Turney compared the Baldwin Effect to 4 other trajectories:

Fixed 0.75 This experiment is plotted in Figure 3.6, and as expected from Baldwin Effect's aspects, this is the experiment with the longer wait until the first individual with non-zero fitness is found.

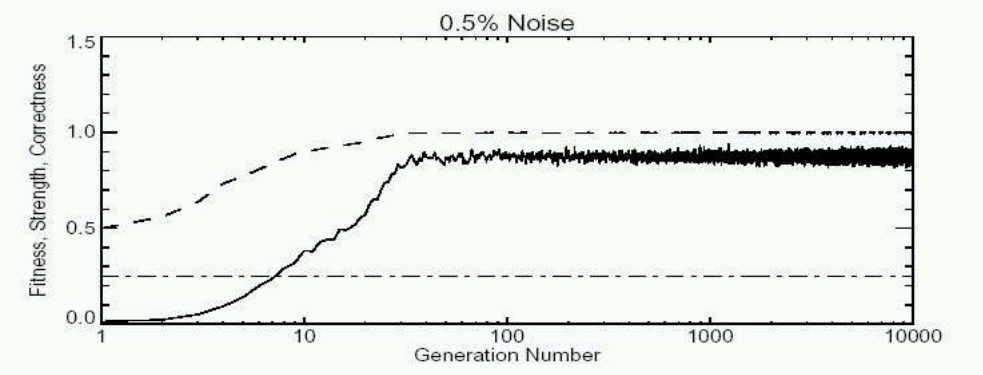

Figure 3.8: Bias strength fixed at 0.25.

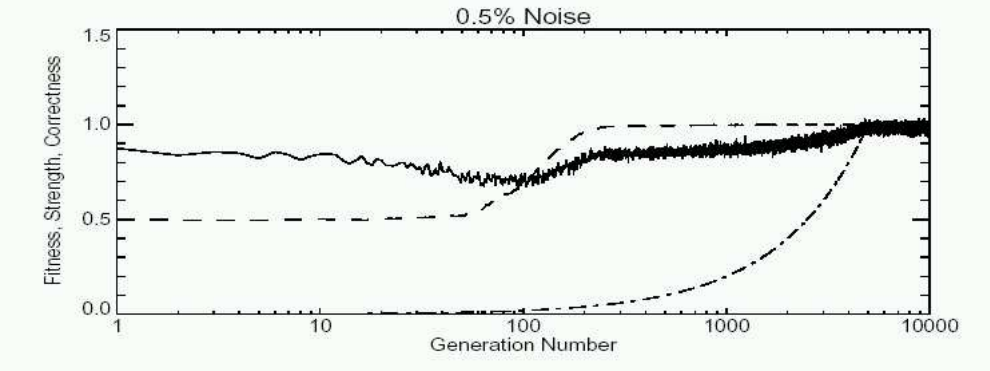

Figure 3.9: Bias strength increases linearly from 0 in the first generation to 1 in the generation 5000. Afterwards, the bias is held constant at 1.

Fixed 0.5 Plotted in Figure 3.7, it is just a middle case between the first and third trajectories.

Fixed 0.25 From the last three trajectories, this trajectory, Figure 3.8, finds the first non-zero individual faster. Its final fitness value will be the worst of all four trajectories.

Linear[16] As this experiment resembles more to the Baldwin Effect than the others, it is expected to outperform all others, but will be outperformed by the *true* Baldwinian. It is plotted in Figure 3.9.

Chapter 4

Baldwinian Optimization

In the previous Chapter we reviewed a number of experiments conducted in evolutionary computation created to improve our understanding of the Baldwin Effect. In this Chapter, we provide hints on where and how an algorithm could be turned Baldwinian, and give an adaptation for constrained optimization for two well-known algorithms.

The idea behind a Baldwinian algorithm is very similar to the memetic algorithm reviewed in Section 2.4, in the sense that learning is a local search. A Baldwinian algorithm is an evolutionary algorithm with an extra operator[1], whose main purpose is to perform a learning stage, in many ways similar to the local search stage performed by memetic algorithms.

There are two main differences between memetic algorithms and Baldwinian algorithms: first of all, the local search in memetic algorithms is performed on genotypic space, while in the Baldwinian case, it is performed in phenotypic space; second, the genotype of the individual is not changed by the local search in the Baldwinian algorithm in contrast to the memetic algorithms. This last point is mainly due to the intractability of the reverse mapping from phenotype to genotype we discussed in Section 3.1.3.

In this sense, we can think of memetic algorithms to be Lamarckian in nature. It has been stated, however, that the memes could behave more like a Baldwinian factor than a Lamarckian one [20]. In this thesis, however, we are not interested in discussing whereas this is actually true or not, and is left to the reader to come up with his own conclusions.

At every stage in the evolutionary algorithm where a local search can be performed, we can make a Baldwinian search (i.e. lifetime learning in contrast to genetic modification). Figure 4.1 gives a schematic representation of the learning process.

In many complex evolutionary algorithms there is a clear difference among the phenotype and the genotype. It is crucial to take into account that learning takes place in phenotypic space. Although some simpler algorithms don't make the difference

[1]It has been proposed that the learning should substitute the mutation in evolutionary strategies, but we consider it to be an additional operator.

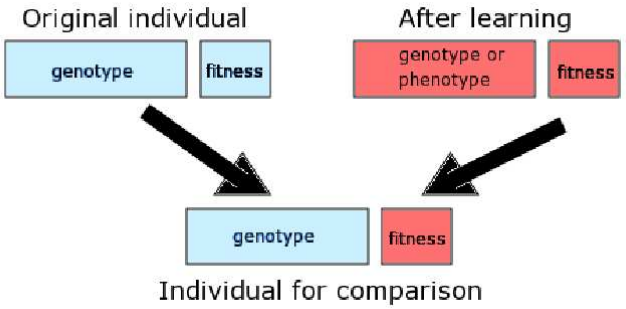

Figure 4.1: Schematic representation of the Baldwinian implementation for learning. The upper left individual is the original individual before learning. Then, at the upper right corner, the individual after learning with modified fitness and/or genotype. Finally, at the bottom, the individual as is to be compared with other individuals. Observe that it retains its original genome, and only the fitness is changed.

between phenotype and genotype, or the coding is straightforward, we must try our best to state the difference as clearly as possible in order to better introduce the Baldwinian concept in the algorithms.

If any representation is being made to create the phenotype, this representation is to be used as the basis for learning. This might be straightforward in genetic algorithms, but could be less than clear in evolutionary strategies. In the following section we will try to introduce the learning operator to two of the most well-known algorithms; and give general ideas on where can any other algorithm be transformed into a Baldwinian algorithm.

4.1 The Learning Operator

Every evolutionary algorithm has a number of evolutionary operators associated with it. The exact type of these operators and the order in which they are applied to the population is what defines the algorithm itself. The reader should be familiar with the concepts developed in Section 2 before he attempts to read this chapter.

Virtually every evolutionary algorithm has a mutation operator associated[2]. This operator will serve as the basis for the learning operator in the Baldwinian version of the algorithm. The basic idea is to create a loop of *mutation-like* local variations, each time the individual is allowed to learn.

The scheme of the algorithm is as follows:

Baldwinian Algorithm

`initialize-population` P_0;

[2]Or, at least, a mutation step inside an operator.

```
Let i = 0;
while( termination-criteria-is-not-met )
{
    P_f = O( P_i, rand() );
    F_{B_i} = L( P_i, rand() );
    F_{B_f} = L( P_f, rand() );
    P_{i+1} = σ( P_i, P_f, F_{B_i}, F_{B_f}, rand() );
    i = i + 1;
}
```

In the algorithm, we note the function L which represents the learning step of the algorithm. It is important to note that the selection ($P_{k+1} = \sigma(P_i, P_f, F_{B_i}, F_{B_f})$) is performed over the same individuals with the adjusted fitness values.

