Evolutionary Programming Based on Non-uniform Mutation*

Xinchao Zhao[1], Xiao-Shan Gao[1] (Member, IEEE), and Zechun Hu[2,3]
[1]Institute of Systems Science, AMSS, Academia Sinica, Beijing, 100080, China
Email: (xczhao,xgao)@mmrc.iss.ac.cn
[2]Department of Mathematics, Nanjing University, Nanjing, 210093, P.R. China
[3]College of Mathematics, Sichuan University, Chengdu, 610064, P.R. China
Email: huzc@nju.edu.cn

Abstract–**A new evolutionary programming algorithm (NEP) using the non-uniform mutation operator instead of Gaussian or Cauchy mutation operators is proposed. NEP has the merits of "long jumps" of the Cauchy mutation operator at the early stage of the algorithm and "fine-tunings" of the Gaussian mutation operator at the later stage. Comparisons with the recently proposed sequential and parallel evolutionary algorithms are made through comprehensive experiments. NEP significantly outperforms the adaptive LEP for most of the benchmarks. NEP outperforms some parallel GAs and performs comparably to others in terms of the solution quality and algorithmic robustness. We give a detailed theoretical analysis of NEP. The probability convergence is proved. The expected step size of the non-uniform mutation is calculated. Based on this, the key property of NEP with "long jumps" at the early stage and "fine-tunings" at the later stage is proved strictly. Furthermore, the feature at the whole process of the algorithm, especially at the middle stage of it is appended.**

Index Terms: **Evolutionary programming, genetic algorithm, non-uniform mutation, global optimization, probability convergence, theoretical analysis.**

I Introduction

INSPIRED by the biological evolution and natural selection, intelligent computation algorithms are proposed to provide powerful tools for solving many difficult problems. Genetic algorithms (GAs) [2, 3], evolutionary strategies (ESs) [4], and the evolutionary programming (EP) [5, 21] are especially noticeable among them. In GAs, the crossover operator plays the major role and the mutation is always seen as an assistant operator. In ESs and EP, however, the mutation has been considered as the main operator. GAs usually adopt a high crossover probability and a low mutation probability, while ESs and EPs apply mutation to every individual. In binary GAs, one, two, multi-point, or uniform crossover and uniform mutation [1, 3] are often used. Some new mutation operators are proposed recently, such as the { frame-shift} and {translocation} operators [22], the transposition operator [18], etc. For real-coded GAs, the non-uniform mutation operator [1] is introduced. Besides the Gaussian mutation [5, 21], self-adaptation mutations [11, 12], self-adaptation rules from ESs [13], Cauchy [14] and Lévy-based [15] mutations are also incorporated into evolutionary programming. These new operators greatly enhance the performance of the algorithms.

In this paper, a new evolutionary programming algorithm (abbr. NEP) using the non-uniform mutation instead of Gaussian or Cauchy mutations is proposed. This work is inspired by the following observations. First, Yao et al [14, 15] argued that "higher probability of making longer jumps" is a key point that FEP and LEP perform better than CEP. However, "longer jumps" are detrimental if the current point is already very close to the global optimum. Second, the non-uniform mutation operator introduced in [1] has the feature of searching the space uniformly at the early stage and very locally at the later stage. In other words, the non-uniform mutation has the common merits of "higher probability of making far long jumps" at the early stage and "much better local fine-tuning ability" at the later stage. In [1], the non-uniform mutation operator is used in GAs by Michalewicz. As we mentioned before, the mutation operator is generally seen as an assistant operator in GAs. While in NEP, the mutation operator is treated as the major operator.

At the initial stage of NEP, the jumping steps are so long that they almost cover the whole search space (Section IV-A). The greedy idea and the idea of mutating a single component of the individual vector rather than modifying all the components are incorporated into the NEP algorithm in order to avoid possible random jumps and to ensure the algorithm "stays" at the promising area just found by the search engine.

Comparisons with the adaptive LEP demonstrate that

* Partially supported by a National Key Basic Research Project of China and by a USA NSF grant CCR-0201253.

NEP greatly outperforms the adaptive LEP for most of the separable and nonseparable, unidomal and multimodal benchmarks. Comparisons with five parallel genetic algorithms on high-dimensional benchmarks are also made. NEP performs much better than *R-DC* and *R-DS*, comparable to *ECO-GA*, and slightly worse than *CHC-GA* and *GD-RCGA*. Detailed introduction on these parallel GAs can be found in [6].

Convergence is an important issue in the theoretical study of EAs and many nice results [7, 8, 9, 10] have been obtained. Based on the Markov process theory, we prove that NEP with greedy selection is convergent in probability one. Theoretical analysis on how NEP works is also given based on the theories of stochastic process. First, the expected step size of the non-uniform mutation is calculated. Its derivative with respect to the generation variable t is less than zero, which implies the monotonously decreasing exploring region with the progress of the algorithm. Second, we obtain a quantitative description of the step size through analyzing the step size equation. Theoretical analysis also strongly supports the fact that NEP is not sensitive to the search space size [31].

According to Michalewicz [1], the parameter b in Eq.(2) of the non-uniform mutation determines the non-uniformity. Through theoretical analysis, we show that when b becomes larger, the step size of the mutation deceases faster. So an *adaptive NEP* is proposed based on different values of b. We apply several different mutations with different values of b for a parent and choose the best offspring for the next generation.

The rest of the paper is organized as follows. In Section II, we present the NEP algorithm and prove its convergence property. Comprehensive experiments are done to compare NEP with the recently proposed adaptive LEP [15] and five parallel genetic algorithms [6] in Section III. In Section IV, further theoretical analysis on the executing process of NEP is given. An adaptive NEP algorithm is proposed in Section V. In Section VI, conclusions are reached.

II Non-uniform Evolutionary Programming

In this section, we introduce an evolutionary programming algorithm based on the non-uniform mutation operator and prove its probability convergence.

A Non-uniform Mutation

Michalewicz [1] proposed a dynamical non-uniform mutation operator to reduce the disadvantage of random mutation in the real-coded GA. This new operator is defined as follows.

For each individual X_i^t in a population of t-th generation, create an offspring X_i^{t+1} through non-uniform mutation as follows: if $X_i^t = \{x_1, x_2, \ldots, x_m\}$ is a chromosome (t is the generation number) and the element x_k is selected for this mutation, the result is a vector $X_i^{t+1} = \{x'_1, x'_2, \ldots, x'_m\}$, where

$$x'_k = \begin{cases} x_k + \Delta(t, UB - x_k), & \text{if a random } \xi \text{ is } 0 \\ x_k - \Delta(t, x_k - LB), & \text{if a random } \xi \text{ is } 1 \end{cases} \tag{1}$$

and LB and UB are the lower and upper bounds of the variables x_k. The function $\Delta(t, y)$ returns a value in the range [0, y] such that $\Delta(t, y)$ approaches to zero as t increases. This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages. This strategy increases the probability of generating a new number close to its successor than a random choice. We use the following function:

$$\Delta(t, y) = y \cdot (1 - r^{(1-\frac{t}{T})^b}), \tag{2}$$

where r is a uniform random number from [0, 1], T is the maximal generation number, and b is a system parameter determining the degree of dependency on the iteration number.

B Simple Crossover in Evolutionary Programming

Similar to the binary genetic algorithm, we adopt the simple crossover. Let m be the dimension of a given problem and components of chromosomes $\mathbf{X}, \mathbf{Y}$ are all float numbers. Choose a pair of individuals:

$$\mathbf{X} = (x_1, \ldots, \mathbf{x_{pos_1}}, \ldots, \mathbf{x_{pos_2}}, \ldots, x_m)$$

$$\mathbf{Y} = (y_1 \ldots, \mathbf{y_{pos_1}}, \ldots, \mathbf{y_{pos_2}}, \ldots, y_m)$$

and randomly generate a decimal r. If $r < pc$ (crossover probability), apply simple two-point crossover to them as follows. Generate two random integers pos_1, pos_2 in the interval $[1, m]$. The components of two individuals between the numbers pos_1 and pos_2 will be exchanged. Then the new individuals are generated as follows.

$$\mathbf{X}' = (x_1, \ldots, \mathbf{y_{pos_1}}, \ldots, \mathbf{y_{pos_2}}, \ldots, x_m)$$

$$\mathbf{Y}' = (y_1, \ldots, \mathbf{x_{pos_1}}, \ldots, \mathbf{x_{pos_2}}, \ldots, y_m)$$

C Greedy Selection in NEP

We first give the definition of **neighborhood** of an individual.

