FPBIL: A Parameter-free Evolutionary Algorithm

Gustavo Caldas and Roberto Schirru
CNEN and COPPE/UFRJ
Brazil

1. Introduction

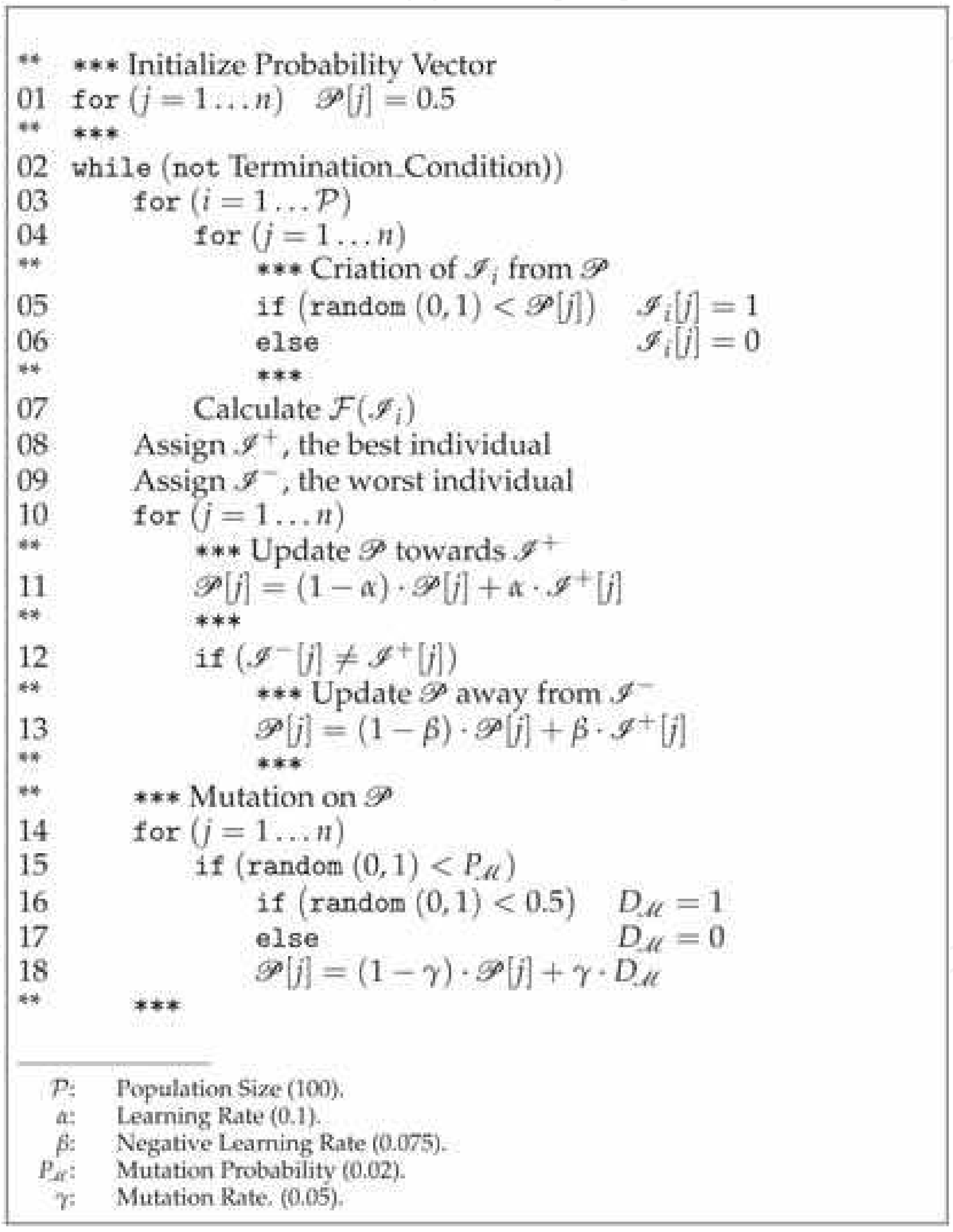
The purpose of this chapter is to describe a new algorithm named FPBIL (parameter-Free PBIL), an evolution of PBIL (Population-Based Incremental Learning). FPBIL, as well as PBIL (Baluja, 1994), Genetic Algorithms (GAs) (Holland, 1992) and others are general purpose population-based evolutionary algorithms. The success of GAs is unquestionable (Goldberg, 1989). Despite that, PBIL has shown to be superior in many aspects.
PBIL is a evolutionary algorithm developed as an attempt to mimic the behavior of the Genetic Algorithms in an advanced stage of its execution, "in equilibrium". The result shows unexpectedly that the PBIL surpasses (Baluja, 1995) the genetic algorithms in almost all aspects. The PBIL is faster and finds better results (Machado, 1999). However, PBIL depends on five parameters which need to be adjusted before each application. For example, variations in the learning rate produce completely different behaviors (Baluja, 1994).
Up to today, every evolutionary algorithm, like PBIL, just mentioned, depends on at least one parameter which, if not adjusted properly, can cause the algorithm to be very inefficient. Consequently, the less parameters an algorithm has, the minor the risk of it not reaching all its potential in some particular application; and the less the time spent in finding the appropriate parameter's values.
One of the benefits of FPBIL—perhaps the most important—is that it is a parameter free algorithm (the origin of the F in FPBIL), which means that a parameter optimization, an application-dependent procedure required by other algorithms in order to achieve better results, is not necessary in FPBIL. Parameter optimization demands intense computational effort, a precious time often not taken into account when somebody claims that an algorithm finds a better result in a shorter amount of time.
Based on PBIL, FPBIL is built with the guarantee of a better performance than that of PBIL, which also means (whenever the PBIL has a good outcome) a better performance in comparison to other algorithms, besides the advantage of none additional computational cost in adjusting parameters.
We begin this chapter by describing the PBIL algorithm and, then, we present the main steps to the FPBIL algorithm it self. Afterwards, we compare the performance of FPBIL against other algorithms in typical benchmark problems and finally we propose some concluding remarks.

2. PBIL algorithm

The PBILwas created in 1994, by Shumeet Baluja. It was inspired in its previous work with Ari Juels (Juels et al., 1993) in an attempt to simulate the behavior of the genetic algorithms

Source: Advances in Evolutionary Algorithms, Book edited by: Witold Kosiński, ISBN 978-953-7619-11-4, pp. 468, November 2008,
I-Tech Education and Publishing, Vienna, Austria

(Holland, 1992; Goldberg, 1989) in "equilibrium state", after repeated applications of the crossover operator. The algorithm referenced here by "PBIL" had its publication later, in 1995 (Baluja, 1995), in which 27 problems, commonly explored in the literature of genetic algorithms, were examined by seven different optimization techniques, PBIL having achieved optimum performance in more than 80% of the cases. PBIL algorithm is shown in figure 1.



```
*** Initialize Probability Vector
for ($j = 1 \ldots n$)   $\mathscr{P}[j] = 0.5$
***
while (not Termination_Condition))
    for ($i = 1 \ldots \mathcal{P}$)
        for ($j = 1 \ldots n$)
            *** Criation of $\mathscr{I}_i$ from $\mathscr{P}$
            if (random (0,1) $< \mathscr{P}[j]$)   $\mathscr{I}_i[j] = 1$
            else                                   $\mathscr{I}_i[j] = 0$
            ***
        Calculate $\mathcal{F}(\mathscr{I}_i)$
    Assign $\mathscr{I}^+$, the best individual
    Assign $\mathscr{I}^-$, the worst individual
    for ($j = 1 \ldots n$)
        *** Update $\mathscr{P}$ towards $\mathscr{I}^+$
        $\mathscr{P}[j] = (1 - \alpha) \cdot \mathscr{P}[j] + \alpha \cdot \mathscr{I}^+[j]$
        ***
        if ($\mathscr{I}^-[j] \neq \mathscr{I}^+[j]$)
            *** Update $\mathscr{P}$ away from $\mathscr{I}^-$
            $\mathscr{P}[j] = (1 - \beta) \cdot \mathscr{P}[j] + \beta \cdot \mathscr{I}^+[j]$
            ***
    *** Mutation on $\mathscr{P}$
    for ($j = 1 \ldots n$)
        if (random (0,1) $< P_{\mathcal{M}}$)
            if (random (0,1) $< 0.5$)   $D_{\mathcal{M}} = 1$
            else                        $D_{\mathcal{M}} = 0$
            $\mathscr{P}[j] = (1 - \gamma) \cdot \mathscr{P}[j] + \gamma \cdot D_{\mathcal{M}}$
    ***
```

$\mathcal{P}$:	Population Size (100).
α:	Learning Rate (0.1).
β:	Negative Learning Rate (0.075).
$P_{\mathcal{M}}$:	Mutation Probability (0.02).
γ:	Mutation Rate. (0.05).

Fig. 1. PBIL Algorithm.

In PBIL, a subset $\mathcal{S}_{\mathcal{B}}$ of the search space $\mathcal{B}$ of some optimization problem is explored from one hypercube $\mathcal{H}_n \equiv [0, 1]^n$, in such a way that each vertex of $\mathcal{H}_n$, that is, each point of $\check{\mathcal{H}}_n \equiv \{0, 1\}^n$ corresponds to a point of $\mathcal{S}_{\mathcal{B}}$. This correspondence is made by the mapping

$$\begin{aligned}\mathcal{M}^n_{\mathcal{S}_\mathcal{B}} &: \check{\mathcal{H}}_n \longrightarrow \mathcal{S}_\mathcal{B} \\ \mathscr{I} \equiv (I_1, I_2, \ldots, I_n) &\longmapsto \mathcal{M}^n_{\mathcal{S}_\mathcal{B}}(\mathscr{I}),\end{aligned} \tag{1}$$

with $I_k \equiv \mathscr{I}[k] \in \{0, 1\}$, meaning that $\mathcal{M}^n_{\mathcal{S}_\mathcal{B}}$ maps a bit vector with n bits into a point of $\mathcal{S}_\mathcal{B}$ — a candidate solution to the problem.
After the vertices of $\mathcal{H}_n$ are duly mapped into $\mathcal{S}_\mathcal{B}$, the PBIL works exactly in the same way, independently of the current application and this is what makes the PBIL algorithm[1] versatile, meaning that the necessary and sufficient condition in order that an optimization problemcan be boarded by PBIL is the existence of $\mathcal{M}^n_{\mathcal{S}_\mathcal{B}}$.