We will use the algorithm developed by Turney in Section 3.2.2.4 as a model to create our own algorithm. Mainly, the idea of including the bias strength in the genotype, but an independent part of it will be used. This bias strength will be changed in concept to best fit the learning concept of search instead of machine learning.

We will use the term *instinct strength* as an analogous to Turney's bias strength in the sense that it measures the probability that the individual may follow instinct instead of learning. This strengths are going to be introduced in the genotype in a way that resembles the introduction of the control values σ_i in the evolutionary strategy (Section 2.3).

The general form of the learning operator is sketched in the next algorithm:

Learning Operator

```
function L( population P, real r )
{
    F =vector[ sizeof(P) ];
    x =getRandomValue( r );
    for( i = 0 to sizeof(P) )
    {
        if( strength(p_i) < x )
            F_i =Baldwinian(p_i);
        else
            F_i = f ∘ τ(p_i);
        x = getRandomValue( r );
    }
    return F;
}
```

Observe that the main part of this operator is in the function called *Baldwinian*. This function returns the Baldwinian fitness associated to the individual, which is problem

dependant. It will usually be the result of a local search. Note that the original individual is not changed as only a number (the Baldwinian fitness) is associated to its position. The selection method will only be interested in this number to either select the individual or not.

4.2 Baldwinian Algorithms

In this section we will provide the examples of Baldwinian algorithms developed as the main contribution of this thesis. We will define the learning operators[3] used, and will compare the results with the non-Baldwinian version of the same algorithm to place them into an *equal-rights* state. The parameters of the algorithms will be set to the same values and we will report a number of statistical values over 30 runs for every problem to be solved. In each case, the algorithm was left to run until 350000 evaluations of the fitness function were performed. This is accordance to the experiments made by Runarsson [17] in the same benchmark. This was done to allow a comparison between this results and those obtained by him. The best known or optimal solutions to the benchmark functions are in Table 4.1.

Function	Optimum known	max/min
$g01$	-15	Minimize
$g02$	0.803619	Maximize
$g03$	1	Maximize
$g04$	-30665.539	Minimize
$g05$	5126.4981	Minimize
$g06$	-6961.81388	Minimize
$g07$	24.3062091	Minimize
$g08$	0.095825	Minimize
$g09$	680.6300573	Minimize
$g10$	7049.3307	Minimize
$g11$	0.75	Minimize
$g12$	1	Maximize
$g13$	0.0539498	Minimize
$i1$	1.724852309	Minimize
$i2$	6059.71434795	Minimize
$i3$	0.012665	Minimize

Table 4.1: The known or reported optimum values for the test functions. The column *max/min* tells whether the problem is a maximization or a minimization to better interpret the results.

[3] In particular, the implementation of the *Baldwinian* function to calculate the Baldwinian fitness of individuals.

The test functions are of constrained optimization, and they can be found in the appendix. For a detailed explanation on the functions, the reader should check [17, 13].

4.2.1 Baldwinian evolutionary strategy

The classical evolutionary strategy $ES(\mu, \lambda)$ with self-adaptation parameters (σ_i), reviewed in Section 2.3, with a technique of rules with total sum of violations seen in Section 1.3.2.2 will be used.

The genotype will be augmented with the values of strength, so that it will be

$$p = (x_1, x_2, \ldots, x_l; \sigma_1, \sigma_2, \ldots, \sigma_l; s_1, s_2, \ldots, s_l)$$

where $0 \leq s_i \leq 1$ for every $1 \leq i \leq l$, and they represent the strength of instinct in the objective i. We will use $\vec{x} = (x_1, x_2, \ldots, x_l)$ to denote the objective portion of the genotype, $\vec{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_l)$ to denote the control portion, and $\vec{s} = (s_1, s_2, \ldots, s_l)$ to denote the strength portion.

This new evolution strategy will be compared with the strategy discussed in Section 2.3.4, and it will have the same parameter setting, except for the added strength portion.

The idea behind the learning operator is to use the same local search introduced by the σ's in the learning step. This is to avoid the appearance of unnecessary parameters in the algorithm.

The crossover operator used in the objective values will be intermediate-generalized, while in the control values and strengths will be discrete (see Section 2.3.3). The mutation will be as usual for the self-adaptive evolutionary strategy for objective and control values, and the strength will be mutated as follows

$$s_i^m = \max\{0, \min\{s_i + \text{Normal}(0,1), 1\}\}$$

i.e. the strength will be added a standard normal value, cropped to $[0, 1]$. The function

$$\rho(x) = \sum_{i=1}^{n} g_i^+(x) + \sum_{i=1}^{m} h_j^+(x)$$

represents the total sum of violations of $x \in S$, with all the weights equal to 1.

In order to calculate the Baldwinian fitness value of an individual, we will use the following algorithm

Baldwinian fitness

$\vec{x}_B$ = $\vec{x}$;
for($i = 1$ to l)
    if($s_i <$rand())
    {

$\vec{x}_- = \vec{x}_B - (0, 0, \ldots, \sigma_i, \ldots, 0)$; //at the i-th position.
$\vec{x}_+ = \vec{x}_B + (0, 0, \ldots, \sigma_i, \ldots, 0)$; //at the i-th position.
$\vec{x}_B = \arg\min\{\rho \circ \tau(\vec{x}_B), \rho \circ \tau(\vec{x}_-), \rho \circ \tau(\vec{x}_+)\}$;
}
setFitness($\vec{x}$, $f \circ \tau(\vec{x}_B)$);

As we can observe, the learning can take place in every objective value, or in none. It all depends on the values of the learning strength. Here, in contrast to Turney, we are not interested in the evolutionary process affecting the strengths, or whether the strengths follow the path predicted by the Baldwin Effect; that is left for a future work. Instead we are interested in whether this Baldwinian learning aids the optimization process or not.

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	14.999983	15.643469
worst	12.884572	13.029993
mean	14.524280	15.0092431
median	14.999897	15.296720
variance	0.564467	0.404876
standard deviation	0.751310	0.636299
# feasibles	30	21*
# ϵ-feasibles	30	30

Table 4.2: Results for function $g01$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	0.403915	0.416605
worst	0.295487	0.284842
mean	0.360394	0.358947
median	0.366991	0.3610752
variance	6.5839E-4	0.001052
standard deviation	0.025659	0.032445
# feasibles	30	13*
# ϵ-feasibles	30	30

Table 4.3: Results for function $g02$

The results for the benchmark functions are summarized in Tables 4.2–4.14. The results for the engineering problems are in Tables 4.15–4.17.