Definition: [31] Given a vector $\mathbf{X} = (x_1, \ldots, x_i, \ldots, x_m)$ (m is the dimension of vectors), we call $\mathbf{X}'$ is its neighbor, *if and only if* one of its component is changed and other components remain unchanged. The neighborhood $\mathbf{N}$ of a vector $\mathbf{X}$ consists of all its neighbors. That is,

$$\mathbf{N} = \{\mathbf{X}' | \mathbf{X}' \text{ is a neighbor of } \mathbf{X}\}. \tag{3}$$

Different from the traditional local search which performs greedy local search until a local optimum is obtained, we only mutate every component *one time* in certain probability for each component of every real-coded individual (*vector*). The current individual will be replaced by the mutated one only when the new one is **not worse than** the current individual. This strategy can overcome the plateaus of constant fitness problem as indicated by Jansen and Wegener [17]. Such a greedy selection procedure for a current individual $\mathbf{X} = (x_1, \ldots, x_i, \ldots, x_m)$ is as follows.

```
For $i$ from 1 to the functional dimension $m$
    Mutate the $i$th component $x_i$ of $\mathbf{X}$ and obtain
    a new vector $\mathbf{X}'$ using Eqs.(1, 2);
    if fitness of $\mathbf{X}'$ is not worse than that of $\mathbf{X}$ then
        $\mathbf{X}'$ becomes the current individual;
End for.
```

D Non-Uniform Evolutionary Programming

For a function $f(\mathbf{X})$, NEP will find an

$$\mathbf{X}^* \text{ such that } \forall\ \mathbf{X},\ \ f(\mathbf{X}^*) \le f(\mathbf{X}). \tag{4}$$

The NEP algorithm adopts real encoding, two-point crossover and non-uniform mutation. The procedure of NEP is given as follows.

Procedure of NEP

1) Generate the initial population consisting of n individuals, each of which, $\mathbf{X_i^0}$, has m independent components, $\mathbf{X_i^0} = (x_1, x_2, \ldots, x_m)$.
2) Evaluate each individual based on the objective function, $f(\mathbf{X}_i^0)$.
3) Apply two-point crossover to the current population.
4) For each parental individual $\mathbf{X_i^t} = (x_1^t, \ldots, x_m^t)$.
    4.1) for each component x_j^t constructing $x_j^{t\prime}$ using Eq.(1,2) (new individual denoted as $\mathbf{X_i^t}'$)
    4.1.1) If $f(\mathbf{X_i^t}') \le f(\mathbf{X_i^t})$
        $\mathbf{X_i^t}'$ replaces $\mathbf{X_i^t}$ as the current individual.
5) Conduct pairwise comparison over the offspring population. For each comparison, q individuals are chosen uniformly at random from the offspring population. The fittest individual is put into the next generation.
6) Repeat steps 3-5, until the stopping criteria are satisfied.

End of Procedure

The crossover is firstly applied to the population in a probability 0.4, the non-uniform mutation operator followed at a mutation probability 0.6. The summation of the mutation and crossover probabilities is kept as **1**. Thus NEP will generate **equal number** of offsprings in the sense of probability comparing with other EPs. The parameter b in NEP remains unchanged during the evolution.

E Analysis on the Convergence of NEP

In this section, the convergence of NEP will be proved based on the stochastic process theory. Since NEP mutates single component only in once mutation operation, we only consider the one dimensional case, i.e., $n = 1$. In this case, there is no crossover and we need only consider the non-uniform mutation operation. We divide the objective functions into two classes in the following analysis.

1) Unimodal Functions

We assume that $f(x)$ has a unique minimal value at x^*. In Fig.(I), let x_0 be one initial solution, x_0' another initial solution lying between x^* and x_0, $\underline{x_0}$ a number satisfying $f(\underline{x_0}) = f(x_0)$ and $\underline{x_0} \ne x_0$, and ε an arbitrary small positive number. Without loss of generality, we assume that variable x lies on the right side of x^* and $\underline{x}$ lies on the left side.

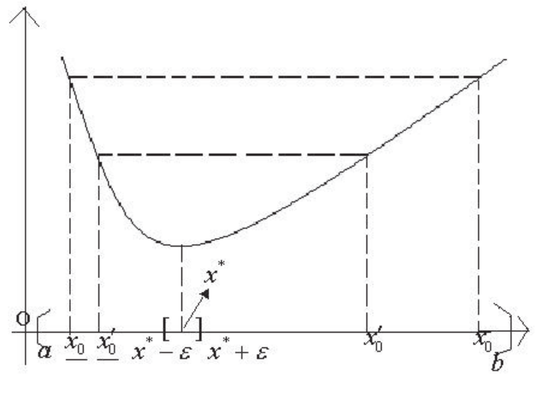


Figure I: Analysis on the Unimodal Function

Based on Eq.(1, 2) and via NEP algorithm, we have

$$x_1 = \begin{cases} x_0, \text{ if } \xi{=}0, \text{ or if } \xi{=}1 \text{ and } x_0 - \Delta(1, x_0 - a) \le \underline{x_0} \\ x_0 - \Delta(1, x_0 - a), \text{ if } \xi{=}1 \text{ and } x_0 - \Delta(1, x_0 - a) > \underline{x_0} \end{cases}$$

$$x_1' = \begin{cases} x_0', \text{ if } \xi{=}0, \text{ or if } \xi{=}1 \text{ and } x_0' - \Delta(1, x_0' - a) \le \underline{x_0'} \\ x_0' - \Delta(1, x_0' - a), \text{ if } \xi{=}1 \text{ and } x_0' - \Delta(1, x_0' - a) > \underline{x_0'} \end{cases}$$

Lemma 1: Let $p_1 = P\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$, $p_1' = P\{x_1{}' \notin (x^* - \varepsilon, x^* + \varepsilon)\}$. If $x^* < x_0' < x_0$ then $p_1' < p_1$. Similarly, if $x_0 < x_0' < x^*, p_1' < p_1$ also holds.

Proof: We have

$p_1 = 1 - P\{x_1 \in (x^* - \varepsilon, x^* + \varepsilon)\}$

$\quad = 1 - P\{\xi = 1,\ x^* - \varepsilon < x_0 - \Delta(1, x_0 - a) < x^* + \varepsilon)\}$

$\quad = 1 - \frac{1}{2} P\{x^* - \varepsilon < x_0 - \Delta(1, x_0 - a) < x^* + \varepsilon)\}$

Similarly,

$p_1^{'} = 1 - \frac{1}{2} P\{x^* - \varepsilon < x_0^{'} - \Delta(1, x_0^{'} - a) < x^* + \varepsilon)\}$

Let $q = P\{x^* - \varepsilon < x_0 - \Delta(1, x_0 - a) < x^* + \varepsilon)\}$

$\quad q^{'} = P\{x^* - \varepsilon < x_0^{'} - \Delta(1, x_0^{'} - a) < x^* + \varepsilon)\}$

Thus it is enough to show that

$$q < q^{'} \tag{5}$$

By Eq.(2) we have

$q = P\{x^* - \varepsilon < x_0 - \Delta(1, x_0 - a) < x^* + \varepsilon)\}$

$= P\{x^* - \varepsilon < x_0 - (x_0 - a)(1 - r^{(1-\frac{1}{T})^b}) < x^* + \varepsilon)\}$

$= P\{(\frac{x^*-a-\varepsilon}{x_0-a})^{\frac{1}{m}} < r < (\frac{x^*-a+\varepsilon}{x_0-a})^{\frac{1}{m}}\}$

$$= (\frac{x^* - a + \varepsilon}{x_0 - a})^{\frac{1}{m}} - (\frac{x^* - a - \varepsilon}{x_0 - a})^{\frac{1}{m}} \tag{6}$$

where $m = (1 - \frac{1}{T})^b$

Similarly, we have

$q^{'} = P\{x^* - \varepsilon < x_0^{'} - \Delta(1, x_0^{'} - a) < x^* + \varepsilon)\}$

$$= (\frac{x^* - a + \varepsilon}{x_0^{'} - a})^{\frac{1}{m}} - (\frac{x^* - a - \varepsilon}{x_0^{'} - a})^{\frac{1}{m}} \tag{7}$$

From Eq. (6, 7) and let q subtracts q', we may derive Eq. (5) and thus the correctness of the Lemma. ▮