A point $\mathscr{P} \in \mathcal{H}_n$ is called probability vector and it plays a central role in PBIL-like algorithms. Its n components $p_k \equiv \mathscr{P}[k] \in [0, 1]$ are suitable for representing the probability of choosing by chance the number 1 in a set $\Omega = \{0, 1\}$. From $\mathscr{P}$ is possible to construct an army of $\mathscr{I}$ objects. All we have to do is to pick I_k to be 1 or 0, probabilistically, according to p_k—the more p_k is close to 1, the more is I_k likely to be 1.
At the beginning of PBIL each point of $\mathcal{S}_\mathcal{B}$ must be treated as potential best solution and $\mathcal{P}$ vertices of $\mathcal{H}_n$ are, therefore, chosen randomly from a uniform probability distribution. This uniform probability distribution is nothing but $\mathscr{P}_0 = (0.5, 0.5, \ldots, 0.5)$, the center of $\mathcal{H}_n$.
In PBIL's terminology, the $\mathcal{P}$ vertices $\mathscr{I}_k$ of $\mathcal{H}_n$ selected from $\mathscr{P}_\mathcal{G}$ forma "population"—the "generation" $\mathcal{G}$—and each $\mathscr{I}_k$ is called an "individual". The PBIL algorithm consists in, once established the individuals of generation 0, constructing $\mathscr{P}_1$, which will generate the next population—generation 1. The process is repeated until an individual of some generation is considered to be good enough. In this sense, the PBIL algorithm may be viewed as the motion of $\mathscr{P}$ inside $\mathcal{H}_n$ until $\mathscr{P}$ gets close enough to some point of $\check{\mathcal{H}}_n$ corresponding to a satisfactory solution; the laws of motion being the PBIL rules by which $\mathscr{P}$ is updated from generation to generation.
The measure of how good an individual is, is given by the fitness function

$$\begin{aligned}\mathcal{F} &: \check{\mathcal{H}}_n \longrightarrow \mathbb{R}^+ \\ \mathscr{I} &\longmapsto \mathcal{F} = \mathcal{F}(\mathscr{I}),\end{aligned} \tag{2}$$

whose form depends explicitly on the application.
The construction of $\mathscr{P}_{\mathcal{G}+1}$ from the individuals of the generation $\mathcal{G}$ is the main process in a PBIL-like algorithm. Any point $\mathscr{P}_{\mathcal{G}+1}$ of $\mathcal{H}_n$ different from $\mathscr{P}_0$ generates a non-uniform probability distribution on $\check{\mathcal{H}}_n$. The strategy is to modify, generation after generation, this probability distribution trying to turn ever more likely the sprouting of $\mathscr{I}^\dagger$, the optimum solution. In PBIL, $\mathscr{P}_{\mathcal{G}+1}$ is constructed in two steps.
In the first step, the following operations are carried through:

$$\mathscr{P}_{\mathcal{G}+1/2} = \mathscr{P}_\mathcal{G} + \alpha \cdot (\mathscr{I}^+ - \mathscr{P}_\mathcal{G}) \tag{3}$$

[1] This is also true for other algorithms working with bitstrings, such as Genetic Algorithms.

$$\mathscr{P}'_{\mathcal{G}+1}[j] = \begin{cases} \mathscr{P}_{\mathcal{G}+1/2}[j] + \beta \cdot (\mathscr{I}^{+}[j] - \mathscr{P}_{\mathcal{G}+1/2}[j]) & \text{if } \mathscr{I}^{+}[j] \neq \mathscr{I}^{-}[j] \\ \mathscr{P}_{\mathcal{G}+1/2}[j] & \text{if } \mathscr{I}^{+}[j] = \mathscr{I}^{-}[j] \end{cases} \tag{4}$$

where $\mathscr{I}^{+}$ and $\mathscr{I}^{-}$ are respectively the best and worst individuals of generation $\mathcal{G}$. That is, $\mathscr{P}_{\mathcal{G}}$ initially is dislocated towards $\mathscr{I}^{+}$ and then, away from $\mathscr{I}^{-}$, with the intention to favor the occurrence of better individuals in the following generation.
In the second step $\mathscr{P}'_{\mathcal{G}+1}$ suffers mutation, whose objective is to allow that some component of $\mathscr{P}$ reaching the value 1 (or 0) has the possibility to evolve again—since, once $p_k = 1$ (or $p_k = 0$), it can not change by means of equation (4). Such mutation consists of moving the components of $\mathscr{P}'_{\mathcal{G}+1}$ in the direction of $D_{\mathcal{M}}$ (randomly 0 or 1). This means that each component $\mathscr{P}'_{\mathcal{G}+1}[j]$ will suffer, or not, a displacement (according to the “mutation probability”) in the form of:

$$\mathscr{P}_{\mathcal{G}+1}[j] = \mathscr{P}'_{\mathcal{G}+1}[j] + \gamma \cdot (D_{\mathcal{M}} - \mathscr{P}'_{\mathcal{G}+1}[j]). \tag{5}$$

As can be verified in figure 1, the PBIL algorithm needs five parameters to work, whose values were determined experimentally in order to maximize the average performance of the algorithm in a set of different applications. In the next section, we will show how to extend PBIL to be parameter-free.

3. FPBIL: parameter-Free PBIL

FPBIL is a variation of PBIL which basically tries to eliminate the necessity of the PBIL’s parameter by modifying some of its fundamental principles. The result is a more efficient algorithm, with a superior search power and without parameters.
As in PBIL, the FPBIL algorithm presents a probability vector $\mathscr{P}$, with n components $p_k \in [0, 1]$, from which $\mathcal{P}$ individuals $\mathscr{I}_k$ of some generation are created. The characteristic that differentiates them is that FPBIL uses generic mechanisms to become free of parameters, especially in the way $\mathscr{P}$ is updated and the mutation is implemented. The FPBIL Algorithm is presented in figure 2.

3.1 $\mathscr{P}$ update: eliminating the parameters α and β

In the algorithm PBIL, the probability vector is updated by suffering a small displacement approaching to the best individual and another displacement moving away from the worst individual. In some variants of the PBIL (Baluja & Caruana, 1995; Baluja & Davies, 1998; Machado, 2005), only the best individual is used, or only the worst individual, or also, the average of the first best individuals. The fact is that, in order to evaluate who are the best and worst individuals, all the individuals must be evaluated, which means that all PBIL algorithms waste almost all the information available about the search space.

```
*** Definition of Functions $C_x$ and $D_x$
$D_x \equiv 1/(1+x)$
$C_x \equiv D_x^{-1} \equiv 1/x - 1$
***
*** Initialization of the Probability Vector, $\mathcal{P}_0$ and $d$
for ($j = 1 \ldots n$)   $\mathscr{P}[j] = 0.5$
$\mathcal{P}_0 = 7(1+1/n)^n$
$d = 1/3$
***
while (not Termination_Condition))
    *** Determination of $\mathcal{P}$
    if (Fluctuation in $c$)
        $\mathcal{P}_0 = \mathcal{P}_0 + 1$
        if ($\Delta\langle c\rangle_{\mathcal{G}} < 1\%$)
            $d = 1/3$
            $\mathscr{P} = \mathscr{P}_0$
    $c = C_d$
    $\mathcal{P} =$ (int)$\mathcal{P}_0(1+1/c)^c(\mathcal{P}_0/7)^{-c/n}$
    ***
    for ($i = 1 \ldots \mathcal{P}$)
        Creation of $\mathscr{I}_i$ from $\mathscr{P}$, as in PBIL
        Calculate $\mathcal{F}(\mathscr{I}_i)$
    for ($j = 1 \ldots n$)
        *** Update $\mathscr{P}$
        $\mathscr{P}[j] = \left(\sum_{i=1}^{\mathcal{P}} \mathcal{F}(\mathscr{I}_i) \cdot \mathscr{I}_i[j]\right) / \left(\sum_{i=1}^{\mathcal{P}} \mathcal{F}(\mathscr{I}_i)\right)$
        ***
    *** Mutation in $\mathscr{P}$
    $c$  = Count of Cases ($\mathscr{P}[j] \leq d$             or $\mathscr{P}[j] \geq 1 - d$)
    $c'$ = Count of Cases ($\mathscr{P}[j] \leq D_{C_d-1}$   or $\mathscr{P}[j] \geq 1 - D_{C_d-1}$)
    if        ($c > C_d$)       $d = D_{c+1}$
    else if ($c' < C_d - 1$)   $d = D_{c-1}$
    if ($d > 1/3$)   $d = 1/3$
    for ($j = 1 \ldots n$)
        if ($\mathscr{P}[j] < d$)       $\mathscr{P}[j] = d$
        if ($\mathscr{P}[j] > 1 - d$)   $\mathscr{P}[j] = 1 - d$
    ***
```

Fig. 2. FPBIL Algorithm.