It is important to explain the apparently lower number of true feasible solution found by the algorithms. First of all, when the problem has equality constraints, it is impossible, due to a discretization error, to achieve the actual equality. Instead, every solution is ϵ-feasible, with an $\epsilon \sim 10^{-5}$. For the rest of the problems, the status

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	0.999759	1.002580
worst	0.963843	0.99873
mean	0.996605	1.000645
median	0.998753	1.000156
variance	4.14449E-5	1.34930E-6
standard deviation	0.006438	0.0011659
# feasibles	0	0
# ϵ-feasibles	30	30

Table 4.4: Results for function $g03$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	-30573.688537	-30684.810456
worst	-30298.460286	-30364.822278
mean	-30414.804487	-30593.128911
median	-30413.755796	-30663.190301
variance	-3372.159939	8785.872779
standard deviation	58.070301	93.732986
# feasibles	30	1*
# ϵ-feasibles	30	30

Table 4.5: Results for function $g04$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	5126.50995	5126.51885
worst	5285.52789	5263.58777
mean	5191.10762	5177.33432
median	5186.64882	5168.70377
variance	1706.38217	1241.9597
standard deviation	41.30838	35.24145
# feasibles	0	0
# ϵ-feasibles	30	30

Table 4.6: Results for function $g05$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	-6961.81382	-6961.816175
worst	-1206.19807	-6961.813383
mean	-6576.84885	-6961.813834
median	-6961.21074	-6961.813767
variance	2046571.365351	2.36233E-7
standard deviation	1430.58427	4.86038E-4
# feasibles	30	13*
# ϵ-feasibles	30	30

Table 4.7: Results for function $g06$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	24.876975	24.584865
worst	28.592849	29.549636
mean	26.196125	25.654459
median	25.924144	25.163657
variance	0.866379	1.233E-7
standard deviation	0.930795	1.110668
# feasibles	30	14*
# ϵ-feasibles	30	30

Table 4.8: Results for function $g07$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	0.095825	0.095825
worst	0.004505	0.013637
mean	0.068281	0.059710
median	0.095825	0.065143
variance	0.001185	0.001188
standard deviation	0.034431	0.034478
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.9: Results for function $g08$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	680.656456	677.608343
worst	685.906514	680.676598
mean	681.738274	680.465658
median	681.149711	680.636904
variance	1.4761804	0.3083865
standard deviation	1.2149816	0.555325
# feasibles	30	4*
# ϵ-feasibles	30	30

Table 4.10: Results for function $g09$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	14743.296345	12903.07097
worst	20334.78599	19457.0973
mean	16775.12296	16503.5519
median	16668.745345	16354.4338
variance	1094703.55833	2162402.151
standard deviation	1046.280821	1470.5108
# feasibles	1	2*
# ϵ-feasibles	30	30

Table 4.11: Results for function $g10$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	0.749955	0.731125
worst	0.751152	0.750204
mean	0.750309	0.743945
median	0.750214	0.747179
variance	8.3655E-8	4.5196E-5
standard deviation	2.8923E-4	0.006723
# feasibles	0	0
# ϵ-feasibles	30	30

Table 4.12: Results for function $g11$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	1.0	1.0
worst	0.939999	0.9699999
mean	0.986333	0.9903333
median	0.99000	0.990000
variance	1.2322E-4	6.9888E-5
standard deviation	0.011100	0.008359
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.13: Results for function $g12$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	0.624860	0.4664885
worst	0.99990	0.9999465
mean	0.91796	0.8497140
median	0.989343	0.899186
variance	0.013152	0.017600
standard deviation	0.114683	0.132667
# feasibles	0	0
# ϵ-feasibles	30	30

Table 4.14: Results for function $g13$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	1.836054	1.724852
worst	2.384537	2.574196
mean	2.050582	1.974465
median	2.006084	1.924799
variance	0.016603	0.03675
standard deviation	0.128856	0.19172
# feasibles	30	15*
# ϵ-feasibles	30	30

Table 4.15: Results for function $i1$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	6488.3890	6890.85390
worst	16783.24940	13053.0751
mean	10811.4691	10113.5241
median	10796.35730	9850.33251
variance	4984724.191	3011815.7516
standard deviation	2232.6495	1735.4583
# feasibles	30	13*
# ϵ-feasibles	30	30

Table 4.16: Results for function $i2$

Statistic value	Normal $ES(\mu + \lambda)$	Baldwinian $ES(\mu + \lambda)$
best	0.012704	0.0124919
worst	0.013231	0.0130756
mean	0.012875	0.0128605
median	0.012838	0.0128425
variance	1.5581E-8	1.3777E-8
standard deviation	1.2482E-4	1.1737E-4
# feasibles	30	20*
# ϵ-feasibles	30	30

Table 4.17: Results for function $i3$

of ϵ-feasible is changing; it is about 10^{-6} times the maximum achieved absolute value of the fitness function.

In the ending, it might seem that the Baldwinian algorithm fails to reach feasible solution in almost every problem, but this is just a misinterpretation of the results. As the best individual is often one who has learned (i.e. has an increased fitness value due to learning), but the genotype remains unchanged, it is fairly difficult to know for sure what is its Baldwinian violation by just looking at the genes. The numbers presented in the tables are only the individual's *genetical* violations, not the *actual* best violations. In order to obtain that value of fitness, the individual had a violation of effectively 0 after learning, making him feasible in its Baldwinian value[4].

Under the light shed by the last observation, we are safe to assure that the Baldwinian version of the algorithm outperforms, in general, the non-Baldwinian version. And actually, it performed fairly well for such a simple algorithm used on well-known difficult optimization problems.

In the next section we will introduce a Baldwinian version of a more powerful evolutionary algorithm.

4.2.2 Baldwinian Differential Evolution

The differential evolution algorithm $DE\_1(CR, F)$ reviewed in Section 2.5 , with a technique of rules with total sum of violations seen in Section 1.3.2.2 will be used.

The genotype will be augmented with the value of strength, so that it will be

$$p = (x_1, x_2, \ldots, x_l; s)$$

where $0 \leq s \leq 1$, and it represents the strength of instinct. If the individual is to learn, it will have MAX attempts to improve its constraint vector from a local variation on the F parameter. Usually, the value of MAX is set to 2, but various tries pointed to the good robustness of this parameter.

This new differential evolution will be compared with the differential evolution discussed in Section 2.5.4, and it will have the same parameter setting, except for the added strength portion.

The idea behind the learning operator is to use the local search with the parameter F in the learning step, as a solution with values near the produced individual is likely to have similar values in the difference part of the process.

As in the last section, we will use $\vec{x} = (x_1, x_2, \ldots, x_l)$ to denote the objective portion of the genotype.

The crossover operator used is the same than in the normal algorithm. The strength of the created vector will be set to the parent's value, plus a normal random number with standard deviation 0.1, with a probability of C, otherwise it is set to $0.9s$. The value of C can be used to control the increasing ratio of the strength.

The creation of a new individual changes a bit in this algorithm, but it is essentially the same as the original differential evolution. Assume we are creating the offspring

[4] That is the reason for the asterisk at the tables' # *feasibles* row.

of individual i in the population P, i.e. $\vec{x}$ is the objective part of the individual p_i, and s is the strength part.

Baldwinian comparison

```
x⃗_off = createOffspring(F);
x⃗_B = x⃗_off;
if( s < rand() )
    for( i = 1 to MAX )
    {
        x⃗_temp = createOffspring(F+Normal(0,0.1);
        if( ρ∘τ(x⃗_B) > ρ∘τ(x⃗_temp) )
            x⃗_B = x⃗_temp;
    }
if(better( x⃗_B, x⃗ ))
    if( 0.9 < rand() )
        p_next,i = ( x⃗_off, 0.9s );
    else
        p_next,i = ( x⃗_off, rand() );
else
    p_next,i = p_i;
```

As we can observe, the learning can take place MAX times or 0 times. As with the case of the evolutionary strategy, it depends on the values of the learning strength.

Again, the individual is not modified; observe that the offspring $\vec{x}_{off}$ is assigned to the next generation if the Baldwinian individual is better than the parent individual's part $\vec{x}$.

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	-15	-15
worst	-15	-15
mean	-15	-15
median	-15	-15
variance	0	0
standard deviation	0	0
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.18: Results for function $g01$

The results for the benchmark functions are summarized in Tables 4.18–4.30, and the results for the engineering problems are in Tables 4.31–4.33.