Since x_0 is a given initial solution (individual), we can assume that $x_0 > x^*$. Let

$p_1^+ := P\{x_0$ is the intial solution and $x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$p_1^- := P\{\underline{x_0}$ is the intial solution and $x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

Then $p_1 := p_1^+$ (or p_1^-)

For $n \geq 2$, we define

$p_n^+ := P\{x_{n-1} > x^*,\ x_n \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$p_n^- := P\{x_{n-1} < x^*,\ x_n \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$p_n := p_n^+ + p_n^-$

Theorem 2: For any $\varepsilon > 0$, we have

$$\lim_{n \to \infty} p_n = 0$$

Proof: By the description of NEP, it is easy to know that the stochastic process $\{x_i,\ i = 0, 1, 2, \cdots\}$ is a Markov process. By the property of conditional expectation, Markov property and Lemma 1, we can obtain that

$p_2 = P\{x_2 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$\quad = E[I_{\{x_2 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}]$

$\quad = E(E[I_{\{x_2 \notin (x^* - \varepsilon, x^* + \varepsilon)\}} | x_1])$

$\quad = E(E^{x_1}[I_{\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}])$

$\quad = E^{x_1}[I_{\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}] \cdot P\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$\quad \leq \max\{p_1^+, p_1^-\} \cdot p_1$

$p_3 = P\{x_3 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$\quad = E[I_{\{x_3 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}]$

$\quad = E(E[I_{\{x_3 \notin (x^* - \varepsilon, x^* + \varepsilon)\}} | x_2])$

$\quad = E(E^{x_2}[I_{\{x_3 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}])$

$\quad = E^{x_2}[I_{\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}] \cdot P\{x_2 \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$\quad \leq \max\{E^{x_0}[I_{\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}], E^{\underline{x_0}}[I_{\{x_1 \notin (x^* - \varepsilon, x^* + \varepsilon)\}}]\}$
$\qquad \times p_2$

$\quad \leq (\max\{p_1^+, p_1^-\})^2 \cdot p_1$

By induction we have

$p_n = P\{x_n \notin (x^* - \varepsilon, x^* + \varepsilon)\}$

$\quad \leq (\max\{p_1^+, p_1^-\})^{n-1} \cdot p_1$

Obviously, $0 < p_1^+, p_1^- < 1$, so

$$\lim_{n \to \infty} p_n = 0$$

The proof is complete. ▮

By the greedy selection of NEP, it is easy to know that

$p_n = P\{x_i \notin (x^* - \varepsilon, x^* + \varepsilon),\ i = 1, 2, \ldots, n\}$

So Theorem 2 implies that for any $\varepsilon > 0$ we have

$$\lim_{n \to \infty} (1 - P\{x_i \notin (x^* - \varepsilon, x^* + \varepsilon),\ i = 1, \ldots, n\}) = 1$$

i.e.,

$$\lim_{n \to \infty} P\{\exists\ i = 1, \ldots, n,\ s.t.\ x_i \in (x^* - \varepsilon, x^* + \varepsilon)\} = 1 \tag{8}$$

Eq.(8) indicates that for any $\varepsilon > 0, \{x_i\}_{i=1}^{\infty}$ is to enter the domain $(x^* - \varepsilon, x^* + \varepsilon)$ almost surely, and so $\{x_i\}_{i=1}^{\infty}$ converges to x^* almost surely.

2) Multimodal Functions

We assume that $g(x)$ is a multimodal function with a minimal value at x^*. Without loss of generality, we assume that g has only one global optimum. Let $x_0, x_0^{'}$ be initial solutions (individuals).

Without loss of generality, suppose $x_0, x_0^{'}$ are two points on the left side of point "c" as in Fig. (II). Denote the offsprings of them as $x_1, x_1^{'}$ respectively. Now we will consider how to choose the interval points "c" and "d" which

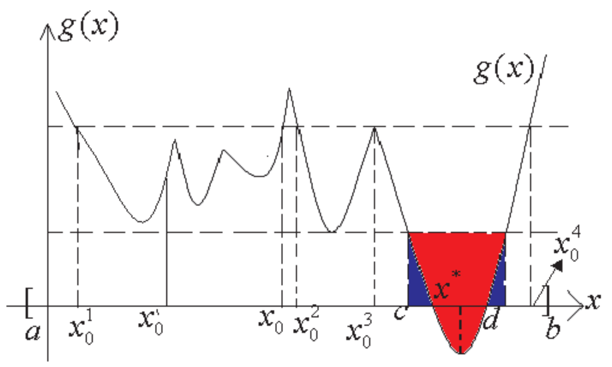


Figure II: Analysis on the Multimodal Function

satisfy the following conditions. First, $g(x)$ is unimodal in the subinterval of $[c,\ d]$. Second, we assume $g(c) = g(d)$. Third, there is no other local optimal region below the line through $g(c)$ and $g(d)$ on function $g(x)$. We have the following lemma.

Lemma 3: Let

$p_1 = P\{x_1(\omega) \notin (c,d), x_0 \text{ is the initial point}\}$,

$p_1^{'} = P\{x_1{'}(\omega) \notin (c,d), x_0^{'} \text{ is the initial point}\}$.

Then if $x_0 < x_0^{'}$ we have $p_1^{'} < p_1$.

Proof: We have

$p_1 = 1 - P\{x_1 \in (c,d), x_0 \text{ is the initial point}\}$

$= 1 - P\{\xi = 0,\ c < x_0 + \Delta(1, b - x_0) < d\}$

$= 1 - \frac{1}{2} P\{c < x_0 + \Delta(1, b - x_0) < d\}$

Similarly,

$p_1^{'} = 1 - \frac{1}{2} P\{c < x_0^{'} + \Delta(1, b - x_0^{'}) < d\}$

The proof is similar with the Lemma 1. ▮

Remark: This means that if the initial point is closer to (c,d), the probability that its offspring enters (c,d) is larger. Next, we want to establish a similar result as Theorem 2.

Let x_0 be a given initial individual as indicated in Fig.(II). Define

$A_0 := \{u \in [a,b] : g(u) = g(x_0)\}$

$x_0^- := \min\{u \in A_0 : u < c\}$

$x_0^+ := \max\{u \in A_0 : u > d\}$.

For example, $x_0^- = x_0^1, x_0^+ = x_0^4$ in Fig. (II). For $n \geq 1$, we define

$A_n(\omega) := \{u \in [a,b] : g(u) = g(x_n(\omega))\}$,

$x_n^- := \min\{u \in A_n(\omega) : u < c\}$,

$x_n^+ = \max\{u \in A_n(\omega) : u > d\}$.

Let $p_1^- := P\{x_1(\omega) \notin (c,d), x_0^- \text{ is the initial point}\}$;

$p_1^+ := P\{x_1(\omega) \notin (c,d), x_0^+ \text{ is the initial point}\}$;

For $n \geq 1$ (all the following equations should be understood under the condition that x_0 is the beginning point), we define

$p_n = P\{x_n(\omega) \notin (c,d)\}$.

Theorem 4: We have that

$$\lim_{n\to\infty} p_n = 0$$

Proof. By the procedure of NEP, we know that the stochastic process $\{x_i, i = 0, 1, \cdots\}$ is a Markov process. By the property of conditional expectation, Markov property and Lemma 3 we can obtain that

$$\begin{aligned}
p_2 &= P\{x_2(\omega) \notin (c,d)\} \\
&= E[I_{\{x_2(\omega)\notin(c,d)\}}] \\
&= E(E[I_{\{x_2(\omega)\notin(c,d)\}} | x_1(\omega)]) \\
&= E(E^{x_1(\omega)}[I_{\{x_1(\omega)\notin(c,d)\}}]) \\
&= E^{x_1(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}) \cdot I_{\{x_1(\omega)\notin(c,d)\}} + \\
&\quad E^{x_1(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}) \cdot I_{\{x_1(\omega)\in(c,d)\}} \\
&= E^{x_1(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}) \cdot I_{\{x_1(\omega)\notin(c,d)\}} \\
&\leq \max\{E^{x_0^-}(I_{\{x_1(\omega)\notin(c,d)\}}), E^{x_0^+}(I_{\{x_1(\omega)\notin(c,d)\}})\} \\
&\quad \times P\{x_1(\omega) \notin (c,d)\} \\
&\leq p_1 \cdot \max\{p_1^+, p_1^-\}
\end{aligned}$$