The rule according to which the FPBIL updates its probability vector is

$$\mathscr{P}'_{\mathcal{G}+1} = \frac{\sum_{i=1}^{\mathcal{P}} \mathcal{F}_i \cdot \mathscr{I}_i}{\sum_{i=1}^{\mathcal{P}} \mathcal{F}_i}, \tag{6}$$

which reflects exactly an average in which all $\mathcal{P}$ individuals are used. The difference is that this average is weighed by the fitness $\mathcal{F}_i \equiv \mathcal{F}(\mathscr{I}_i)$ of each individual. In order to appreciate better the change caused by this detail, it can be deduced that

$$\mathscr{P}'_{\mathcal{G}+1} = \frac{\sum_{i=1}^{\mathcal{P}} \mathscr{I}_i}{\mathcal{P}} + \xi \cdot \mathscr{D}, \tag{7}$$

with

$$\xi \equiv \frac{1}{\mathcal{P}} \sum_{\mathcal{F}_i > \langle \mathcal{F} \rangle} \frac{\mathcal{F}_i - \langle \mathcal{F} \rangle}{\langle \mathcal{F} \rangle} \tag{8}$$

and

$$\mathscr{D} \equiv \frac{\sum_{\mathcal{F}_i > \langle \mathcal{F} \rangle} (\mathcal{F}_i - \langle \mathcal{F} \rangle) \cdot \mathscr{I}_i}{\sum_{\mathcal{F}_i > \langle \mathcal{F} \rangle} (\mathcal{F}_i - \langle \mathcal{F} \rangle)} - \frac{\sum_{\mathcal{F}_i < \langle \mathcal{F} \rangle} (\langle \mathcal{F} \rangle - \mathcal{F}_i) \cdot \mathscr{I}_i}{\sum_{\mathcal{F}_i < \langle \mathcal{F} \rangle} (\langle \mathcal{F} \rangle - \mathcal{F}_i)}. \tag{9}$$

Note that $\sum_{i=1}^{\mathcal{P}} \mathscr{I}_i / \mathcal{P}$ is approximately $\mathscr{P}_{\mathcal{G}}$ so that equation (7) resembles the structure of equations (3) and (4) corresponding to PBIL.
The advantage in using $\mathscr{D}$ is that the direction of the displacement is not based only on the best and worst individuals, but in all the available information about the search space at some generation ($\mathcal{P}$ evaluated individuals). Another detail about $\mathscr{D}$ is that the averages are not simple, but weighed by the differences between the fitness of each individual and the average fitness, so that very bad or very good individuals exert more influence than others with fitness next to the average.
It is worth noting that each point $\mathscr{P}$ of $\mathcal{H}_n$ can be associated to an average fitness ${}_{\mathscr{P}}\langle \mathcal{F} \rangle$ through

$$\begin{aligned} {}_{\mathscr{P}}\langle \mathcal{F} \rangle : \mathcal{H}_n &\longrightarrow \mathbb{R} \\ \mathscr{P} &\longmapsto {}_{\mathscr{P}}\langle \mathcal{F} \rangle = \sum_{i=1}^{2^n} \mathcal{F}_i \cdot \mathscr{P}(\mathscr{I}_i). \end{aligned} \tag{10}$$

The reason is that from $\mathscr{P}$, each individual $\mathscr{I}_i$ has a probability $\mathscr{P}$ ($\mathscr{I}_i$) of being picked. After $\mathcal{P}$ tries, the individual $\mathscr{I}_i$ is picked $\mathcal{P}_i$ times. In the limit when $\mathcal{P}$ becomes sufficiently big, we have

$$\lim_{\mathcal{P} \to \infty} \langle \mathcal{F} \rangle = \lim_{\mathcal{P} \to \infty} \frac{\sum_{j=1}^{\mathcal{P}} \mathcal{F}_j}{\mathcal{P}} = \lim_{\mathcal{P} \to \infty} \sum_{i=1}^{2^n} \mathcal{F}_i \cdot \frac{\mathcal{P}_i}{\mathcal{P}} \tag{11}$$

$$= \sum_{i=1}^{2^n} \mathcal{F}_i \cdot \mathscr{P}(\mathscr{I}_i). \tag{12}$$

Since ${}_{\mathscr{P}}\langle \mathcal{F} \rangle$ is such an average, it is continuous, differentiable and it doesn't have any local maximum or minimum in $\mathcal{H}_n$ - $\check{\mathcal{H}}_n$, which means that the extreme points of ${}_{\mathscr{P}}\langle \mathcal{F} \rangle$ in $\mathcal{H}_n$ occurs for $\mathscr{I}^\dagger$ and $\mathscr{I}^\perp$ in $\check{\mathcal{H}}_n$, with ${}_{\mathscr{I}^\dagger}\langle \mathcal{F} \rangle = \mathcal{F}(\mathscr{I}^\dagger)$ and ${}_{\mathscr{I}^\perp}\langle \mathcal{F} \rangle = \mathcal{F}(\mathscr{I}^\perp)$—where $\mathscr{I}^\perp$ represents the worst individual in $\check{\mathcal{H}}_n$. And that is just interesting.

In each generation of FPBIL, we have $\langle \mathcal{F} \rangle \approx {}_{\mathscr{P}}\langle \mathcal{F} \rangle$ and the $\mathcal{P}$ individuals $\mathscr{I}_k$ are divided into two groups: those with $\mathcal{F}(\mathscr{I}_k) > \langle \mathcal{F} \rangle$ and those with $\mathcal{F}(\mathscr{I}_k) < \langle \mathcal{F} \rangle$. If we represent each of these groups respectively by the points

$$\langle \mathscr{P} \rangle_{>} \equiv \frac{\sum_{\mathcal{F}_i > \langle \mathcal{F} \rangle} (\mathcal{F}_i - \langle \mathcal{F} \rangle) \cdot \mathscr{I}_i}{\sum_{\mathcal{F}_i > \langle \mathcal{F} \rangle} (\mathcal{F}_i - \langle \mathcal{F} \rangle)} \tag{13}$$

and

$$\langle \mathscr{P} \rangle_{<} \equiv \frac{\sum_{\mathcal{F}_i < \langle \mathcal{F} \rangle} (\langle \mathcal{F} \rangle - \mathcal{F}_i) \cdot \mathscr{I}_i}{\sum_{\mathcal{F}_i < \langle \mathcal{F} \rangle} (\langle \mathcal{F} \rangle - \mathcal{F}_i)} \tag{14}$$

we see from equation (9) that FPBIL works in such a way that $\mathscr{P}$ moves in the direction that ${}_{\mathscr{P}}\langle \mathcal{F} \rangle$ grows, leading, theorically at least, to $\mathscr{I}^{+}$. Just to compare, in PBIL, $\mathscr{I}_{+}$ and $\mathscr{I}_{-}$ are used instead of $\langle \mathscr{P} \rangle_{>}$ and $\langle \mathscr{P} \rangle_{<}$, which means that PBIL is much easier to get caught by local optimums.

Obviously we can only bet that the approximation $\langle \mathcal{F} \rangle \approx {}_{\mathscr{P}}\langle \mathcal{F} \rangle$ is good enough. Only in the limit $\mathcal{P} \to \infty$ can we be sure. The same limit when we would have already evaluated every element of $\check{\mathcal{H}}_{n}$, so that we would no longer need a search algorithm. Fortunately, the FPBIL algorithm also have proper mechanisms that compensate for the finiteness of $\mathcal{P}$. ξ can be considered to be one of those.

It can be verified that ξ plays a similar role just like α or β, related to the intensity of the displacement suffered by $\mathscr{P}$. While α and β are constants, ξ varies in accordance to the fitness distribution of each generation. More precisely, ξ is the half of the mean absolute deviation, relative to the average, of the fitness:

$$\xi = \frac{1}{\langle \mathcal{F} \rangle} \frac{1}{\mathcal{P}} \sum_{\mathcal{F}_i > \langle \mathcal{F} \rangle} \mathcal{F}_i - \langle \mathcal{F} \rangle \tag{15}$$

$$= \frac{1}{2} \frac{1}{\langle \mathcal{F} \rangle} \left(\frac{1}{\mathcal{P}} \sum_{i=1}^{\mathcal{P}} |\mathcal{F}_i - \langle \mathcal{F} \rangle| \right) \tag{16}$$

$$= \frac{1}{2} \frac{\delta}{\langle \mathcal{F} \rangle} = \frac{1}{2} \delta_r. \tag{17}$$

The mean absolute deviation (δ) is a measure of dispersion of a distribution, just like the standard deviation. δ_r is only another way to express the same dispersion relative to the average.

At the beginning of an execution of the FPBIL, the individuals generally possess a very bad fitness. While no individual detaches, ξ is small—the algorithm does not take risks by making a decision on which direction to follow. When the first good individuals appear, ξ increases considerably. As the average fitness goes up, ξ diminishes gradualy—preventing itself from premature convergence. Finally, when the optimum solution is near, ξ becomes very small, making sure that $\mathscr{P}$ will not have great oscillations around it but, instead, it might be reached.

3.2 Mutation: eliminating the parameters $P_{\mathcal{M}}$ and γ

The role of mutation is to give "second chances" to the components of $\mathscr{P}$ that reach the values 0 or 1 when they were not supposed to do so. In the limit $\mathcal{P} \rightarrow \infty$, FPBIL would not need mutation at all, as we have already discussed. But in a real situation, mutation is another mechanism that compensates for finite $\mathcal{P}$, and it is essential to FPBIL.

The PBIL carries mutation probabilistically (in accordance to $P_{\mathcal{M}}$) through random displacements (proportional to γ) in the components of $\mathscr{P}$. The FPBIL algorithm follows a more direct strategy, exploring the meaning of the probability vector. First, the algorithm hinders any component of $\mathscr{P}$ from reaching the values 0 or 1. This way the emergence of any individual in $\check{\mathcal{H}}_n$ is always possible. That is accomplished by restricting every component p_k of $\mathscr{P}$ to the interval $[d, 1 - d]$. As a consequence, the probability of choosing by chance any individual from $\mathscr{P}$ will always be between d^n and $(1 - d)^n$.