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	0.8036189	0.8036189
worst	0.8029367	0.8014754
mean	0.8035043	0.8032154
median	0.8036163	0.8036028
variance	3.45886E-8	3.67254E-7
standard deviation	1.8598E-4	6.06014E-4
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.19: Results for function $g02$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	1	1
worst	0.9999873	0.9998754
mean	0.9999990	0.9999846
median	0.9999999	0.9999998
variance	5.47056E-12	1.0087E-9
standard deviation	2.33892E-6	3.17606E-5
# feasibles	0	0
# ϵ-feasibles	30	30

Table 4.20: Results for function $g03$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	30665.538671	30665.538671
worst	30665.538671	30665.538671
mean	30665.538671	30665.538671
median	30665.538671	30665.538671
variance	1.2837E-22	1.4999E-22
standard deviation	1.1330E-11	1.2247E-11
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.21: Results for function $g04$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	5126.49686	5126.498109
worst	5126.49839	5126.522733
mean	5126.49784	5126.501413
median	5126.498106	5126.498126
variance	1.9935E-7	3.4270E-5
standard deviation	4.4649E-4	0.005854
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.22: Results for function $g05$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	-6961.813875	-6961.813875
worst	-6961.813875	-6961.813875
mean	-6961.813875	-6961.813875
median	-6961.813875	-6961.813875
variance	3.3087E-24	3.3087E-24
standard deviation	1.8189E-12	1.8189E-12
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.23: Results for function $g06$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	24.306209	24.306209
worst	24.306643	24.313465
mean	24.306327	24.307553
median	24.306209	24.30620
variance	2.8891E-8	4.8469E-6
standard deviation	1.6997E-4	0.0022015
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.24: Results for function $g07$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	0.09582	0.095825
worst	0.09582	0.095825
mean	0.09582	0.095825
median	0.09582	0.095825
variance	5.6493E-34	4.0445E-34
standard deviation	2.3768E-17	2.0110E-17
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.25: Results for function $g08$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	680.630057	680.630057
worst	680.630057	680.630057
mean	680.630057	680.630057
median	680.630057	680.630057
variance	1.7879E-25	2.4987E-25
standard deviation	4.2283E-13	4.9987E-13
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.26: Results for function $g09$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	7049.248020	7049.248020
worst	7049.260829	7049.359981
mean	7049.250584	7049.270574
median	7049.248020	7049.24802
variance	1.6376E-5	2.29355E-4
standard deviation	0.004046	0.01514
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.27: Results for function $g10$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	0.749904	0.750000
worst	0.750028	0.750813
mean	0.749982	0.750163
median	0.749999	0.750001
variance	1.1238E-9	6.0038E-8
standard deviation	3.3524E-5	2.4502E-4
# feasibles	0	0
# ϵ-feasibles	30	30

Table 4.28: Results for function $g11$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	0.98	1.0
worst	0.59	0.520018
mean	0.852052	0.775251
median	0.865	0.784590
variance	0.007546	0.012312
standard deviation	0.086868	0.110963
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.29: Results for function $g12$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	0.053943	0.053949
worst	0.438851	0.73930
mean	0.272058	0.35564
median	0.438829	0.43885
variance	0.036378	0.034533
standard deviation	0.190731	0.18583
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.30: Results for function $g13$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	1.724852	1.724852
worst	1.724852	1.724852
mean	1.724852	1.724852
median	1.724852	1.724852
variance	1.2325E-30	1.2325E-30
standard deviation	1.1102E-15	1.1102E-15
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.31: Results for function $i1$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	6059.774753	6059.774753
worst	6059.774753	6059.775651
mean	6059.774753	6059.77481
median	6059.774753	6059.77475
variance	3.3087E-24	2.9341E-8
standard deviation	1.8189E-12	1.7129E-4
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.32: Results for function $i2$

Statistic value	Normal $DE(0.9, 0.9)$	Baldwinian $DE(0.9, 0.9)$
best	0.012665	0.012665
worst	0.012665	0.012665
mean	0.012665	0.012665
median	0.012665	0.012665
variance	3.5910E-35	3.3202E-35
standard deviation	5.9925E-18	5.7621E-18
# feasibles	30	30
# ϵ-feasibles	30	30

Table 4.33: Results for function $i3$

4.3 Conclusions on the Experiments

As we might see in the tables, the performance of the Baldwinian version of well-known optimization algorithms is fairly better or equal than the non-Baldwinian counterpart. As expected, the variance is greater, but the best value is usually better or equal to the one obtained by the normal version.

Our main observation in the tables is that, when the problem is a difficult one, the Baldwinian version outperforms, on average, the normal version of the algorithm; whereas this means that the Baldwinian algorithm is better[5] than the normal one or not remains unknown as the variance is usually greater in the Baldwinian case. At least this behavior proves what we expected from the observations on the Baldwin Effect in Chapter 3.

The learning operator for both cases is very simple as it was only used to illustrate the effects that learning can have on evolution. Better learning operators will lead to better results, but, as the Baldwin Effect teaches us, we must exercise caution when using learning because the computation time expend in learning is time lost from evolution.

All the experiments performed until now studied the Baldwinian algorithms to see whether the Baldwin Effect was present or not in the evolution–learning interactions. What we wanted to measure was the strengths of Baldwinian Algorithms, and if they are worth the try.

In order to see a full Baldwinian behavior on evolutionary algorithms, a huge amount of computational power was spent in order to better understand its effects. As we can see in the experiments performed by Hinton, Nowlan, Belew and Turney, the longer we let the algorithm run, the better the results we obtain are.

In contrast, we wanted to see if Baldwinian optimization can be applied to a problem with limited computational resources (as are 350000 evaluations of the fitness function) and still succeed in the optimization process by obtaining respectable solutions.

[5] In the sense of statistical robustness and behavior.

Conclusions

It is undeniable that more and more researchers are being attracted by the offerings of new hybridization techniques. Nature has always been a source of inspiration to man-kind, and we can clearly see this in the development of biologically inspired algorithms.

The Baldwin Effect might be a not-well-understood force in evolution, or can be just a biological curiosity. Either case, we can exploit it to be of use to evolutionary computation. Early experiments pointed to the strength of learning by solving problems of the type *needle in a hay stack* which are well known difficult optimization problems. The catch seems to be in learning and the way it was implemented. Learning is costly, and the experiments were more concerned with idealized algorithms with virtually unlimited computation resources.

In this thesis we wanted to issue the performance problem derived from learning. We compared the algorithm Baldwinian algorithm with the non-Baldwinian version of it, and the results are presented. Whether the Baldwinian version is better or not is something that we are not directly interested in. Instead we wanted to verify if it was possible to create a competitive algorithm based on the concepts from the Baldwin Effect.

Fortunately, most results were expected, and the issue of *better* is not easy to address with high variance results as obtained. However, learning was expected to increase the variance of results, and in general, the Baldwinian algorithm demonstrated an excellent better result, fairly good mean and median, and slightly large standard deviation.

We see the Baldwinian algorithms as a promising area of research, and expect the ideas to spread in the computing community. A good example of this can be seen in the birth of memetic algorithms, which resemble Baldwinian ones to the point in which many people even think they are the same.

In addition, if the concepts of Lamarckism have been used as valid computer models (although not biologically accurate) for optimization, using Baldwinian models is certainly as valid as Lamarckian. In the end, we can exploit more the Baldwinian concepts as are susceptible to be further studied in biology and, in consequence, better understood by computer scientists.

Bibliography

[1] Bäck. T., Hammel, U. and Schwefel, H.-P., (1997). Evolutionary Computation: comments on the history and current state. *IEEE Trans. on Evo. Comp.* 1 (1): 3-17.

[2] Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist* 30: 441-451, 536-553.

[3] Belew, R. K. (1990). Evolution, learning and culture. Computational metaphors for adaptive algorithms. *Complex Systems 4*: 11-49.