Similarly we have

$$\begin{aligned}
p_3 &= P\{x_3(\omega) \notin (c,d)\} \\
&= E[I_{\{x_3(\omega)\notin(c,d)\}}] \\
&= E(E[I_{\{x_3(\omega)\notin(c,d)\}} | x_2(\omega)]) \\
&= E(E^{x_2(\omega)}[I_{\{x_1(\omega)\notin(c,d)\}}]) \\
&= E^{x_2(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}) \cdot I_{\{x_2(\omega)\notin(c,d)\}} + \\
&\quad E^{x_2(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}) \cdot I_{\{x_2(\omega)\in(c,d)\}} \\
&= E^{x_2(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}) \cdot I_{\{x_2(\omega)\notin(c,d)\}} \\
&\leq \max\{E^{x_1^-(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}}), E^{x_1^+(\omega)}(I_{\{x_1(\omega)\notin(c,d)\}})\} \\
&\quad \times P\{x_2(\omega) \notin (c,d)\} \\
&\leq \max\{E^{x_0^-}(I_{\{x_1(\omega)\notin(c,d)\}}), E^{x_0^+}(I_{\{x_1(\omega)\notin(c,d)\}})\} \times p_2 \\
&\leq\ p_1 \cdot (\max\{p_1^+, p_1^-\})^2
\end{aligned}$$

By induction we have

$$\begin{aligned}
p_n &:= P\{x_n(\omega) \notin (c,d)\} \\
&\leq p_1 \cdot (\max\{p_1^+, p_1^-\})^{n-1}
\end{aligned}$$

By the procedure of NEP, we know that $0 \leq p_1^+, p_1^- < 1$ (if $x_0 \in (c,d)$, then obviously we have $p_1^+ = p_1^- = 0$). Thus we have

$$\lim_{n\to\infty} p_n = 0.$$

The proof is complete. ∎

By greedy selection of NEP, it is easy to know that $p_n = P\{x_i(\omega) \notin (c,d), i = 1,2,\cdots,n\}$. And by Theorem 4, we know that

$$\lim_{n\to\infty} (1 - P\{x_i(\omega) \notin (c,d), i = 1,2,\cdots,n\}) = 1,$$

i.e.

$$P\{\exists i = 1,2,\cdots,n, \text{ such that } x_i(\omega) \in (c,d)\} = 1$$

Obviously, the function (g(x), $x \in [c\ d]$) is a unimodal function. So from Theorem 2 we know that

$$P\{\omega : \ x_i(\omega) \text{ converges to } x^*\} = 1 \tag{9}$$

Remark: Similarly, it is easy to reach the same conclusions for the high-dimensional problems because algorithm NEP only mutates one component of a vector (individual) per mutation operation.

III Experiments and Analysis

In our algorithm, we set the simple crossover probability $pc = 0.4$ and the non-uniform mutation probability $pm = 0.6$ in all cases. Following Michalewicz [1], the parameter b in Eq.(2) is set to be 5. The population size and the maximal evolutionary generation numbers will vary for the comparing algorithms. To make the comparison fair in terms of computing time, the population size of NEP is reduced proportionally according to the problem dimensions, since each individual in NEP generates m (dimension of function) offsprings. For example, for a four dimensional function, if the population size of CEP is 100 then the population size of NEP will be 25. It is worth pointing out that NEP actually uses less computing time, because operations, such as selection, use less time in a small population. The programs are implemented with the **C** programming language.

A Comparison with Adaptive LEP [15]

Yao [14] et al. proposed an evolutionary programming algorithm (FEP) based on the *Cauchy* probability distribution. Lee and Yao [15] proposed an evolutionary programming algorithm (LEP) based on the Lévy probability distribution. The *Cauchy* probability distribution is a special case of the Lévy probability distribution and the adaptive LEP performs at least as well as the nonadaptive LEP with a fixed α [15]. So we just compare our algorithm with the adaptive LEP.

1) Benchmark Functions and Parameters Setting

We use the same test functions with the adaptive LEP which can be found in Table I or [15]. The large number of benchmarks is necessary as Wolpert and Macready [19] have shown that under certain assumptions no single search algorithm is best on average for all problems. If the number of benchmarks is too small, it would be very difficult to make a generalized conclusion and have the potential risk that the algorithm is biased toward the chosen problems. Among them, f_1, f_2, f_3 are high-dimension and unimodal functions, $f_4, \ldots, f_9$ are mutilmodal functions where the number of local minims increases exponentially with the augment of the problem dimensions and are the most difficult class of problems for many optimization algorithms [14]. $f_{10}, \ldots, f_{14}$ are low-dimensional functions which only have a few local minima. Some properties of several benchmarks are listed below [6].

- f_3 is a continuous and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom and the nonlinear interactions between the variables, i.e., it is *nonseparable* [20]. These features make the search direction have to continually change to reach the optimum. Experiments show that it is even more difficult than those multimodal benchmarks.
- The difficulty of f_2 concerns the fact that searching along the coordinate axes only gives a poor rate of convergence since its gradient is not oriented along the axes. It presents similar difficulties with f_3, but its valley is narrower.
- f_7 is difficult to optimize because it is nonseparable and the search algorithm has to climb a hill to reach the next valley [20].

The maximal evolutionary generation numbers are set to be the same to adaptive LEP (1500 for functions $f_1, \cdots, f_9$, 30 for f_{10}, f_{11} and 100 for $f_{12}, \cdots, f_{14}$). Due to the population size of the adaptive LEP being 100 and each individual generates four offsprings at every generation, the population size of the adaptive LEP is equivalent to 400. Consequently, we set the population size of NEP to be an integer less than $\frac{400}{Dimensions\ of\ Problem}$ (dimension is 30 for high dimensional functions) which are 13 for functions $f_1, \ldots, f_9$, 200 for f_{10}, f_{11} and 100 for f_{12}, f_{13}, f_{14}. A computational precision of **50** digits after point is used in NEP. So a result **being 0** means that it is **less that** 10^{-50} in NEP and vice versa in this subsection. We do not find the computational precision demand of [15].

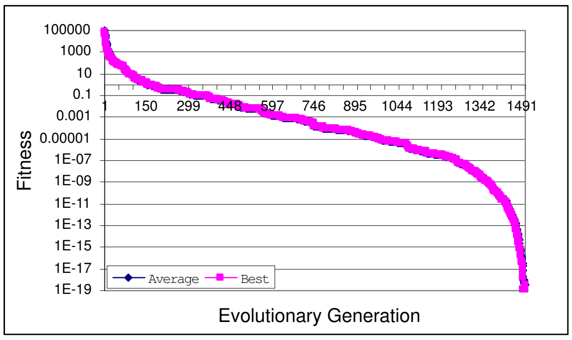
2) Performance Comparison and Analysis

Comparisons between NEP and the adaptive LEP are given in Figures III-V and Table II which includes the average re-

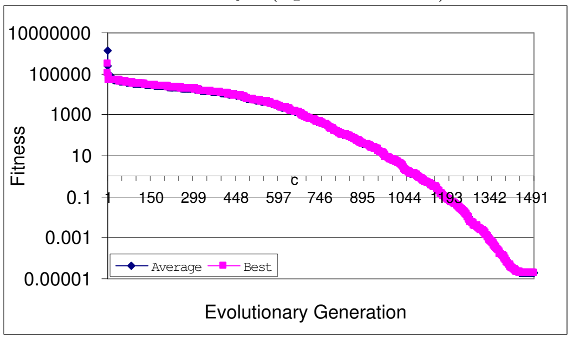
Benchmark Functions	n	**D**	f_{min}
high-dimension unimodal functions			
$f_1 = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
$f_2 = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	30	$[-100, 100]^n$	0
$f_3 = \sum_{i=1}^{n-1} [100(x^{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^n$	0
high-dimension multimodal functions with many local minima			
$f_4 = -\sum_{i=1}^{n} (x_i \sin(\sqrt{\lvert x_i \rvert}))$	30	$[500, 500]^n$	-12569.48
$f_5 = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]^n$	0
$f_6 = -20\exp[-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}]-$ $\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$ +20+e	30	$[-32, 32]^n$	0
$f_7 = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	30	$[-600, 600]^n$	0
$f_8 = \frac{\pi}{n}\{10\sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})]$ $+(y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4), y_i = 1 + \frac{1}{4}(x_i + 1)$	30	$[-50, 50]^n$	0
$f_9 = 0.1\{10\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + 10\sin^2(3\pi x_{i+1})]$ $+(x_n - 1)^2[1 + sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$	0
low-dimension functions with only a few local minima			
$f_{10} = (4 - 2.1x_1^2 + \frac{1}{3}x_1^4)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$	2	$[-5, 5]^n$	-1.0316
$f_{11} = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 -$ $14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times$ $(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2, 2]^n$	3
$f_{12} = -\sum_{j=1}^{5}[\sum_{i=1}^{4}(x_i - a_{ij})^2 + c_j]^{-1}$	4	$[0, 10]^n$	-10.1532
$f_{13} = -\sum_{j=1}^{7}[\sum_{i=1}^{4}(x_i - a_{ij})^2 + c_j]^{-1}$	4	$[0, 10]^n$	-10.40282
$f_{14} = -\sum_{j=1}^{10}[\sum_{i=1}^{4}(x_i - a_{ij})^2 + c_j]^{-1}$	4	$[0, 10]^n$	-10.53628