Given any value d, the number c of components of $\mathscr{P}$ with $p_k \leq d$ or $p_k \geq 1 - d$ is considere to be the number of components which are in the correct position. Then it is possible to find the optimum value of d, so that it maximizes the probability of choosing from $\mathscr{P}$ an individual with the corresponding c correct components and so that is also capable of inverting the trend of some component going toward the wrong direction. The probability which must be maximized is, therefore,

$$p(d) = d(1-d)^c, \tag{18}$$

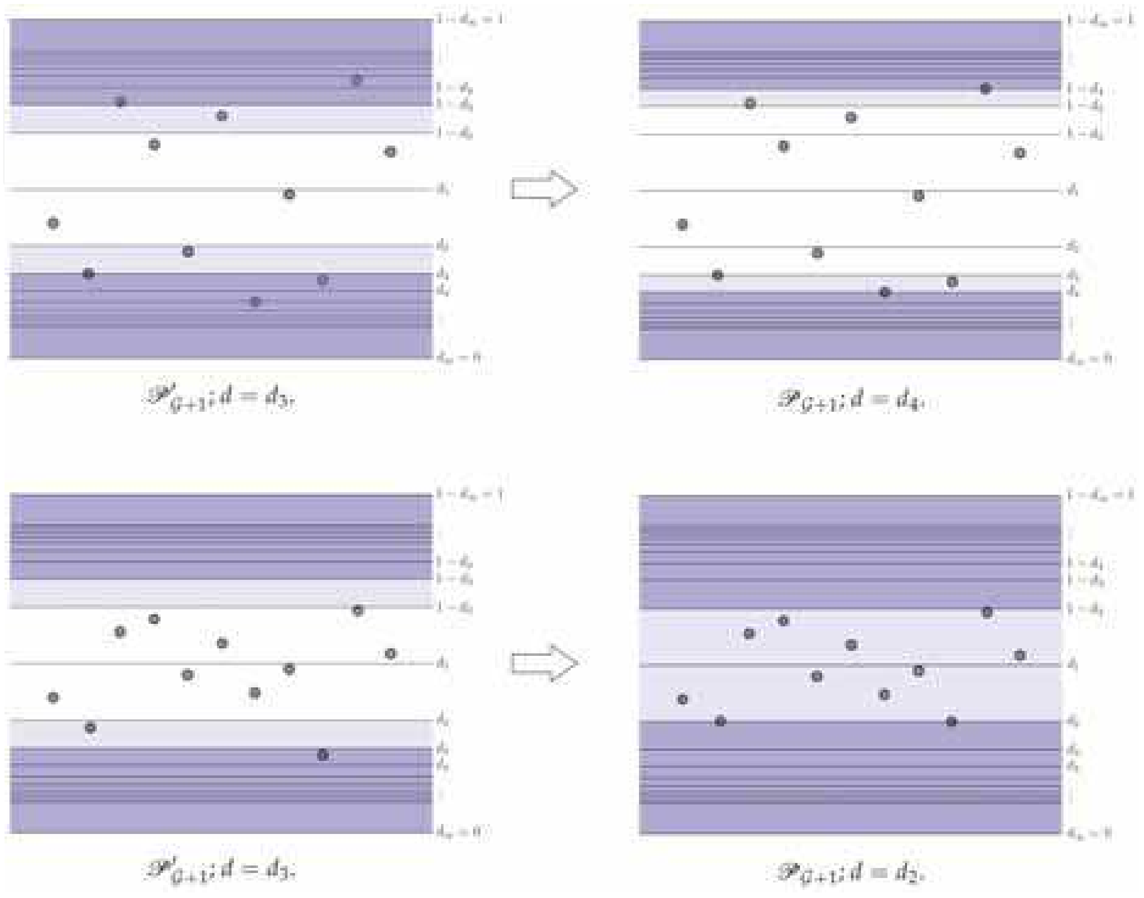
giving

$$d_c = \frac{1}{1+c}. \tag{19}$$

The FPBIL algorithm takes d to be initially (in generation 0) $d_2 = 1/3$—the biggest value of d_c different from 0.5. After $\mathscr{P}_{\mathcal{G}}$ is updated to $\mathscr{P}'_{\mathcal{G}+1}$, we count how many ($c$) components of $\mathscr{P}'_{\mathcal{G}+1}$ satisfy $p_k \leq d_2$ (or $p_k \geq 1 - d_2$). If $c \geq 3$, d becomes $d_3 = 1/4$. If $d = d_3$ and $c \geq 4$ (the number of components of $\mathscr{P}'_{\mathcal{G}+1}$ that satisfy $p_k \leq d_3$ (or $p_k \geq 1 - d_3$)), d becomes $d_4 = 1/5$, and so on. Thus, it is possible to diminish d gradually as P gets close to some point in $\check{\mathcal{H}}_n - \mathscr{I}^+$, expectedly.

But there is also a mechanism that allows d to grow. If, for example, $d = d_5 = 1/6$ but $c \ngeq 6$, we count how many (c') components of $\mathscr{P}'_{\mathcal{G}+1}$ satisfy $p_k \leq d_4$ (or $p_k \geq 1-d_4$). If $c' < 5$, d becomes $d_4 = 1/5$. If $d = d_4$, $c \ngeq 5$ and $c'< 4$ (the number of $\mathscr{P}'_{\mathcal{G}+1}$ components that satisfy $p_k \leq d_3$ (or $p_k \geq 1 - d_3$)), d becomes $d_3 = 1/4$, and so on, until d hits the value $d_2 = 1/3$, the biggest allowed.

After we count c and $c'$and update d, mutation do its real job: it brings back to d (or to $1-d$) any $\mathscr{P}'_{\mathcal{G}+1}$ component smaller than d (or bigger than $1-d$), transforming $\mathscr{P}'_{\mathcal{G}+1}$ into $\mathscr{P}_{\mathcal{G}+1}$. FPBIL's mutation is illustrated in figure 3, where each point ● represents a component of $\mathscr{P}$. As we can see, d values work as "gates" that open or close depending on the values of c and c'.



$1-d_0=1$
$1-d_1$
$1-d_2$
$1-d_3$
d_4
d_3
d_2
d_1
$d_0=0$
$\mathscr{P}'_{\mathcal{G}+1}; d=d_3.$
$\mathscr{P}_{\mathcal{G}+1}; d=d_4.$
$\mathscr{P}'_{\mathcal{G}+1}; d=d_3.$
$\mathscr{P}_{\mathcal{G}+1}; d=d_2.$

Fig. 3. Two examples of mutation: in the first, d diminishes; in the second, it grows.

3.3 Variable population size and reinitializations: eliminating the parameter $\mathcal{P}$

The size of $\mathcal{H}_n$ is 2^n, which is usually very large. The population sizes commonly used in PBIL are very small fractions of this value. Therefore, it is reasonable to use the relation

$$\mathcal{P} = 2^{\frac{n}{w}} \tag{20}$$

for some w.

Perhaps the most remarkable aspect of FPBIL (and PBIL) is that the population size does not have to be a constant—sheer nonsense for GA users. Since every population is generated from $\mathscr{P}_{\mathcal{G}}$ instantly after $\mathscr{P}_{\mathcal{G}}$ is created, it does not matter whether we generate only one or a thousand individuals. There is no higher complexity involved than choosing how many individuals we want.

As the number c of correct components of $\mathscr{P}$ increases, we must, therefore, need only

$$\mathcal{P}_c = \frac{k}{(1-d)^c} \cdot 2^{\frac{n-c}{w}} \tag{21}$$

individuals, where the factor $k/(1-d)^c$ only appears to assure that the correct component are reproduced with 99.9% of probability (for $k = 7$) (Caldas, 2006). Using equation (19), it can be written as

$$\mathcal{P}_c = \left(1+\frac{1}{c}\right)^c \cdot \mathcal{P}_0 \left(\frac{\mathcal{P}_0}{k}\right)^{-\frac{\xi}{n}} \tag{22}$$

where $\mathcal{P}_0$ is the initial population, corresponding $c = 0$. FPBIL is initiated with $\mathcal{P}_0 = \mathcal{P}_n$ and every time c suffers a fluctuation, $\mathcal{P}_0$ is increased by 1. That occurs because, when the time average $\langle c \rangle_{\mathcal{G}}$ of c stops varying, the algorithm must be imprisoned in a local optimum, so it must be reinitiated. The difference is that in each reinitialization $\mathcal{P}_0$ will be each time bigger (due to the fluctuations of c), increasing gradually the power of search of the FPBIL. A fluctuation in c will be computed whenever c does not grow or decrease directly, that is, whenever c, as a function of $\mathcal{G}$, reaches a minimum, a maximum or simply remains constant; and $\langle c \rangle_{\mathcal{G}}$ stands for the time average of c between reinitializations.

3.4 About the fitness function

Although FPBIL is parameters-free, it still depends on the form of the fitness function. There are several functional forms for $\mathcal{F}$ capable of determining the same order $\mathcal{F}(\mathscr{I}^-) \le \mathcal{F}(\mathscr{I}_i) \le \mathcal{F}(\mathscr{I}_j) \le \cdot \cdot \le \mathcal{F}(\mathscr{I}^+)$ and each one of them can generate different $\mathscr{D}$ and ξ values, which would result in equally different behaviors. Consider the analysis of equations (8) and (9) in two simple examples:

1. With the transformation $\mathcal{F}'_i = f \cdot \mathcal{F}_i$ ($f \in \mathbb{R}$), one has $\mathscr{D}' = \mathscr{D}$ e $\xi' = \xi$, that is, the multiplication of the fitness by a constant factor, does not modify anything in the behavior of FPBIL.
2. With the transformation $\mathcal{F}'_i = t + \mathcal{F}_i$ ($t \in \mathbb{R}$), however, $\mathscr{D}' = \mathscr{D}$, but

   $$\xi' = \frac{\langle \mathcal{F} \rangle}{\langle \mathcal{F} \rangle + t}\,\xi, \tag{23}$$

   meaning that if $t \gg \langle \mathcal{F} \rangle$ then $\xi' \approx 0$, that is, the addition of the fitness to a constant term modifies the intensity of the steps of the FPBIL, making the FPBIL impracticable for big values of t.

Item 2 suggests that one good practice may be the use of the fitness $\mathcal{F}'_i = \mathcal{F}_i - \mathcal{F}(\mathscr{I}^-)$, guaranteeing that ξ will never be smaller than necessary. Following such recommendation, a generic procedure was adopted to construct the fitness—based on the procedure used by Koza in the genetic programming algorithm (Koza, 1992)—described as follows.