[4] Buckles, B. P., Coello, C. A. and Hernández, A. (1998). Estrategias evolutivas: La versión alemana del algoritmo genético. (I & II). *Soluciones Avanzadas. Tecnologías de Información y Estrategias de Negocios.* Año 6, (62), 38-45.

[5] Coello, C. and Mezura, E. (2004). What makes a constrained optimization problem difficult to solve. *Evolutionary Computation Group at CINVESTAV*, D.F., Mexico.

[6] Dawkins, R. (1976). The selfish gene. *Oxford: Oxford University Press.*

[7] Eiben, A. E. and Smith, J. E., (2003), Introduction to evolutionary computing. *Natural computing series. Springer.* (10): 173-180.

[8] Fogel, L. J., (1962). Autonomous automata, *Industrial Research* 4: 14-19.

[9] Harvey, I. (1993). The puzzle of the persistent question marks: A case study of genetic drift. *Proceedings of the 5th International Conference on GA.* Morgan Kaufmann

[10] Hinton, G. E., and Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems,* 1, 495-502.

[11] Holland, J. H., (1975). Adaptation in natural and artificial systems. *The University of Michigan Press*, Ann Abor, MI.

[12] Koza, J. R., (1992). Genetic Programming: On the programming of computers by means of natural selection. *MIT Press.*

[13] Koziel, S., and Michalewicz, Z., (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1), 19-44.

[14] Mitchell, M. (1998). An introduction to Genetic Algorithms. *The MIT Press* (3), 87-95.

[15] Price, K. and Storn, R. (1996). Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical Report* TR-95-012, ICSI.

[16] Rechenberg, I., (1965). Cybernetic solution path of an experimental problem. *Journal of the ACM* 3, 297-314.

[17] Runarsson, T. P., and Yao, X. (2000). Stochastic Ranking for constrained evolutionary optimization. *IEEE Trans. on Evo. Comp.*, TEC#311R.

[18] Schwefel, H.-P., (1968). Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasenädse, Teil I. *Technischer Bericht* 11.034/68, 35.

[19] Simpson, G. G. (1956). The Baldwin Effect. *Evolution* 7, 110-117.

[20] Turney, P. (1996). Myths and legends of the Baldwin Effect. *Evo. Comp. and Machine Learning* (ICML-96), 135-142. NRC-39220.

[21] Turney, P. (1996). How to shift Bias: Lessons from the Baldwin Effect. *National Research Council Canada*, Institute for IT. K1A 0R6.

[22] Waddington, C. H. (1942). Canalization of development and the inheritance of acquired characters. *Nature* 150, 563-565.

Appendix A

Benchmark functions

Here we present the test bed used to compare the algorithms in this thesis. The benchmark functions $g01$ to $g12$ were put together by Michalewicz and Koziel, and are described in [13]. The function $g13$ was proposed by Runarsson and Yao in [17]. The engineering problems $i1$, $i2$ and $i3$ are described in [???]. Another proposed engineering (thought to be very hard) problems, here referred as $c01$ to $c08$, were proposed by Mezura and Coello in [5]. Only for the sake of completeness, all the functions are reproduced here.

1. $g01$
   Minimize:
   $$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$
   subject to:
   $$\begin{aligned} g_1(\vec{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 &\leq 0 \\ g_2(\vec{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 &\leq 0 \\ g_3(\vec{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 &\leq 0 \\ g_4(\vec{x}) = -8x_1 + x_{10} &\leq 0 \\ g_5(\vec{x}) = -8x_2 + x_{11} &\leq 0 \\ g_6(\vec{x}) = -8x_3 + x_{12} &\leq 0 \\ g_7(\vec{x}) = -2x_4 - x_5 + x_{10} &\leq 0 \\ g_8(\vec{x}) = -2x_6 - x_7 + x_{11} &\leq 0 \\ g_9(\vec{x}) = -2x_8 - x_9 + x_{12} &\leq 0 \end{aligned}$$
   where the bounds are $0 \leq x_i \leq 1$ $(i = 1, 2, \ldots, 9)$, $0 \leq x_i \leq 100$ $(i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. The global minimum is at $\vec{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ where six constraints are active (g_1, g_2, g_3, g_7, g_8 and g_9), and $f(\vec{x}^*) = -15$.

2. $g02$
Maximize:

$$f(\vec{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = 0.75 - \prod_{i=1}^{n} x_i &\leq 0 \\ g_2(\vec{x}) = \sum_{i=1}^{n} x_i - 7.5n &\leq 0 \end{aligned}$$

where $n = 20$, the bounds are $0 \leq x_i \leq 10$ $(i = 1, 2, \ldots, n)$. The global minimum is unknown, the best found reported previously is $f(\vec{x}) = 0.803619$, with $\vec{x}^*$ =(3.171456, 3.175499, 3.121430, 3.065424, 3.024695, 2.985945, 2.956863, 2.880306, 0.506161, 0.509743, 0.486445, 0.481882, 0.487077, 0.459685, 0.467321, 0.445682, 0.439956, 0.444745, 0.431957, 0.424569) with the constraint $g02$ being close to active.

3. $g03$
Maximize:

$$f(\vec{x}) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$$

subject to:

$$h_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where $n = 10$ and the bounds are $0 \leq x_i \leq 1$ $(i = 1, 2, \ldots, n)$. The global maximum is at $x_i^* = 1/\sqrt{n}$ $(i = 1, 2, \ldots, n)$ where $f(\vec{x}^*) = 1$.

4. $g04$
Minimize:

$$f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 & \\ -0.0022053x_3x_5 - 92 &\leq 0 \\ g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 & \\ +0.0022053x_3x_5 &\leq 0 \\ g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 & \end{aligned}$$

$$\begin{aligned} +0.0021813x_3^2 - 110 &\leq 0 \\ g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 & \\ -0.0021813x_3^2 + 90 &\leq 0 \\ g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 & \\ +0.0019085x_3x_4 - 25 &\leq 0 \\ g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 & \\ -0.0019085x_3x_4 + 20 &\leq 0 \end{aligned}$$

where the bounds are $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$, and $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$. The best solution is $\vec{x}^* = (78, 33, 29.995256, 45, 36.775813)$ where $f(\vec{x}^*) = -30665.539$. Two constraints are active (g_1 and g_6).

5. $g05$
   Minimize:
   $$f(\vec{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$
   subject to:
   $$\begin{aligned} g_1(\vec{x}) = -x_4 + x_3 - 0.55 &\leq 0 \\ g_2(\vec{x}) = -x_3 + x_4 - 0.55 &\leq 0 \\ h_3(\vec{x}) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 &= 0 \\ h_4(\vec{x}) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 &= 0 \\ h_5(\vec{x}) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 &= 0 \end{aligned}$$
   where the bounds are $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$, and $-0.55 \leq x_4 \leq 0.55$. The best known solution is $\vec{x}^* = (679.9453, 1026.067, 0.118876, -0.396234)$ where two constraints are active (g_1 and g_6), and $f(\vec{x}^*) = 5126.4981$.

6. $g06$
   Minimize:
   $$f(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$
   subject to:
   $$\begin{aligned} g_1(\vec{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 &\leq 0 \\ g_2(\vec{x}) = -(x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 &\leq 0 \end{aligned}$$
   where the bounds are $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The optimum solution is $\vec{x}^* = (14.095, 0.84296)$ where both constraints are active, and $f(\vec{x}^*) = -6961.81388$.