Table I: Benchmark functions used by LEP & NEP

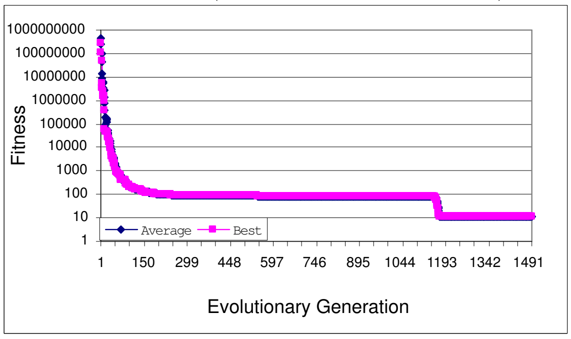
sult for 50 independent runs. Figures III-V show the evolutionary behaviors of the best and average fitness versus the evolutionary generations. Due to the small population size (13) of high-dimensional functions, the plots of the average and best fitness nearly completely overlap each other.



f_1 (Sphere Model)



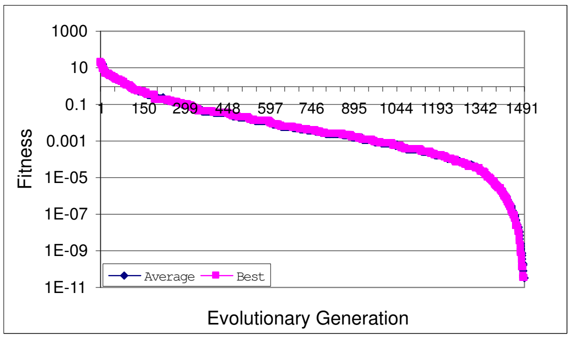
f_2 (Schwefel's Problem 2.22)



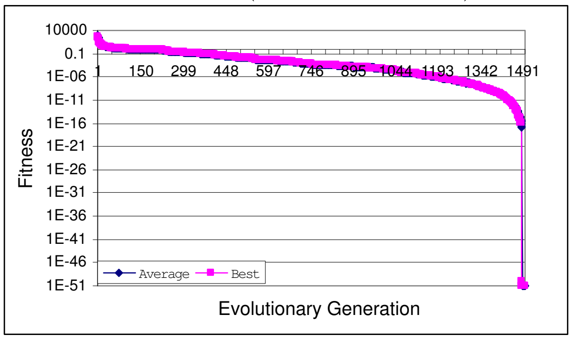
f_3 (Generalized Rosenbrock's Function)

Figure III: Best and average fitness of unimodal functions vs evolutionary generation

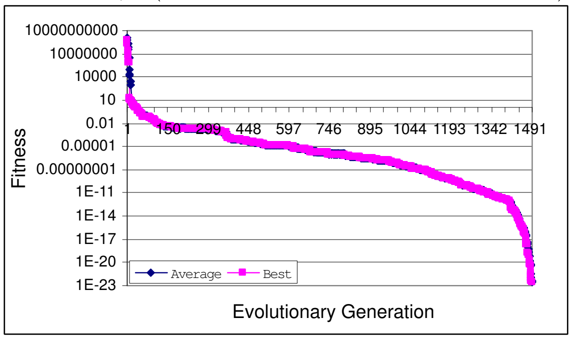
Figure IV showing strong exploring abilities for multimodal functions with high-dimensions and very large search spaces, NEP can find the potential area quickly in the initial stage of algorithm. The results approach to 0.1-0.001 after 200 generations. In the middle stage of the algorithm, a smooth behavior is indicated when it maybe patiently locate the position of the global optimum. A fast convergent speed once again appears in the later stage of the algorithm, which once again illuminate the *fine-tuning ability* of the non-uniform mutation operator. For function f_7, at about the last ten generations, the algorithm reached the minimal value. These results show that NEP has the merits of *long jumping (initial stage)* and *fine-tuning (later stage)*



f_6 (Ackley's Function)



f_7 (K.Generalized Griewank Function)



f_8 (L.Generalized Penalized Function)

Figure IV: Best and average fitness of multimodal functions vs evolutionary generation

abilities. For unimodal functions, NEP shows similar behaviors as the multimodal functions observed from Fig. III excluding function f_3. Although f_3 is the most difficultone, NEP still outperforms the adaptive LEP as Table II shows.

Table II shows that NEP is only outperformed by the adaptive LEP on the low dimensional multimodal functions with only a few local minima. For the high dimensional functions with the unimodal and multimodal, NEP is better in terms of the convergent ability and robustness. What is more, this encouraging result is achieved without introducing any extra parameters and no extra computation cost comparing with CEP.

Function f_3 is *nonseparable* [20] whose result is the worst one comparing with the results of other functions for NEP. Even though, it still outperforms the adaptive LEP in terms of the solution quality and the robustness. It is noteworthy that NEP nearly finds the global optima for all the

	Population Size		**Mean Best**		**Std Dev**	
f	NEP	Adaptive LEP	NEP	Adaptive LEP	NEP	Adaptive LEP
1	13	100	3.66e-19	6.32e-4	1.65e-18	7.6e-5
2	13	100	1.62e-5	4.18e-2	8.34e-6	5.97e-2
3	13	100	1.35e1	4.34e1	7.91	3.15e1
4	13	100	-12569.49	-11469.2	5.15e-12	5.82e1
5	13	100	5.68e-14	5.85e0	0	2.07
6	13	100	1.07e-11	1.9e-2	2.93e-11	1.0e-3
7	13	100	9.94e-3	2.4e-2	9.39e-3	2.8e-2
8	13	100	2.02e-21	6.0e-6	3.86e-21	1.0e-6
9	13	100	1.93e-20	9.8e-5	4.25e-19	1.2e-5
10	200	100	-1.03162845	-1.031	2.37e-11	0.00
11	200	100	3.008	3.000	2.22e-2	0.000
12	100	100	-6.71	-9.54	2.68	1.69
13	100	100	-7.82	-10.30	2.76	0.74
14	100	100	-7.53	-10.54	2.77	4.9e-5

Table II: Performance comparison between NEP and the adaptive LEP. "Mean Best" indicates the average of the minimum values obtained at every run and "Std Dev" stands for the standard deviation.

high-dimension multimodal functions except for function f_7 whose standard deviation just reaches 3 digits after the decimal point. The experimental results further confirm the difficulty of function f_7. The standard deviation of function f_5 is equal to zero, that is, all the 50 runs reach the optimum.

For functions with only a few local minima, the performance of NEP is a little worse than the adaptive LEP. As Schnier and Yao [27] analyzed the last benchmarks are rather deceptive. But we have no need to worry about it based on two reasons. Firstly and most importantly, all the realistic problems from engineering and society are generally very complicated. Secondly, there are many other methods to deal with such problems, such as steady-state GA [1] and multiple representations EA with a modified configuration [27] which perform very well for these functions.

B Comparisons with Parallel Genetic Algorithms

In this section, we will make further comparison between NEP and five parallel genetic algorithms which are *GD-BLX*^{r} [6], *ECO-GA model* [23], *CHC algorithm* [24], *deterministic crowding* (DC) [25] and *disruptive selection* (DS) [26]. All the results of the parallel genetic algorithms are obtained from [6] and detailed introductions about these algorithms can also be found in [6]. All these parallel genetic algorithms are based on the *BLX*-α crossover and non-uniform mutation.