The raw fitness $\mathcal{F}_r$ is the natural amount of the problem that one desires to maximize or minimize, also called objective function. From the raw fitness, the standard fitness $\mathcal{F}_s$ is constructed, which possesses the characteristic of having $0 \leq \mathcal{F}_s(\mathscr{I}_a) < \mathcal{F}_s(\mathscr{I}_b)$ whenever $\mathscr{I}_a$ is better than $\mathscr{I}_b$, and, preferentially, with $\mathcal{F}_s(\mathscr{I}^+) = 0$. From the standard fitness, the adjusted fitness $\mathcal{F}_a$ is calculated from

$$\mathcal{F}_a(\mathscr{I}_i) = \frac{1}{1+\mathcal{F}_s(\mathscr{I}_i)}. \tag{24}$$

Finally, following the recommendation of having $\mathcal{F}_i' = \mathcal{F}_i - \mathcal{F}(\mathscr{I}^-)$, the fitness function used everywhere in this work (excep when expressly told) will be

$$\mathcal{F}(\mathscr{I}_i) = \begin{cases} \mathcal{F}_a(\mathscr{I}_i) - \mathcal{F}_a(\mathscr{I}^-_{\mathcal{G}-1}), & \text{if } \mathcal{F}_a(\mathscr{I}_i) - \mathcal{F}_a(\mathscr{I}^-_{\mathcal{G}-1}) \geq 0; \\ 0, & \text{if } \mathcal{F}_a(\mathscr{I}_i) - \mathcal{F}_a(\mathscr{I}^-_{\mathcal{G}-1}) < 0, \end{cases} \tag{25}$$

where $\mathcal{F}_a(\mathscr{I}^-_{\mathcal{G}-1})$ it is the adjusted fitness of the worse individual of the previous generation. The excuse for using $\mathcal{F}_a(\mathscr{I}^-_{\mathcal{G}-1})$ is that, to find $\mathcal{F}(\mathscr{I}^-)$, it is necessary to evaluate all the individuals of a generation, which implies that, in order to calculate $\mathcal{F}_i' = \mathcal{F}_i - \mathcal{F}(\mathscr{I}^-)$, all the individualsmust be evaluated twice every generation or all the individuals of a generation must be stored in some data structure. The adopted solution, besides economical, does not harm too much the original recommendation since generally $\mathcal{F}_a(\mathscr{I}^-_{\mathcal{G}-1}) \approx \mathcal{F}_a(\mathscr{I}^-)$.

Next, we will see how to put all this into practice.

4. Problems

This section is intended to show how PBIL and FPBIL behave in different problems of growing complexity. These problems belong to specific classes, which are, ultimately, numerical or combinatorial, so we can learn how to proceed in both cases. Besides the opportunity to see how these two algorithms works in practice, we will use the results then achieved to quantitatively compare them and, whenever interesting, compare their results to those of other techniques. Let us begin with the simplest.

4.1 A simple problem in $\mathcal{H}_2$

In order to visualize better the differences between FPBIL and PBIL, we will use them in a very simple problem: to find the greatest number in $\mathcal{B} = \mathbb{N}_4 \equiv \{1, 2, 3, 4\}$. We can chose $\mathcal{S}_{\mathcal{B}} = \mathcal{B}$, so that we need only $n = 2$ bits to cover all $\mathcal{S}_{\mathcal{B}}$ (because $2^2 = 4$ = number of elements in $\mathcal{S}_{\mathcal{B}}$)—FPBIL and PBIL will work in $\mathcal{H}^2$, which is nothing but a simple (easy to visualize) square. That means that we can correspond each point of $\breve{\mathcal{H}}_2$ to a member of $\mathcal{S}_{\mathcal{B}}$. We may choose, for example, $\mathcal{M}^n_{\mathcal{S}_{\mathcal{B}}}$ to be the following map:

$$\mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(0,0) = 1; \mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(0,1) = 3; \mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(1,0) = 4; \mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(1,1) = 1. \tag{26}$$

Given the simplicity of this problem and the fact that we are, in this first moment, more interested in seeing what happens inside the hypercube, a few simplifications will be done: we will fix the population size; there will be no reinitializations; and the fitness will be the raw fitness, which we choose to be:

$$\mathcal{F}(\mathscr{I}) = \mathcal{F}_r(\mathscr{I}) = \mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(\mathscr{I}). \tag{27}$$

We make two experiments. In the first, we fix the population size to be $\mathcal{P}$= 1, 000, 000, which compared to the size of $\mathcal{S}_{\mathcal{B}}$ can be considered to be infinite. The result is shown in figure 4. The contour lines represent constant values of $\mathscr{P}\langle\mathcal{F}\rangle$, according to equation (10) for the fitness defined in equation (27). The lines describe the movement of FPBIL's and PBIL's probability vectors.

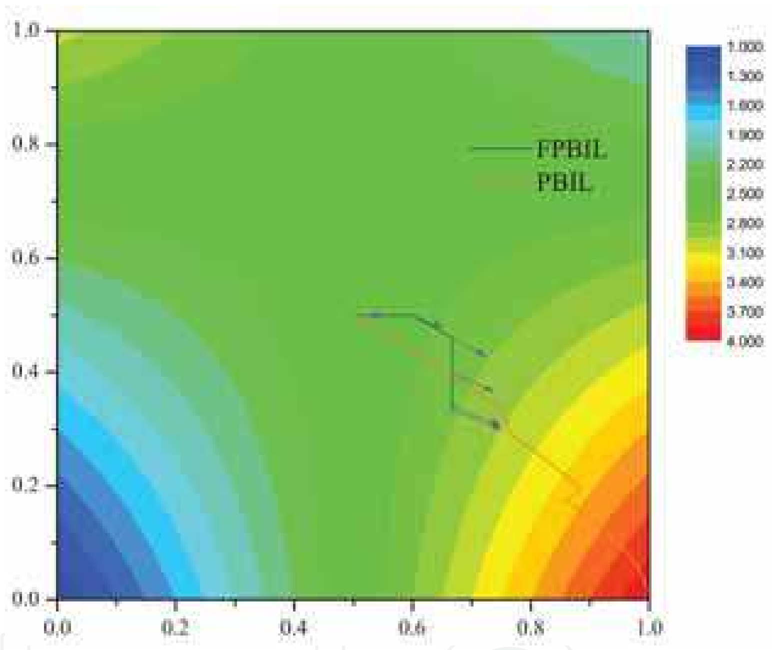


Fig. 4. Comparison between FPBIL and PBIL in $\mathcal{H}_2$; $\mathcal{P}$=1,000,000.

We see clearly that PBIL certainly finds the result to be "4", but FPBIL's line ends mysteriously. This is FPBIL's mutation in action. Since n = 2, the minimum value of d allowed is $d_n = d_2 = 1/3$—FPBIL's $\mathscr{P}$ can move only inside $[1/3, 2/3]^2$. This doesn't mean FPBIL can't find the result "4". In fact, from point (2/3, 1/3), the probability of getting the result "4" is 4/9, 2 times higher than the probability of getting "1" or "2" and 4 times higher than that of getting the result "3".The mutation in PBIL is more subtle and can be observed in the two sudden breaks suffered by PBIL's line.

We also highlight, in the same figure, the blue arrows which represent the gradient of $\mathscr{P}\langle\mathcal{F}\rangle$. Note that before the FPBIL's line reach the limits of $[1/3, 2/3]^2$, it (differently from PBIL's line) follows a direction very near from that of the gradient, which is just excellent, considering the discussion in section 3.1, meaning that in the limit of big values of $\mathcal{P}$, FPBIL's line can follow the gradient of $\mathscr{P}\langle\mathcal{F}\rangle$ to the optimum solution.

From this first experiment we are tempted to think PBIL is much better. But let us not forget this was an "almost infinite" population size experiment. In real applications we generally cannot span $\breve{\mathcal{H}}_n$completely (whenever we can, we surely will not need FPBIL). Hence, in the second experiment, we fix $\mathcal{P}$ = 2 (at maximum, half the elements of $\mathcal{S}_B$). The results are in figure 5. This time we see what generally happens in a real world problem. Both PBIL and FPBIL get more confused, but while FPBIL's mechanisms keep it doing its search inside $[1/3, 2/3]^2$, PBIL converges prematurely to a local optimum.
The next problem is, in a sense, a tougher version of this first.

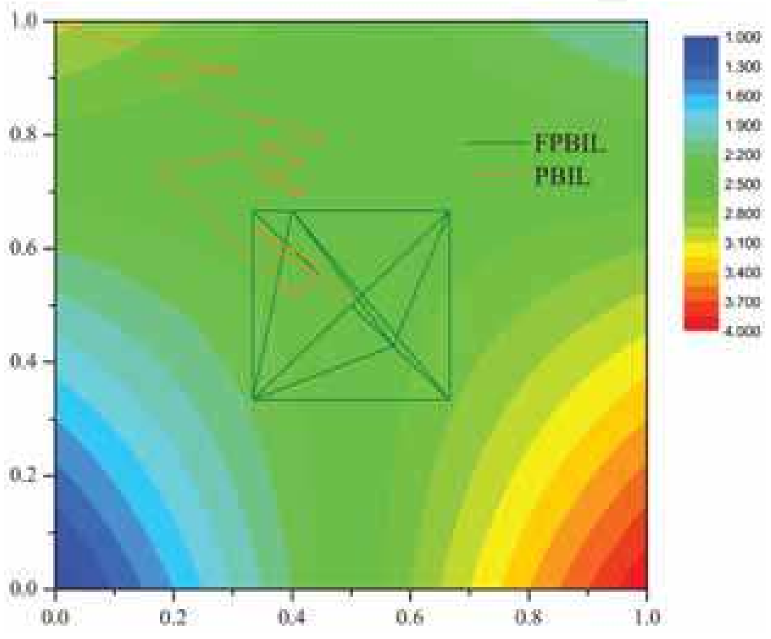


Fig. 5. Comparison between FPBIL and PBIL in $\mathcal{H}_2$; $\mathcal{P}$=2.

4.2 Banana

The banana problem consists in minimizing the Rosenbrocks function (Gill et al., 1981):

$$B(x, y) = 100\,(y - x^2)^2 + (1 - x)^2. \tag{28}$$

From a simple observation of the expression of this equation, we may conclude, without trouble, that a minimum of $B(x, y)$ occurs for $(x, y) = (1, 1)$. Also it is not difficult to show analytically that this is the only point where $B(x, y)$ becomes stationary. However, looking at the graph of $B(x, y)$ it is impossible to come to the same conclusion.