7. $g07$
   Minimize:
   $$\begin{aligned} f(\vec{x}) = {} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\ & +2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned}$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 &\leq 0 \\ g_2(\vec{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 &\leq 0 \\ g_3(\vec{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 &\leq 0 \\ g_4(\vec{x}) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 &\leq 0 \\ g_5(\vec{x}) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 &\leq 0 \\ g_6(\vec{x}) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 &\leq 0 \\ g_7(\vec{x}) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 &\leq 0 \\ g_8(\vec{x}) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} &\leq 0 \end{aligned}$$

where the bounds are $-10 \leq x_i \leq 10$ $(i = 1, 2, \ldots, 10)$. The optimum solution is $\vec{x}^*$ =(2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927) where six constraints are active (g_1, g_2, g_3, g_4, g_5 and g_6), and $f(\vec{x}^*) = 24.3062091$.

8. $g08$
Minimize:

$$f(\vec{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = x_1^2 - x_2 + 1 &\leq 0 \\ g_2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 &\leq 0 \end{aligned}$$

where the bounds are $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum is at $\vec{x}^* = (1.2279713, 4.2453733)$ where $f(\vec{x}^*) = 0.095825$.

9. $g09$
Minimize:

$$\begin{aligned} f(\vec{x}) = &(x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 \\ &+10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \end{aligned}$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 &\leq 0 \\ g_2(\vec{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 &\leq 0 \\ g_3(\vec{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 &\leq 0 \\ g_4(\vec{x}) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 &\leq 0 \end{aligned}$$

where the bounds are $-10 \leq x_i \leq 10$ $(i = 1, 2, \ldots, 7)$. The optimum solution is at $\vec{x}^*$ =(2.330499, 1.951372, −0.4775414, 4.365726, −0.6244870, 1.038131, 1.594227) where two constraints are active (g_1 and g_4), and $f(\vec{x}^*) = 680.6300573$.

10. $g10$
Minimize:
$$f(\vec{x}) = x_1 + x_2 + x_3$$
subject to:
$$\begin{aligned} g_1(\vec{x}) = -1 + 0.0025(x_4 + x_6) &\leq 0 \\ g_2(\vec{x}) = -1 + 0.0025(x_5 + x_7 - x_4) &\leq 0 \\ g_3(\vec{x}) = -1 + 0.01(x_8 - x_5) &\leq 0 \\ g_4(\vec{x}) = -x_1x_6 + 8.33252x_4 + 100x_1 - 83333.333 &\leq 0 \\ g_5(\vec{x}) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 &\leq 0 \\ g_6(\vec{x}) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 &\leq 0 \end{aligned}$$
where the bounds are $100 \leq x_i \leq 10000$, $1000 \leq x_i \leq 10000$ $(i = 2, 3)$, and $10 \leq x_i \leq 1000$ $(i = 4, 5, \ldots, 8)$. The optimum solution is $\vec{x}^*$ =(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979) where three constraints are active (g_1, g_2 and g_3), and $f(\vec{x}^*) = -15$.

11. $g11$
Minimize:
$$f(\vec{x}) = x_1^2 + (x_2 - 1)^2$$
subject to:
$$h_1(\vec{x}) = x_2 - x_1^2 = 0$$
where the bounds are $-1 \leq x_1 \leq 1$, $-1 \leq x_2 \leq 1$. The optimum solution is at $\vec{x}^* = (\pm 1/\sqrt{2}, 1/2)$ where $f(\vec{x}^*) = 0.75$.

12. $g12$
Maximize:
$$f(\vec{x}) = (100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100$$
subject to:
$$g_1(\vec{x}) = \min_{p,q,r}\{(x_1 - p)^2 - (x_2 - q)^2 - (x_3 - r)^2 - 0.0625 | p, q, r \in \{1, 2, \ldots, 9\}\} \leq 0$$
where the bounds are $0 \leq x_i \leq 1$ $(i = 1, 2, 3)$. This problem has been restated to fit the standard definition. The global maximum is at $\vec{x}^* = (5, 5, 5)$ where $f(\vec{x}^*) = 1$.

13. $g13$
Minimize:
$$f(\vec{x}) = \exp^{x_1x_2x_3x_4x_5}$$

subject to:

$$\begin{aligned} h_1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 &= 0 \\ h_2(\vec{x}) = x_2 x_3 - 5 x_4 x_5 &= 0 \\ h_3(\vec{x}) = x_1^3 + x_2^3 + 1 &= 0 \end{aligned}$$

where the bounds are $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$. The optimum solution is $\vec{x}^*$ =(−1.717143, 1.595709, 1.827247, −0.7636413, −0.763645) where $f(\vec{x}^*) = 0.0539498$.

14. $i1$
Minimize:

$$f(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14 + x_2)$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} &\leq 0 \\ g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} &\leq 0 \\ g_3(\vec{x}) = x_1 - x_4 &\leq 0 \\ g_4(\vec{x}) = 0.10471 x_1^2 + 0.04811 x_3 x_4 (14.0 + x_2) - 5 &\leq 0 \\ g_5(\vec{x}) = 0.125 - x_1 &\leq 0 \\ g_6(\vec{x}) = \delta(\vec{x}) - \delta_{\max} &\leq 0 \\ g_7(\vec{x}) = P - P_c(\vec{x}) &\leq 0 \end{aligned}$$

where:

$$\begin{aligned} \tau(\vec{x}) &= \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} \\ \tau' &= \frac{P}{\sqrt{2} x_1 x_2} \\ \tau'' &= \frac{MR}{J} \\ M &= P(L + \frac{x_2}{2}) \\ R &= \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \\ J &= 2\left(\sqrt{2} x_1 x_2 \left(\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right)\right) \\ \sigma(\vec{x}) &= \frac{6PL}{x_4 x_3^2} \\ \delta(\vec{x}) &= \frac{4PL^3}{E x_3^3 x_4} \end{aligned}$$

$$P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2 x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

and $P = 6000$lp, $L = 14$in, $E = 30 \times 10^6$psi, $g = 12 \times 10^6$psi, $\tau_{\max} = 13600$psi, $\sigma_{\max} = 30000$psi, $\delta_{\max} = 0.25$in. The bounds are $0.1 \leq x_1 \leq 2$, $0.1 \leq x_2 \leq 10$, $0.1 \leq x_3 \leq 10$ and $0.1 \leq x_4 \leq 2$. The best known solution is $\vec{x}^*$ =(0.2057296, 3.4704887, 9.0366239, 0.205729) where $f(\vec{x}^*) = 1.724852309$.

15. $i2$
Minimize:

$$\begin{aligned} f(\vec{x}) &= (0.6224)0.0625\lfloor x_1 \rfloor x_3 x_4 + (1.7781)0.0625\lfloor x_2 \rfloor x_3^2 \\ &\quad +3.1661(0.0625\lfloor x_1 \rfloor)^2 x_4 + 19.84(0.0625\lfloor x_1 \rfloor)^2 x_3 \end{aligned}$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = -0.0625\lfloor x_1 \rfloor + 0.0193x_3 &\leq 0 \\ g_2(\vec{x}) = -0.0625\lfloor x_2 \rfloor + 0.00954x_3 &\leq 0 \\ g_3(\vec{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 &\leq 0 \\ g_4(\vec{x}) = x_4 - 240 &\leq 0 \end{aligned}$$

where the bounds are $1 \leq x_i \leq 99$ $(i = 1, 2)$ and $10 \leq x_i \leq 200$ $(i = 3, 4)$. The best known solution is $\vec{x}^*$ =(0.8125, 0.4375, 42.098445, 176.636597) where $f(\vec{x}^*) = 6059.71434795$.