1) Benchmark Functions

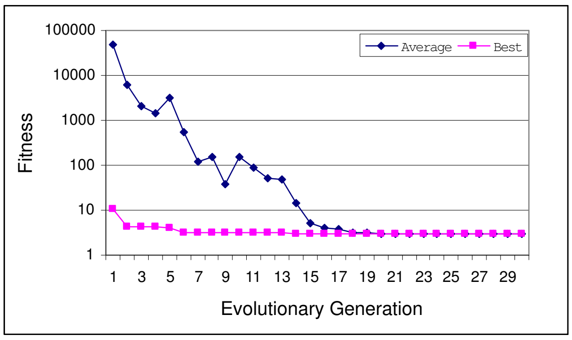
Herrera and Lozano [6] proposed a gradual distributed real-coded genetic algorithm (GD-RCGA), which is a "heterogeneous" distributed real-coded genetic algorithm that applies different configurations (control parameters, genetic operators) to each subpopulation. They proved that GD-RCGA consistently outperforms the sequential real-coded GAs and the homogeneous distributed GAs. In this paper, we will further compare the performance of the sequential NEP with the parallel genetic algorithms based on some typical test functions which can be found in Table III. Function ef_{10} is the expanded version of $f(x, y)$ which has nonlinear interaction between two variables. It is built in such a way that it induces nonlinear interaction across multiple variables.

2) Performance Comparison and Analysis

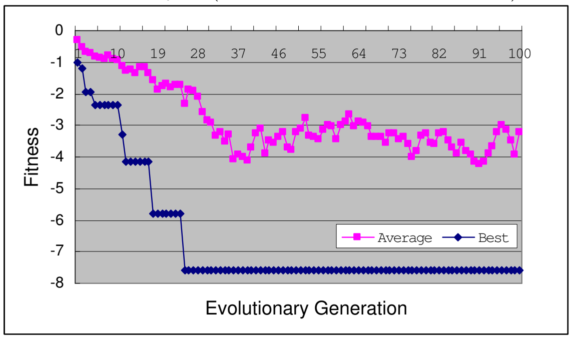
The algorithms were executed 30 times independently. There are 20 individuals in each subpopulation, 8 subpopulations in total (equivalent to 160 in sequential) and the number of evolutional generations is 5000. So we set the population size of NEP to be 7 for the first 5 functions and 8 for ef_{10}. The tournament size is 3 for benchmarks. A computational precision of **60** digits after point is used in NEP. So a result **being 0** means that it is **less that** 10^{-60} in NEP and vice versa in this subsection. But we do not know the computational precision of [6]. There are two groups of experiments for the parallel genetic algorithms. One group of experiment is based on the BLX-α crossover, the other is based on the extended FCB-α crossover [6]. The results of two groups of experiments are somewhat equal, so we just compare the performance of NEP with

Benchmark Functions	n	**D**	f_{min}
$f_{Sph} = \sum_{i=1}^{n} x_i^2$	25	$[-5.12, 5.12]^n$	0
$f_{Ros} = \sum_{i=1}^{n-1} [100(x^{i+1} - x_i^2)^2 + (x_i - 1)^2]$	25	$[-5.12, 5.12]^n$	0
$f_{Sch} = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	25	$[-65.536, 65.536]^n$	0
$f_{Ras} = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	25	$[-5.12, 5.12]^n$	0
$f_{Gri} = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	25	$[-600, 600]^n$	0
$ef_{10}(X) = f(x_1, x_2) + \ldots + f(x_n, x_1)$ $f_{10} = (x^2 + y^2)^{0.25}[\sin^2(50(x^2 + y^2)^{0.1}) + 1]$	10	$[-100, 100]^n$	0

Table III: Benchmark functions used by parallel genetic algorithms & NEP



f_{11} (Goldstein-Price Function)



f_{14} (S.Shekel's Family 10)

Figure V: Best and average fitness of low dimensional functions vs evolutionary generation

the experiments based on the *BLX*-α crossover in Table X in [6]. The experimental results are given in Table IV.

From Table IV, a very encouraging conclusion is obtained that NEP is in the same breath with the parallel genetic algorithms in terms of the solution quality and algorithmic robustness. First, NEP performs best for functions f_{Ros}, f_{Sch} among all the parallel genetic algorithms. Secondly, NEP is consistently and greatly outperforms the parallel genetic algorithms *R-DC-BLX* and *R-DS-BLX* based on *deterministic crowding* (DC) [25] and *disruptive selection* (DS) [26] for all the test functions. Thirdly, NEP significantly outperforms *ECO-BLX* for functions $f_{Ros}, f_{Sch}, f_{Ras}, f_{Gri}$ which can be found from **A** and **B** columns in Table IV, and is worse than it for functions f_{Sph}, ef_{10}. Lastly, NEP is outperformed by *CHC-BLX* and *GD-RCGA* for functions $f_{Sph}, f_{Ras}, f_{Gri}, ef_{10}$, but performs better than them for functions f_{Ros}, f_{Sch}. The result does not surprise us as *CHC-BLX* is a usually used benchmark parallel GA [6] and *GD-RCGA* performs even better than *CHC-BLX*.

Generally speaking, the *sequential* evolutionary algorithm NEP is comparable to the commonly used benchmark *parallel* genetic algorithm *CHC-BLX*.

IV Theoretical Analysis on the Executing Process of NEP

In Section III, experiments are used to show that NEP is fast and robust. In this section, we try to give the underling reasons by conducting theoretical analysis.

Suppose $X = \{x_1, \ldots, x_k, \ldots, x_m\}$ is a real-coded chromosome in the population and the component x_k (whose lower and upper bounds are L_k, U_k) is selected for variation through some mutation operation and x'_k is obtained. A decimal fraction r is uniformly and randomly generated in $[0, 1]$.

A Preliminary Analysis on NEP

It is well known that the normal [28] *Gaussian* random variables (mean 0 and variance 1) has the density function

$$f_{Gaussian}(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{10}$$

The *Cauchy* distribution has the density function

$$f_{Cauchy}(x) = \frac{1}{\pi(1 + x^2)} \tag{11}$$

	f_{Sph}				f_{Ros}				f_{Sch}			
Algorithms	**A**	**SD**	**B**	**O**	**A**	**SD**	**B**	**O**	**A**	**SD**	**B**	**O**
ECO-BLX	1e-31	2e-31	8e-33	2e0	2e1	4e-1	2e1	2e3	6e0	5e0	1e0	6e3
CHC-BLX	9e-32	2e-32	5e-32	4e0	2e1	7e-1	2e1	2e3	1e-1	3e-1	7e-12	1e3
R-DC-BLX	3e-6	2e-6	2e-7	9e0	1e1	5e0	4e0	8e3	3e3	7e2	2e3	1e4
R-DS-BLX	2e0	1e0	4e-1	8e1	3e2	1e2	1e2	9e4	2e3	4e2	1e3	4e5
GD-BLX^r	8e-53	4e-52	8e-58	9e-1	2e1	9e0	1e1	1e3	2e-6	3e-6	7e-8	2e3
NEP	**3e-26**	**6e-26**	**4e-29**	**1e-1**	**5e-1**	**6e-1**	**3e-3**	**2e2**	**3e-9**	**3e-9**	**3e-10**	**1e3**
	f_{Ras}				f_{Gri}				ef_{10}			
ECO-BLX	1e-2	1e1	8e1	3e2	1e0	3e-2	9e-1	9e0	5e-9	2e-8	4e-10	8e0
CHC-BLX	0e0	0e0	100%	7e1	0e0	0e0	100%	2e0	1e-7	2e-8	9e-8	8e0
R-DC-BLX	9e0	1e0	5e0	5e1	1e-2	1e-2	1e-4	3e1	2e-1	1e-1	8e-2	2e1
R-DS-BLX	5e1	1e1	3e1	3e2	6e0	3e0	2e0	3e2	1e1	3e0	7e0	9e1
GD-BLX^r	0e0	0e0	100%	2e1	4e-4	2e-3	96.7%	5e0	9e-39	5e-38	3e-47	6e0
NEP	**9e-10**	**1e-9**	**66.7%**	**7e-1**	**1e-2**	**2e-2**	**46.7%**	**5e-1**	**2e-2**	**5e-2**	**1e-6**	**1e0**

Table IV: Performance comparison between NEP and parallel genetic algorithms *ECO-BLX, CHC-BLX, R-DC-BLX, R-DS-BLX* and *GD-BLX*. All results are averaged over 30 trials. **A**: average of the best fitness of all runs. **SD**: standard deviation. **B**: best fitness of all runs. If the global optimum has be reached sometimes, this performance will represent the percentage of runs in which this happens. **O**: average of the fitness of all the elements appearing in all 30 runs.