It is quite obvious the existence of a valley located at $y = x^2$, but finding the exact point of the valley where $B(x, y)$ is minimal is not simple at all. The difficulty in having such a view is due to the factor 100 that multiplies only $(y-x^2)^2$, leaving out the term $(1-x)^2$. Only when observed in a logarithmic scale, such as in figure 6, does the region where the minimum is located become apparent. The white line is a contour line that shows the banana shape, which names the problem.

The Rosenbrocks function have been classically used to test optimization algorithms, exactly because of the difficulty that this function imposes, especially for gradient-based search algorithms.

The set $\mathcal{S}_{\mathcal{B}} \subset \mathbb{R}$ to be codified into binary vectors, for the use of PBIL and FPBIL algorithms will be $[-4.194304; 4.194304)^2$, with a granularity of 0.000001 in both variables x and y. This means that each variable needs 23 bits to represent $\mathcal{S}_{\mathcal{B}}$, resulting in a total of 2^{46} = 70, 368, 744, 177, 664 possibilities.

More formally we have, with $n = 2 \cdot 23 = 46$,

$$\mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(\mathscr{I}) = (x, y), \tag{29}$$

With

$$x = -4.194304 + \frac{G_1(\mathscr{I})}{1000000}; \tag{30}$$

$$y = -4.194304 + \frac{G_2(\mathscr{I})}{1000000}, \tag{31}$$

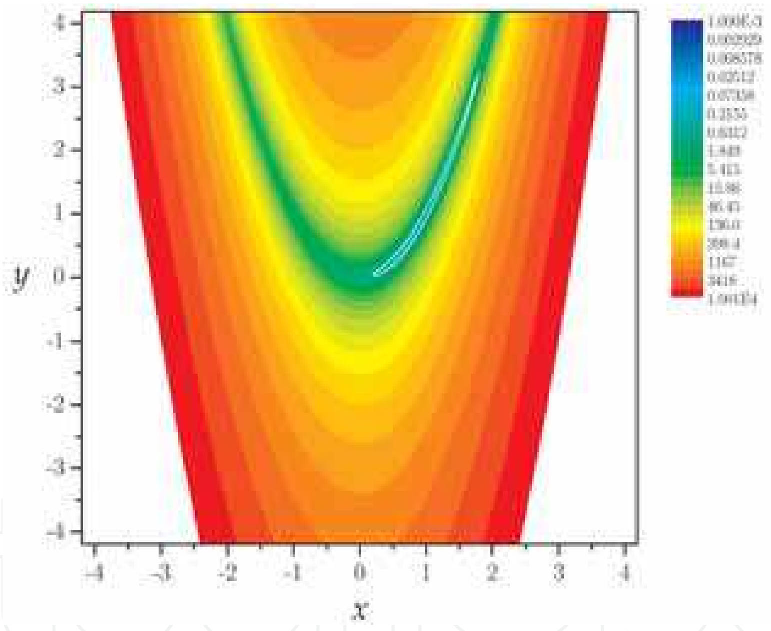


Fig. 6. Contour lines of $\log_{10} B(x, y)$. The white curve, in a banana shape, highlights the blue area where the minimum occurs.

Where $G_1(\mathscr{I})$ is the decoding of the first half of $\mathscr{I}$ and $G_2(\mathscr{I})$, of the second, both using Gra code[2] (Knuth, 2002). The fitness function of the banana problem used in this work is simply $B(x, y)$:

$$\mathcal{F}_b(\mathscr{I}_i) = B(x, y) = B \circ \mathcal{M}^{n}_{\mathcal{S}_{\mathcal{B}}}(\mathscr{I}_i). \tag{32}$$

[2] The use of Gray code may improve results considerably (Baluja, 1995).

Since we are dealing with a minimization problem and $\mathcal{F}_p(\mathscr{I}^+) = 0$, the standard fitness we use will be the raw fitness itself:

$$\mathcal{F}_p(\mathscr{I}_i) = B \circ \mathcal{M}^n_{\mathcal{S}_B}(\mathscr{I}_i). \tag{33}$$

In order to compare FPBIL to PBIL, we executed each algorithm 100 times and computed the average of the corresponding best individuals after a number of fitness evaluations. The result is shown in figure 7. We can see that the initial advantage of PBIL is amply overcome in the last fitness evaluations (approximately by a factor of 10^6). PBIL stagnates after 2, 000 fitness evaluations while FPBIL keeps finding better results in a constant rate until the end.
The next problem is a classical one concerning evolutionary search algorithms based on bit vectors.

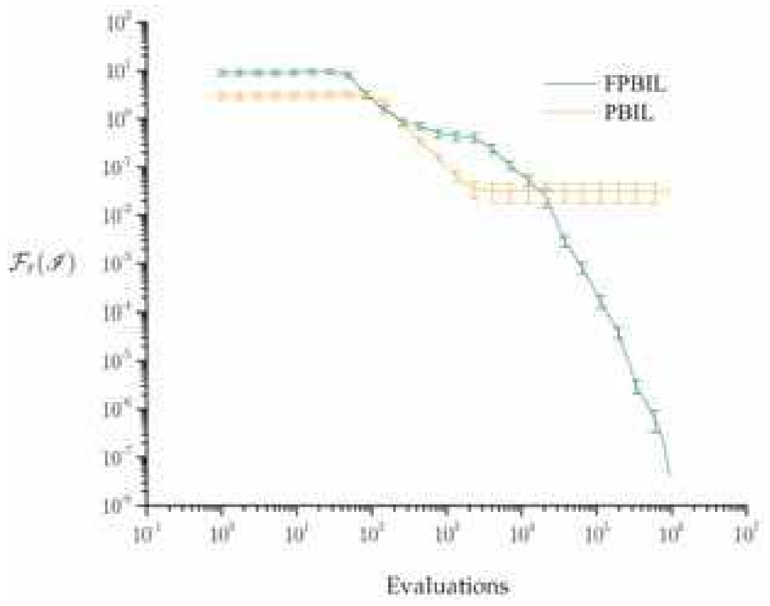


Fig. 7. Comparison between FPBIL and PBIL.

4.3 The four peaks problem

Consider the two functions defined on $\check{\mathcal{H}}_{100}$:

$$O(\mathscr{I}) = \text{number of contiguous 1's of } \mathscr{I} \text{ starting in position 1;} \tag{34}$$

$$Z(\mathscr{I}) = \text{number of contiguous 0's of } \mathscr{I} \text{ ending in position 100;} \tag{35}$$

where, for example, $O(011 \cdots 111) = 0$, $O(111 \cdots 111) = 100$, $Z(111 \cdots 110) = 1$ and $Z(000 \cdots 010) = 1$. Consider also the reward function

$$R(Z(\mathscr{I}),O(\mathscr{I});T) = \begin{cases} 100+T, & \text{if} \quad Z(\mathscr{I}) \geq T \quad \text{and} \quad O(\mathscr{I}) \geq T; \\ 0, & \text{otherwise.} \end{cases} \tag{36}$$

defined on $\{0, 1, 2, \ldots, 100\}^2 \times \{0, 1, 2, \ldots, 50\}$. In the four peaks problem, the objective is to maximize the function

$$F_T(\mathscr{I}) = \max\{Z(\mathscr{I}), O(\mathscr{I})\} + R(Z(\mathscr{I}), O(\mathscr{I}); T). \tag{37}$$

Observing F_T's plot in figure 8, one perceives that the four peaks problem is highly deceptive. There are two regions. One rewarded, corresponding to the upper surface, and another one, not rewarded, corresponding to the lower one. No point of the not-rewarded region (which increases with T) supplies us with any indication of the existence of the reward, giving the wrong impression of the existence of just peaks P_1 and P_2—corresponding to $F_T(\mathscr{I}) = 100$—while there still are the peaks P_3 and P_4—corresponding to $F_T(\mathscr{I}) = 200$, the global optimums.

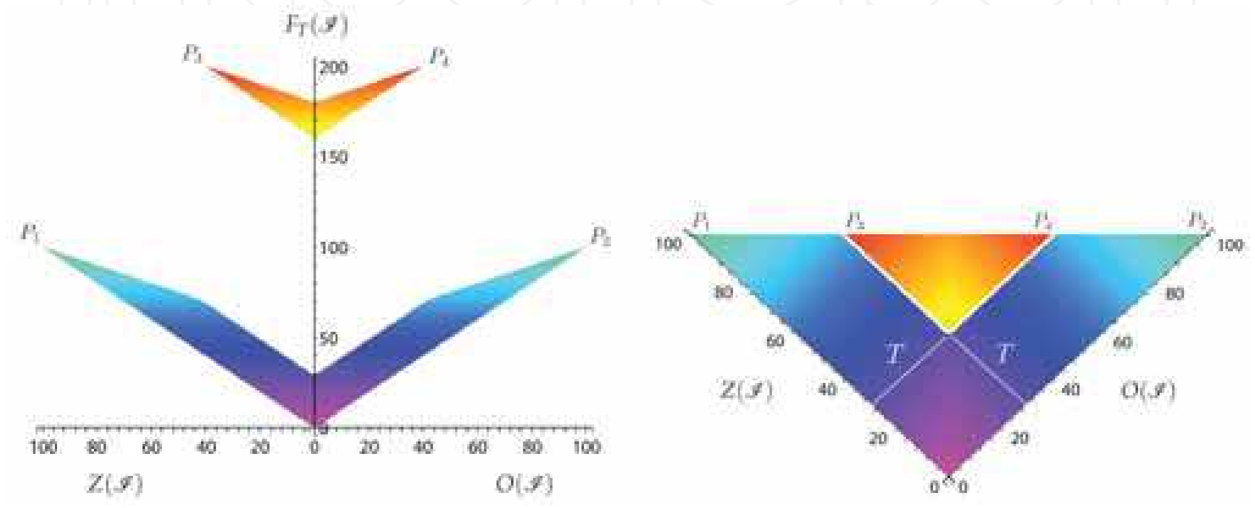


Fig. 8. Plot of $F_T(\mathscr{I})$, the objective function of the four peaks problem.