16. $i3$
Minimize:

$$f(\vec{x}) = (x_3 + 2)x_2 x_1^2$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = 1 - \frac{x_2^3 x_3}{71785x_1^4} &\leq 0 \\ g_2(\vec{x}) = \frac{4x_2^2 - x_1 x_2}{12566(x_2 x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 &\leq 0 \\ g_3(\vec{x}) = 1 - \frac{140.45x_1}{x_2^2 x_3} &\leq 0 \\ g_4(\vec{x}) = \frac{x_2 + x_1}{1.5} - 1 &\leq 0 \end{aligned}$$

where the bounds are $0.05 \leq x_1 \leq 2$, $0.25 \leq x_2 \leq 1.3$ and $2 \leq x_3 \leq 15$. The best known solution is $\vec{x}^*$ =(0.051683, 0.0356577, 11.297236) where $f(\vec{x}^*) = 0.012665$.

17. $c01$
Minimize:

$$f(\vec{x}) = \sum_{i=1}^{10} x_i \left(c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right)$$

subject to:

$$\begin{aligned} h_1(\vec{x}) = x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 &= 0 \\ h_2(\vec{x}) = x_4 + 2x_5 + x_6 + x_7 - 1 &= 0 \\ h_3(\vec{x}) = x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 &= 0 \end{aligned}$$

where the bounds are $0 \leq x_i \leq 1$, $(i = 1, 2, \ldots, 10)$, and $c_1 = -6.089$, $c_2 = -17.164$, $c_3 = -34.0054$, $c_4 = --5.914$, $c_5 = -24.721$, $c_6 = -14.986$, $c_7 = -24.1$, $c_8 = -10.708$, $c_9 = -26.662$, $c_{10} = -22.179$. The best known solution is $\vec{x}^*$ =(0.0407, 0.1477, 0.7832, 0.0014, 0.4853, 0.0007, 0.0274, 0.018, 0.0373, 0.0969) where $f(\vec{x}^*) = -47.761$.

18. $c02$
Minimize:

$$f(\vec{x}) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1 x_2 - x_1 x_3$$

subject to:

$$\begin{aligned} h_1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 - 25 &= 0 \\ h_2(\vec{x}) = 8x_1 + 14x_2 + 7x_3 - 56 &= 0 \end{aligned}$$

where the bounds are $0 \leq x_i \leq 10$, $(i = 1, 2, 3)$. The global optimum is at $\vec{x}^*$ =(3.512, 0.217, 3.552) where $f(\vec{x}^*) = 961.715$.

19. $c03$
Minimize:

$$\begin{aligned} f(\vec{x}) &= f_1(x_1) + f_2(x_2) \text{ and} \\ f_1(x) &= \begin{cases} 30x & \text{if } 0 \leq x < 300 \\ 31x & \text{if } 300 \leq x \leq 400 \end{cases} \\ f_2(x) &= \begin{cases} 28x & \text{if } 0 \leq x < 100 \\ 29x & \text{if } 100 \leq x < 200 \\ 30x & \text{if } 200 \leq x \leq 1000 \end{cases} \end{aligned}$$

subject to:

$$\begin{aligned} h_1(\vec{x}) = x_1 - 300 + \frac{x_3 x_4}{131.078} \cos(1.48577 - x_6) & \\ - \frac{0.90798}{131.078} x_3^2 \cos(1.47588) &= 0 \\ h_2(\vec{x}) = x_2 + \frac{x_3 x_4}{131.078} \cos(1.48577 + x_6) & \end{aligned}$$

$$
\begin{aligned}
-\frac{0.90798}{131.078}x_4^2\cos(1.47588) &= 0\\
h_3(\vec{x}) = x_5 + \frac{x_3x_4}{131.078}\sin(1.48577 + x_6)&\\
-\frac{0.90798}{131.078}x_4^2\sin(1.47588) &= 0\\
h_4(\vec{x}) = 200 - \frac{x_3x_4}{131.078}\sin(1.48577 - x_6)&\\
-\frac{0.90798}{131.078}x_3^2\sin(1.47588) &= 0
\end{aligned}
$$

where the bounds are $0 \le x_1 \le 400$, $0 \le x_2 \le 1000$, $340 \le x_3 \le 420$, $340 \le x_4 \le 420$, $-1000 \le x_5 \le 1000$, $0 \le x_3 \le 0.5236$. The best known solution is $\vec{x}^*$ =(107.81, 196.32, 373.83, 420, 21.31, 0.153) where $f(\vec{x}^*) = 8927.5888$.

20. $c04$
Maximize:

$$f(\vec{x}) = 0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$$

subject to:

$$
\begin{aligned}
g_1(\vec{x}) = x_3^2 + x_4^2 - 1 &\le 0\\
g_2(\vec{x}) = x_9^2 - 1 &\le 0\\
g_3(\vec{x}) = x_5^2 + x_6^2 - 1 &\le 0\\
g_4(\vec{x}) = x_1^2 + (x_2 - x_9)^2 - 1 &\le 0\\
g_5(\vec{x}) = (x_1 - x_5)^2 + (x_2 - x_6)^2 - 1 &\le 0\\
g_6(\vec{x}) = (x_1 - x_7)^2 + (x_2 - x_8)^2 - 1 &\le 0\\
g_7(\vec{x}) = (x_3 - x_5)^2 + (x_4 - x_6)^2 - 1 &\le 0\\
g_8(\vec{x}) = (x_3 - x_7)^2 + (x_4 - x_8)^2 - 1 &\le 0\\
g_9(\vec{x}) = x_7^2 + (x_8 - x_9)^2 - 1 &\le 0\\
g_{10}(\vec{x}) = x_2x_3 - x_1x_4 &\le 0\\
g_{11}(\vec{x}) = -x_3x_9 &\le 0\\
g_{12}(\vec{x}) = x_5x_9 &\le 0\\
g_{13}(\vec{x}) = x_6x_7 - x_5x_8 &\le 0
\end{aligned}
$$

where the bounds are $-1 \le x_i \le 1$ $(i = 1, 2, \ldots, 8)$. The best known solution is $\vec{x}^*$ =(0.9971, −0.0758, 0.553, 0.8331, 0.9981, −0.0623, 0.5642, 0.8256, 0.0000024) where $f(\vec{x}^*) = 0.866$.

21. $c05$
Maximize:

$$f(\vec{x}) = \sum_{i=1}^{10} b_i x_i - \sum_{i=1}^{5}\sum_{j=1}^{5} c_{i,j}x_{10+i}x_{10+j} - 2\sum_{j=1}^{5} d_j x_{10+j}^3$$

subject to:

$$g_j(\vec{x}) = \sum_{i=1}^{10} a_{i,j}x_i - 2\sum_{i=1}^{5} c_{i,j}x_{10+i} - 3d_j x_{10+j}^2 - e_j \quad \leq \quad 0$$

and

$$\begin{aligned}
e &= (-15, -27, -36, -18, -12) \\
c_1 &= (30, -20, -10, 32, -10) \\
c_2 &= (-20, 39, -6, 39, -20) \\
c_3 &= (-10, -6, 10, -6, -10) \\
c_4 &= (32, -31, -6, 39, -20) \\
c_5 &= (-10, 32, -10, -20, 30) \\
d &= (4, 8, 10, 6, 2) \\
a_1 &= (-16, 2, 0, 1, 0) \\
a_2 &= (0, -2, 0, 4, 2) \\
a_3 &= (-35, 0, 2, 0, 0) \\
a_4 &= (0, -2, 0, -4, -1) \\
a_5 &= (0, -9, -2, 1, -2.8) \\
a_6 &= (2, 0, -4, 0, 0) \\
a_7 &= (-1, -1, -1, -1, -1) \\
a_8 &= (-1, -2, -3, -2, -1) \\
a_9 &= (1, 2, 3, 4, 5) \\
a_{10} &= (1, 1, 1, 1, 1)
\end{aligned}$$

where the bounds are $0 \leq x_i \leq 100$ $(i = 1, 2, \ldots, 15)$. The best known solution is $\vec{x}^*$ =(0, 0, 5.147, 0, 3.0611, 11.8395, 0, 0, 0.1039, 0, 0.3, 0.3335, 0.4, 0.4283, 0.224) where $f(\vec{x}^*) = -32.386$.