Yao et al. [14] indicated that the expected length of *Gaussian* and *Cauchy* jumps are 0.8 and ∞.

$$E_{Gau}(x) = E|x'_k - x_k| = 2\int_0^{+\infty} x\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}dx = 0.80 \quad (12)$$

$$E_{Cau}(x) = E|x'_k - x_k| = 2\int_0^{+\infty} x\frac{1}{\pi(1+x^2)}dx = +\infty \quad (13)$$

Lee and Yao [15] also proved that the mean-square displacement is infinite.

$$E_{Levy}(x) = E|x'_k - x_k| = 2\int_0^{+\infty} L_\alpha(y)dy = +\infty \quad (14)$$

where $L_\alpha(y)$ stands for the probability function of *Levy* distribution and it has the following property

$$L_\alpha(y) \sim \frac{1}{y^{\alpha+1}}, \text{when } |y| \gg 1.$$

From Eq.(12, 13, 14), it is obvious that *Gaussian* mutation is much localized than *Cauchy* or *Levy* mutation. From this, FEP [14] and LEP [15] have higher probabilities of making longer jumps than CEP. However, as they analyzed, "longer jumps" are not always beneficial. If the current search point is near the small neighborhood of the global optimum, the "longer jumps" are detrimental. For NEP, comprehensive experiments show that it can find the promising areas quickly and locate the global optimum in a very high probability.

Let ξ be a *Bernoulli* random variable in Eq.(2) with

$$P(\xi = 0) = p(\xi = 1) = \frac{1}{2}$$

Let $\eta = -2\xi + 1$. Then η is a random variable with only two values

$$P(\eta = 1) = p(\eta = -1) = \frac{1}{2}$$

Thus by Eq.(1,2), we have

$$\begin{aligned} x'_k &= x_k + \frac{1+\eta}{2}\Delta(t, U_k - x_k) - \frac{1-\eta}{2}\Delta(t, x_k - L_k) \\ &= x_k + \tfrac{1+\eta}{2}(U_k - x_k)(1 - r^{(1-\frac{t}{T})^b}) + \\ &\quad \frac{1-\eta}{2}(x_k - L_k)(1 - r^{(1-\frac{t}{T})^b}) \qquad (15) \end{aligned}$$

Then the expected step size of the non-uniform mutation is

$$\begin{aligned} E|x'_k - x_k| &= P(\eta = 1)\cdot(U_k - x_k)\cdot E(1 - r^{(1-\frac{t}{T})^b}) + \\ &\quad P(\eta = -1)\cdot(x_k - L_k)\cdot E(1 - r^{(1-\frac{t}{T})^b}) \\ &= \tfrac{U_k - L_k}{2}E(1 - r^{(1-\frac{t}{T})^b}) \\ &= \tfrac{U_k - L_k}{2}\int_0^1 (1 - r^{(1-\frac{t}{T})^b})\mathrm{dr} \\ &= \frac{U_k - L_k}{2}\left(1 - \frac{1}{1 + (1-\frac{t}{T})^b}\right) \qquad (16) \end{aligned}$$

It is not like the *Gaussian* mutation which locally searches and the *Cauchy* or *Levy* mutation which makes long jumps from begin to end of the algorithms. It is these reasons that attract so many researchers to focus on the mutation operation on evolutionary programming [5, 11, 12, 14, 15, 21] or evolution strategy [13, 29, 30]. The expected step size of the non-uniform mutation in an interval depends on the generation t. Let $f(t) = E|x'_k - x_k|$. It is obvious that $f(t)$ is a decreasing function of t, because

$$\frac{\partial f(t)}{\partial t} = \frac{U_k - L_k}{2} \cdot \frac{\partial(1 - \frac{1}{1+(1-\frac{t}{T})^b})}{\partial t}$$
$$= -\frac{U_k - L_k}{2} \times \frac{\frac{b}{T} \cdot (1-\frac{t}{T})^{b-1}}{(1+(1-\frac{t}{T})^b)^2} < 0 \quad (17)$$

Hence the jumping size of the non-uniform mutation monotonously decreases with the progress of the algorithm. This roughly shows that the exploration region of NEP is smaller and smaller when t increases. However, we are still not able to say that the non-uniform mutation has the feature of searching the space uniformly initially and very locally at later stages of algorithms.

B Detailed Analysis on NEP

First, we will give an approximate analysis. From Eq.(15), we have the following *jumping equation*

$$x'_k = \begin{cases} x_k + (U_k - x_k) \cdot (1 - r^{(1-\frac{t}{T})^b}), & \text{if } \eta = 1 \\ x_k - (x_k - L_k) \cdot (1 - r^{(1-\frac{t}{T})^b}), & \text{if } \eta = \text{-1} \end{cases} \quad (18)$$

Without loss of generality, we assume x_k to jump to its right side and obtain x'_k. Then we have

$$x'_k = x_k + (U_k - x_k) \cdot (1 - r^{(1-\frac{t}{T})^b}) \quad (19)$$

From Eq.(19), it can be seen obviously that at the initial stage of the algorithm, i.e., $t \ll T$, $\frac{t}{T} \approx 0$. Since b is a constant,

$$x'_k \approx x_k + (U_k - x_k) \cdot (1 - r) \quad (20)$$

Similarly, to jump to left side, we have

$$x'_k \approx x_k - (x_k - L_k) \cdot (1 - r) \quad (21)$$

From Eq.(20, 21) and r is a uniform random fraction decimal in [0, 1], we can clearly say that at the initial stage of the algorithm, the search engine of NEP nearly explores the whole space uniformly.

Similar to the above analysis and from Eq.(19), if t is close to T (later stage of the algorithm), i.e., $\frac{t}{T} \approx 1$, then $\frac{t}{T} \approx 1 \Longrightarrow (1 - \frac{t}{T})^b \approx 0 \Longrightarrow r^{(1-\frac{t}{T})^b} \approx 1$ regardless of r. Therefore, $1 - r^{(1-\frac{t}{T})^b} \approx 0$. That is to say, the term of $1 - r^{(1-\frac{t}{T})^b}$ in Eq.(18) is "infinitely" approaching to 0. Then

$$x'_k = \begin{cases} x_k + \epsilon \cdot (U_k - x_k), & \text{if } \eta = 1 \\ x_k - \epsilon \cdot (x_k - L_k), & \text{if } \eta = \text{-1} \end{cases} \quad (22)$$

From Eq.(22), we can say that at later stage (t is close to T) of the algorithm, the search engine of NEP very locally exploits the neighborhood of the current solution.

Now, we are able to say that the non-uniform mutation has the feature of searching the space uniformly initially and very locally at later stage of algorithms. The theoretical analysis on the non-uniform mutation consummates the features that it "monotonously reduces" its search region with the run of the algorithm(Eq.(17)), however, the search step size has random factor(Eq.(1, 2)) besides its feature of searching the space uniformly initially and very locally at later stage of algorithms.

C Analysis from Another Perspective

From Eq.(19), mutation explores the interval $[x_k,\ U_k]$. Then we have

$$x'_k - x_k = (U_k - x_k) \cdot (1 - r^{(1-\frac{t}{T})^b}) \quad (23)$$

For any random uniform decimal fraction q in [0, 1], we have

$$P_q := P\{|x'_k - x_k| > (U_k - x_k) \times q\}$$
$$= P\{1 - r^{(1-\frac{t}{T})^b} > q\}$$
$$= P\{r < (1-q)^{(1-\frac{t}{T})^{-b}}\}$$
$$= q^{\frac{1}{(1-\frac{t}{T})^b}} \quad (24)$$

It follows from Eq.(24) that when $t \ll T$, $\frac{t}{T} \approx 0 \Longrightarrow (1 - \frac{t}{T})^b \approx 1$. Hence $P_q \approx q$. In such a case, x'_k approximately searches the whole right interval $[x_k,\ U_k]$ uniformly. It is the same when x_k jumps to the left side.