All the tests of the four peaks problem , carried through in this work have had $T = 30$ corresponding to a great bigger difficulty than the maximum difficulty used in (Baluja & Caruana, 1995), when, amongst a 25 total executions, the PBIL prematurely converged 20 times (the best result) and the genetic algorithms, between 22 and 25 times.

The raw fitness used in the four peaks problem was simply the value of $F_T(\mathscr{I})$: $\mathcal{F}_r(\mathscr{I}_i) = F_T(\mathscr{I}_i)$. Since one is dealing with a maximization problem and $\mathcal{F}_s(\mathscr{I}^+) = 200$, the standard fitness was $\mathcal{F}_s(\mathscr{I}_i) = 200 - F_T(\mathscr{I}_i)$. Figure 9 shows the comparison between FPBIL and PBIL, where the averages of the best fitness, after a number of fitness evaluations, are plotted for each algorithm. In 100 runs, PBIL was not able to reach the rewarded region, while the FPBIL did it every time, having as worst result $\mathcal{F}_r(\mathscr{I}_i) = 178$.

In the four peaks problem, the observation of the probability vector's evolution gives avery interesting insight into the algorithms. Figure 10, for example, illustrates a typical FPBIL run. It can be very clearly seen that during the first 1, 000, 000 fitness evaluations there were 4 reinitializations. After the second reinitialization, around the 2000th generation, FPBIL clearly reaches the global optimum. The PBIL, on the other hand, as shown in figure 11, converges, by the 2000th generation, to P_2. It is also worth noting the occurrence of mutation in PBIL. The white region corresponds to the probability vector's component equal to 1. The many red spots are the effects of mutation on the several components, making them change toward the value 0.5.

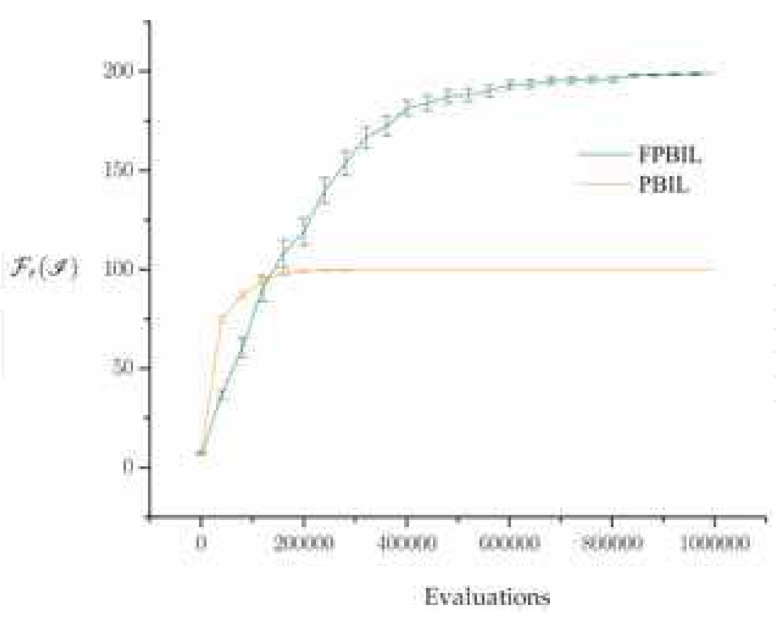


Fig. 9. Comparison between FPBIL and PBIL.

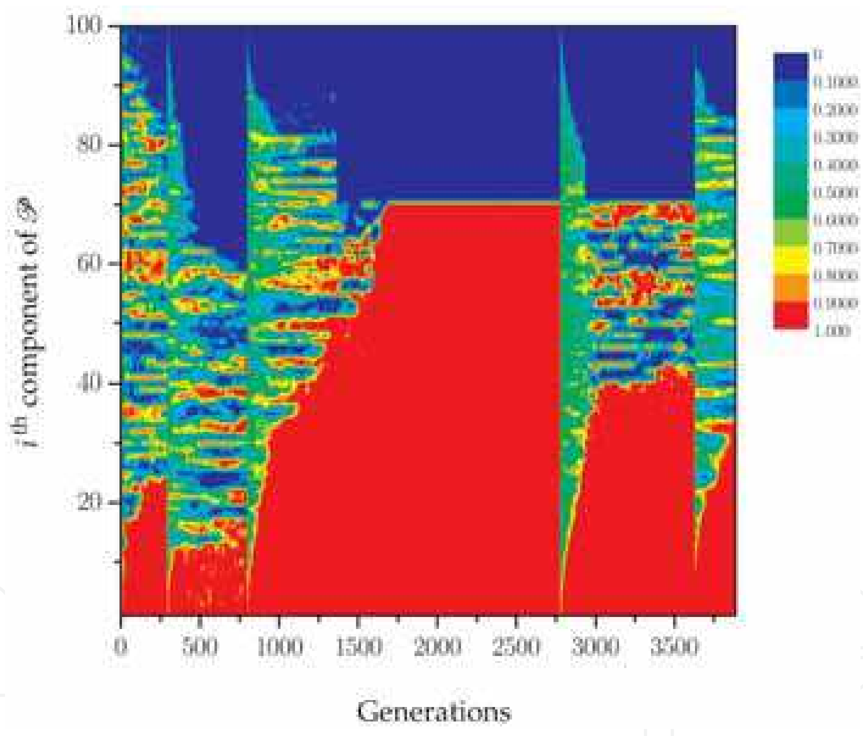


Fig. 10. Typical evolution of the probability vector in FPBIL.

4.4 TSP Rykel48

A traveling salesman must visit N cities, returning, in the end, to the city of origin, so that no city is visited twice. There are several possible routes (for $N > 2$). In fact, the number of routes is $(N-1)!$. The traveling salesman problem (TSP) consists in finding the shortest route.

The TSP is a NP problem, meaning that there is not yet an algorithm of polynomial order that can solve it. TheNP class can be considered as an intermediary computational complexity class, between classes P and EXP; as only a great amount of combinations is responsible for the demand of time (Lewis & Papadimitriou, 2000), the evaluation of each combination is usually the easy part.

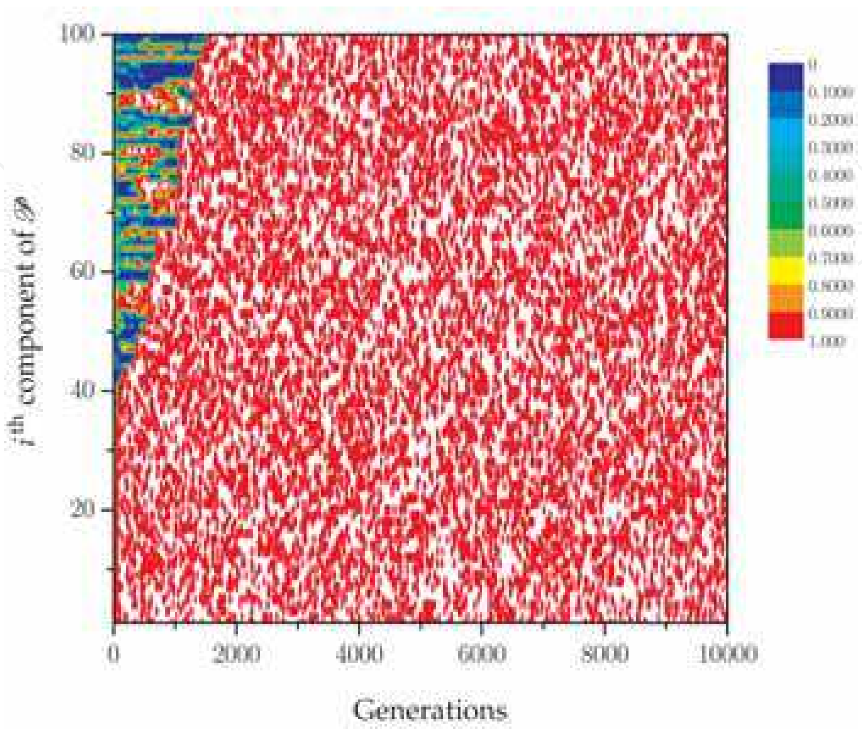


Fig. 11. Typical evolution of the probability vector in PBIL.

Rykel48 (TSPLIB, 2006) is a asymmetrical TSP with 48 cities resulting in a total of 258,623, 241,511,168,180,642,964,355,153,611,979,969,197,632,389,120,000,000,000 possible routes. In an asymmetric TSP the distance from one city A to another city B may be different from the distance from B to A, modeling, perhaps, single handed roads. Although the symmetric and asymmetric TSPs share the same number of routes (for the same amount of N), the asymmetry mixes up the search space topology, resulting in more complexes TSPs.
An important difference between Rykel48 TSP and the former problems is that the restriction that no city can be visited more than once prevents the direct codification of routes into bit vectors. The routes must be represented in an indirect way. In this work, we used the random keys representation (Bean, 1994; Caldas, 2006).
The Rykel48 TSP's raw fitness used in this work was simply the length of each route C_i corresponding to individual $\mathscr{I}_i$:

$$\mathcal{F}_b(\mathscr{I}_i) = C_i. \tag{38}$$

Since it is a minimization problem, we could have $\mathcal{F}_p(\mathscr{I}_i) = \mathcal{F}_b(\mathscr{I}_i)$. But since $\mathcal{F}_p(\mathscr{I}^+) = 14,422 \neq 0$, the standard fitness used will be

$$\mathcal{F}_p(\mathscr{I}_i) = C_i - 14,422. \tag{39}$$

Figure 12 shows the result. As it can be seen, FPBIL keeps the lead formost of its execution, especially in the latest 500,000 fitness evaluations.