22. $c06$
Minimize:

$$f(\vec{x}) \quad = \quad x_1$$

subject to:

$$\begin{aligned}
g_1(\vec{x}) = -x_1 + 35x_2^{0.6} + 35x_3^{0.6} &\leq 0 \\
h_2(\vec{x}) = -300x_3 + 7500x_5 - 7500x_6 - 25x_4x_5 + 25x_4x_6 + x_3x_4 &= 0 \\
h_3(\vec{x}) = 100x_2 + 155.365x_4 + 2500x_7 - x_2x_4 - 25x_4x_7 - 15536.5 &= 0 \\
h_4(\vec{x}) = -x_5 + \ln(-x_4 + 900) &= 0 \\
h_5(\vec{x}) = -x_6 + \ln(x_4 + 300) &= 0 \\
h_6(\vec{x}) = -x_7 + \ln(-2x_4 + 700) &= 0
\end{aligned}$$

where the bounds are $0 \leq x_1 \leq 1000$, $0 \leq x_2 \leq 40$, $0 \leq x_3 \leq 40$, $100 \leq x_4 \leq 300$, $6.3 \leq x_5 \leq 6.7$, $5.9 \leq x_6 \leq 6.4$, and $4.5 \leq x_7 \leq 6.25$. The best known solution is $\vec{x}^*$ =(193.77835, 0, 17.3272, 100.01566, 6.6846, 5.9915, 6.2145) where $f(\vec{x}^*) = 193.7783$.

23. $c07$
Minimize:

$$f(\vec{x}) \;=\; -9x_5 - 15x_8 + 6x_1 + 16x_2 + 10(x_6 + x_7)$$

subject to:

$$\begin{aligned} h_1(\vec{x}) = x_1 + x_2 - x_3 - x_4 &= 0 \\ h_2(\vec{x}) = 0.03x_1 + 0.01x_2 - x_9(x_3 + x_4) &= 0 \\ h_3(\vec{x}) = x_3 + x_6 - x_5 &= 0 \\ h_4(\vec{x}) = x_4 + x_7 - x_8 &= 0 \\ g_5(\vec{x}) = x_9x_3 + 0.02x_6 - 0.025x_5 &\leq 0 \\ g_6(\vec{x}) = x_9x_4 + 0.02x_7 - 0.025x_8 &\leq 0 \end{aligned}$$

where the bounds are $0 \leq x_i \leq 300$ $(i = 1, 2, 6)$, $0 \leq x_i \leq 100$ $(i = 3, 5, 7)$, $0 \leq x_i \leq 200$ $(i = 4, 8)$, and $0.01 \leq x_9 \leq 0.03$. The optimum solution is at $\vec{x}^*$ =(0, 100, 0, 100, 0, 0, 100, 200, 0.1) where $f(\vec{x}^*) = -400$.

24. $c08$
Minimize:

$$f(\vec{x}) \;=\; -x_1 - x_2$$

subject to:

$$\begin{aligned} g_1(\vec{x}) = -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 - 2 &\leq 0 \\ g_2(\vec{x}) = -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 &\leq 0 \end{aligned}$$

where the bounds are $0 \leq x_1 \leq 3$ and $0 \leq x_2 \leq 4$. The optimum solution is at $\vec{x}^*$ =(2.3295, 3.17846) where $f(\vec{x}^*) = -5.5079$.

Appendix B

Results for the Mezura-Coello Benchmark

The results for the engineering problems proposed as benchmark by Mezura and Coello [5] are in Tables B.2–B.9. The optimal values for these problems are summarized in Table B.1.

Function	Optimal value	max/min
$c01$	-47.761	Minimize
$c02$	961.715	Minimize
$c03$	8927.5888	Minimize
$c04$	0.866	Maximize
$c05$	-32.386	Maximize
$c06$	193.7783493	Minimize
$c07$	-400	Minimize
$c08$	5.5079	Minimize

Table B.1: The known or reported optimum values for the rest of the test functions. The column *max/min* tells whether the problem is a maximization or a minimization to better interpret the results.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best	-45.14793	-47.761	47.761	47.761
worst	-40.4601	-42.47365	47.759	47.6708
mean	-43.4492	-46.81045	47.7609	47.757
median	-43.449	-46.81045	47.761	47.761
variance	0.947	1.6654	1.021E-7	2.602E-4
standard dev.	0.973	1.2654	3.196E-4	0.016
# feasibles	0	0	0	0
# ϵ-feasibles	30	30	30	30

Table B.2: Results for function $c01$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best	961.7181	961.7244	961.7151	961.7151
worst	969.401	966.21389	961.7151	961.7151
mean	964.003	963.11675	961.7151	961.7151
median	963.3009	962.8708	961.7151	961.7151
variance	4.689	1.74	3.761E-14	2.457E-12
standard dev.	2.1654	1.319	1.939E-7	1.567E-6
# feasibles	0	0	0	0
# ϵ-feasibles	30	30	30	30

Table B.3: Results for function $c02$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best				
worst				
mean				
median				
variance				
standard dev.				
# feasibles	0	0	0	0
# ϵ-feasibles	30	30	30	30

Table B.4: Results for function $c03$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best	0.866	0.866	0.866	0.866
worst	0.512	0.571	0.866	0.866
mean	0.7657	0.828	0.866	0.866
median	0.8623	0.864	0.866	0.866
variance	0.0167	0.0081	5.05E-13	5.739E-11
standard dev.	0.1294	0.0904	7.106E-7	7.575E-6
# feasibles	30	30	30	30
# ϵ-feasibles	30	30	30	30

Table B.5: Results for function $c04$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best				
worst				
mean				
median				
variance				
standard dev.				
# feasibles	30	30	30	30
# ϵ-feasibles	30	30	30	30

Table B.6: Results for function $c05$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best	452.557	440.694	193.786	193.785
worst	691.273	657.992	325.149	325.157
mean	550.058	544.574	220.059	206.923
median	539.914	536.7099	193.786	193.786
variance	3482.56	3481.63	2760.89	1553.06
standard dev.	59.013	59.005	52.544	39.408
# feasibles	0	0	0	0
# ϵ-feasibles	30	30	30	30

Table B.7: Results for function $c06$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best	-401.92	-402.426	400	400
worst	-397.906	-397.756	399.943	399.789
mean	-400.21	-399.955	399.995	399.962
median	-400.335	-400.28	399.999	399.999
variance	1.0639	1.1892	1.17E-4	0.0036
standard dev.	1.0314	1.0905	0.0108	0.0602
# feasibles	0	0	0	0
# ϵ-feasibles	30	30	30	30

Table B.8: Results for function $c07$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.

Statistic value	$ES(\mu+\lambda)$	B-$ES(\mu+\lambda)$	DE_1	B-DE_1
best	5.50801	5.50801	5.50801	5.50801
worst	5.50801	5.50801	5.50801	5.50801
mean	5.50801	5.50801	5.50801	5.50801
median	5.50801	5.50801	5.50801	5.50801
variance	1.908E-16	7.045E-13	3.155E-30	3.155E-30
standard dev.	1.381E-8	8.393E-7	1.776E-15	1.776E-15
# feasibles	30	30	30	30
# ϵ-feasibles	30	30	30	30

Table B.9: Results for function $c08$. The second and third column represent the comparison between the normal ES and the Baldwinian one, respectively. The fourth and fifth is the comparison between the normal DE and the Baldwinian one respectively.