Figure 13 shows the minimum and maximum values found after a number of the algorithms execution. The shortest route found by FPBIL was 14, 674, only 1.75% higher than the global optimum. Note that the PBIL presented a greater dispersion around the average.
At this point, it must be emphasized that the route length 14, 422 is not easily reached by any general purpose search algorithm. For example, the genetic algorithms only reach values close to 16, 500 (Machado, 1999) and the algorithms based on ant colonies—designed specifically to find smaller routes—achieve the optimum value only when processed in parallel, even so, only when assisted with heuristics (de Lima, 2005). Fig. 13 shows that PBIL is capable of reaching values just below 15, 000. The fact that FPBIL finds routes with the length of 14, 674 is a remarkable achievement.

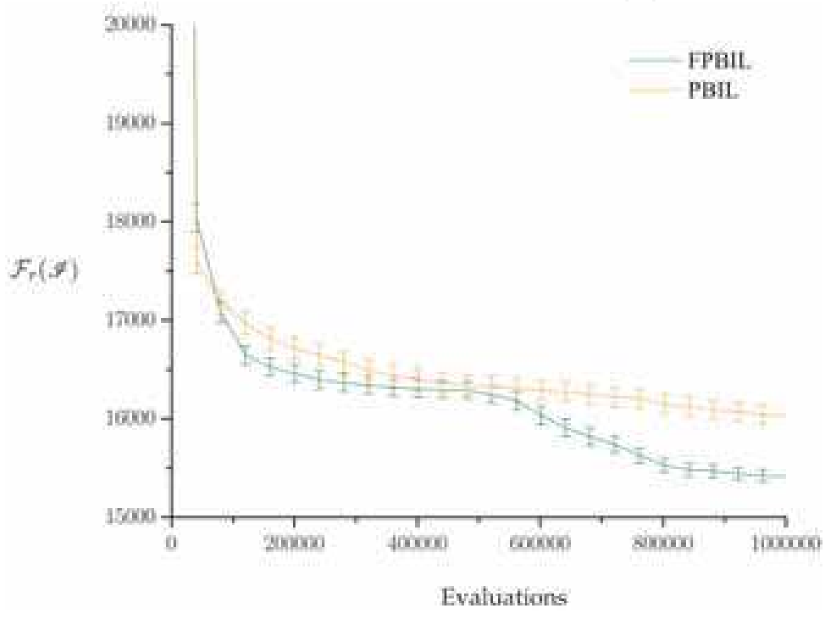


Fig. 12. Comparison between FPBIL and PBIL.

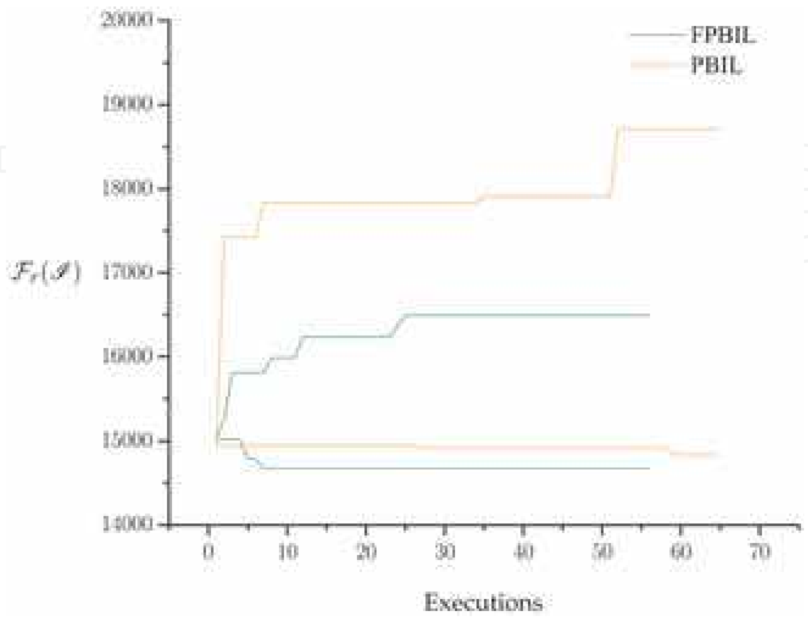


Fig. 13. Maximum and minimum values after a number of executions.

5. Conclusion

It can be affirmed, in conclusion to this chapter, that the FPBIL is an evolutionary algorithm, competitive with the best current optimization techniques, compact, relatively modest in the use of computational resources—like PBIL—, well founded, efficient, robust, self-adaptable, simple and parameterless.

Furthermore, the examples show that the FPBIL is efficient at both numerical and combinatorial problems. Here we should highlight the Four Peaks Problem, a highly deceptive problem handled very well by FPBIL.

FPBIL is conceptually simple and intuitive, since it does not require much sophisticated knowledge; it is compact, in the sense that it can be programmed with a few lines of code; and uses little amount of memory, since there is no need to store individuals of a population in some data structure.

The radically different way the mutation is handled in FPBIL is based on the probabilitie distribution inherent of the probability vector itself. This is updated using all the available information in each generation. These modifications enable the FPBIL to acquire self-adjustable features—such as the mechanism of variable population size—making the algorithm more efficient and more robust. Efficient in the sense that it finds solutions in less time; robust, meaning it has more resources to escape from local optimums.

With the proposition of FPBIL, we expect to have added relevant theoretical and practical tools, presenting feasible improvements with a considerable economic return, in both cost and benefit.

There still are, however, improvements which might be incorporated into FPBIL. After escaping from a local optimum, the FPBIL tends to approach the global optimum more slowly than other algorithms—PBIL, for example. Considering the process as a whole, the FPBIL takes advantage (since PBIL get caught more easily), but maybe it is possible to combine FPBIL with some other fast search algorithm, resulting in an even more efficient algorithm.

Other improvements can appear by constructing a multi-objective FPBIL— adapting the techniques from (Machado, 2005)—or even a parallel FPBIL—based on the techniques of (de Lima, 2005). One can still try to incorporate some kind of heuristic to the FPBIL perhaps some described in (de Lima, 2005). Works in these directions prove that these complementary techniques tends to produce better solutions.

6. Acknowledgments

In first place, I must thank God for all the great things he has done for me; for giving me strength and wisdom to conclude this work.

I also thank the invitation done by Dr. Vedran Kordic who started all of this; who has shown always very interested and helpful in his cordial emails.

Rubem and Lucia Caldas contributed with a fat slice of the costs for the publication of this material. I Thank them very much for this and also for the affection always demonstrated.

I would like to make a special thanks to Rubem Natan, Gisele Flores and Walter Aiken III, for the time they have spent trying to eliminate the maximum number of errors that I hid inside this chapter while I was preparing it. You guys are great!

I thank also all my colleagues from CNEN and COPPE/UFRJ for the support, specially those guys from room 429. You are insane!
I thank a lot my family, which have always encouraged and loved me. Rubem, Lúcia, Mrs. Rose, Mr. Antônio, Gisele, Elso, Ricardo, Anna, Luciana, Rosimar, Alexandre, Antônio Jr. and Tarsila. In particular, I would like to thank my wife, Renata, who promptly changed our vacation for this project. Love you! (not because of that!!!).
I also wanna thank my newest group of friends: Elthom, Susanne, Gustavo, Graciana, Victor and Leilane. You guys have been very important to me lately. You rock!

7. References

Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. Rep. CMU-CS-94-163, Pittsburgh, PA. URL http://citeseer.ist.psu.edu/baluja94population.html

Baluja, S. (1995). An empirical comparison of seven iterative and evolutionary function optimization heuristics. Tech. Rep. CMU-CS-95-193. URL http:// citeseer.ist.psu.edu/baluja95empirical.html

Baluja, S., & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In A. Prieditis, & S. Russel (Eds.) *The Int. Conf. on Machine Learning 1995*, (pp. 38–46). San Mateo, CA: Morgan Kaufmann Publishers. URL http://citeseer.ist.psu.edu /baluja95removing.html

Baluja, S., & Davies, S. (1998). Fast probabilistic modeling for combinatorial optimization. In *AAAI/IAAI*, (pp. 469–476). URL http://citeseer.ist.psu.edu/baluja98fast.html

Bean, J. C. (1994). "genetic algorithms and random keys for sequencing and optimization". *ORSA Journal on Computing*, *6*(2).

Caldas, G. H. F. (2006). *Algoritmo Evolucionário não Parametrizado Aplicado ao Problema da Otimiza¸cão de Recargas de Reatores Nucleares*. Ph.D. thesis, COPPE/UFRJ, Brazil.

de Lima, A. M. M. (2005). *Recarga de Reatores Nucleares Utilizando Redes Conectivas de Colônias Artificiais*. Ph.D. thesis, COPPE/UFRJ, Rio de Janeiro.

Gill, P. E., Murray, W., & Wright, M. H. (1981). *Practical Optimization*. San Diego: Academic Press.

Goldberg, D. E. (1989). *GENETIC ALGORITHMS in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. Massachusetts: MIT Press, second ed.

Juels, A., Baluja, S., & Sinclair, A. (1993). The equilibrium genetic algorithm and the role of crossover. URL http://citeseer.ist.psu.edu/juels93equilibrium.html

Knuth, D. E. (2002). *The Art of Computer Programming*. Stanford: Addison-Wesley, pre fascicle 2A ed.

Koza, J. R. (1992). Genetic Programming - On the Programming of Computers by Means of Natural Selection. Cambridge: MIT Press.

Lewis, H. R., & Papadimitriou, C. H. (2000). *Elementos de teoria da Computação*. Porto Alegre: Bookman, second ed.

Machado, M. D. (1999). *Um Novo Algoritmo Evolucionário com Aprendizado LVQ para a Otimização de Problemas Combinatórios como a Recarga de Reatores Nucleares*. Master's thesis, COPPE/UFRJ, Rio de Janeiro.

Machado, M. D. (2005). *Algoritmo Evolucionário PBIL Multi Objetivo Aplicado ao Problema da Recarga de Reatores Nucleares*. Ph.D. thesis, COPPE/UFRJ, Rio de Janeiro.

TSPLIB (2006). TSPLIB - a library of traveling salesman problem and related problem instances. URL http://nhse.cs.rice.edu/softlib/catalog/tsplib/tsp/