On the other hand, when $t \to T$, $\frac{t}{T} \approx 1 \Longrightarrow (1 - \frac{t}{T})^b \approx 0$ (but > 0). Then $q^{\frac{1}{(1-\frac{t}{T})^b}} \approx 0$ (but slightly > 0). So P_q consistently approaches to 0. In this case, x'_k mainly exploits the local domain of the current solution (x_k). It is the same when x_k jumps to the left side.

For the total varying principle of P_q, especially for the middle process, we let $g(t) = P_q$. Then

$$g'(t) = \frac{\partial g(t)}{\partial t} = \ln q \cdot \frac{b \cdot q^{\frac{1}{1-\frac{t}{T}}}}{T \cdot (1 - \frac{t}{T})^{b+1}} < 0 \quad (25)$$

which means that P_q is decreasing with the run of the algorithm. We can once again make the conclusion from the total perspective that the exploring region is smaller and smaller in the algorithm.

D Experimental Results

In order to illustrate our analysis about the feature of the non-uniform mutation, we consider two functions f_1, f_7 (just investigate their first component) chosen from Table I to see how they are varying during the algorithm. Function

f_1 is a typical unimodal sphere and f_7 is a typical multimodal benchmark. Figure VI clearly shows the feature of the non-uniform mutation. At the initial stage, variables "fly" to scan the whole search space. In the middle stage of algorithm, the exploring space has a decreasing trend. However, it is *not* absolutely monotonous as Fig. VI shows. Sometimes, NEP has another long jump leaving the current solution to find new promising area. At the later stage, they just exploit the neighborhood of the global optima ($x_i^* = 0$). This experiment strongly supports the theoretical analysis above about the feature of the non-uniform mutation.

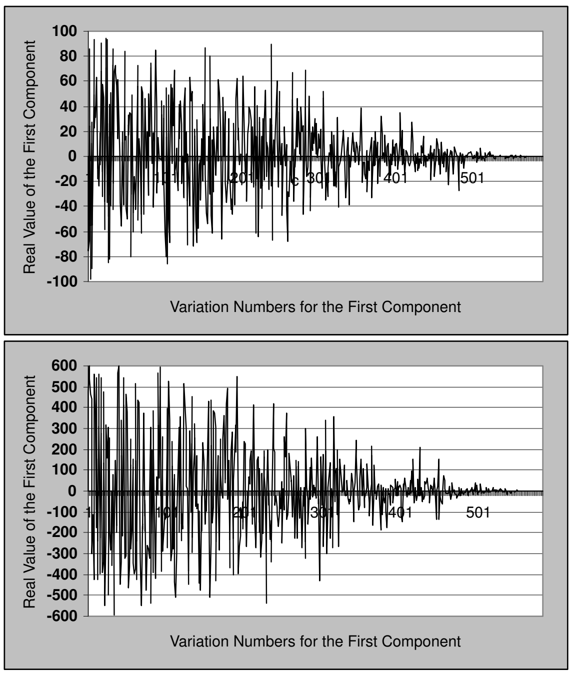


Figure VI: Mutation numbers vs the real value for the first component of functions f_1 and f_7. X coordinate stands for the mutation numbers and Y coordinate is the value of the first component coordinate.

We will make *a simple analogy* with persons to understand the behaviors of NEP. It likes an "intelligent pilot" who quickly scans the total search space initially, then switches to automobile, bike subsequently with the process of algorithm. In the final stage, he throws off all the vehicles decidedly and walks to his destination by small step. So it can exactly locate the global optimum of the problem. There is another important point in the execution of non-uniform mutation. This "intelligent pilot" has keen **greedy** idea who goes to the new region only if it is better than the current place. Otherwise, he will stay there until an even better area is found.

V An Adaptive Non-uniform Evolutionary Programming

A The Influence of the parameter b

Besides t in Eq.(16), there is another parameter b that we have always treat it as a constant. However, as Michalewicz [1] indicated that b is a parameter determining the degree of non-uniformity. Now we will analyze this parameter to show how it affects the search step of the non-uniformity. For a given t in Eq.(16), let

$$\begin{aligned} g(b) &= \frac{U_k - L_k}{2} \times \left(1 - \frac{1}{1 + (1 - \frac{t}{T})^b}\right) \\ &= \frac{U_k - L_k}{2} \times \frac{1}{1 + (1 - \frac{t}{T})^{-b}} \end{aligned}$$

We have

$$\begin{aligned} \frac{\partial g(b)}{\partial b} &= \frac{U_k - L_k}{2} \times \frac{-1}{(1 + (1 - \frac{t}{T})^{-b})^2} \times (1 - \frac{t}{T})^{-b} \times \ln(1 - \frac{t}{T}) \\ &= \frac{U_k - L_k}{2} \times \frac{\ln(\frac{T}{T-t}) \times (1 - \frac{t}{T})^{-b}}{(1 + (1 - \frac{t}{T})^{-b})^2} > 0 \qquad (26) \end{aligned}$$

From the above equation, when b becomes larger, the decreasing speed of the step size of the mutation becomes faster. That is, different b means different non-uniformity in the algorithms. Furthermore, as the experimental results of Table II in Section III-A show that NEP performs rather poor for the low-dimension benchmarks with only a few local optima. Consequently, an adaptive non-uniform evolutionary programming (ANEP) is proposed in this Section based on Eq.(26). It differs from NEP only in step 4 of the algorithm described in Section III-D. In ANEP, we generate three candidate offsprings with $b = 2, 5, 30$ from the same parent and select the best one as the surviving offspring. This scheme is adaptive because the value of b to be used is not predefined and is determined by the evolution.

B Experiments and Discussions

Only three benchmarks f_{12}, f_{13}, f_{14} in Table I are used to investigate the performance of ANEP because the performance of NEP is poor for these three benchmarks and excellent for others. Furthermore, there is no statistical difference between the performance of NEP and the adaptive NEP for other benchmarks (so the results are omitted here). Since ANEP uses three different values for b, we let the population size of ANEP (67) be one third of NEP (200) and all the other parameters remain the same to Section III-A. The computing results are averaged over 50 trials and are listed in Table V.

Although the performance of ANEP is improved based on the experimental results and the theoretical analysis on the

	Mean Best			Std Dev		
Func	Adaptive LEP	NEP	ANEP	Adaptive LEP	NEP	ANEP
f_{12}	-9.54	-6.71	-7.29	1.69	2.68	2.86
f_{13}	-10.30	-7.82	-8.59	0.74	2.76	2.49
f_{14}	-10.54	-7.53	-8.76	4.9e-5	2.77	2.40

Table V: Comparison between ANEP, NEP and the adaptive LEP for the low-dimension benchmarks with only a few local optima. "Mean Best" indicates the average of the minimum values obtained at every run and "Std Dev" stands for the standard deviation.

expected search step, it is still poorer than the adaptive LEP [15] for these benchmarks. As the "No Free Lunch Theorem" [19] states, there is no one optimization algorithm performs best for all problems. By the way, ANEP finds the optimal minima about 25 times in 50 trials for these three benchmarks.

VI Conclusion

In this paper, a new evolutionary programming algorithm NEP based on the non-uniform mutation is proposed. Comparisons with the newly proposed sequential and parallel evolutionary algorithms, NEP is generally faster and more robust.

Detailed theoretical analysis on NEP is presented. The probability convergence of NEP is first proved. Then its working schemes are also analyzed. NEP searches the space uniformly at the early stage of the algorithm and very locally at the later stage. The greedy idea is incorporated into the non-uniform search in order to avoid the *random blind jumping* and to "stay" at the promising solution areas. The probabilistic gradual decreasing jump length makes the algorithm exploring smaller and smaller regions with the progress of algorithm. At the later stage, the algorithm just exploits the neighborhood of the current solution so as to exactly locate the global optimum.

ACKNOWLEDGEMENT: The first author wants to thank Prof. Xin Yao, Dr. Jun He, Dr. Jin Li, Dr. Xiaoli Li and Dr. Yong Xu in the School of Computer Science of the University of Birmingham for their helps and suggestions during his visit.

References

[1] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, (3rd Edition), Springer, 1996.

[2] J.H. Holland, *Adaptation in Nature and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[3] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[4] H.P. Schwefel, *Numerical Optimization of Computer Models*, Wiley, New York, 1977.

[5] L.J. Fogel, *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.

[6] F. Herrera and M. Lozano, "Gradual Distributed Real-Coded Genetic Algorithms", *IEEE Trans. Evol. Comput*, vol4, no.1, pp43-63, 2000.

[7] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms", *IEEE Trans. Neural Networks* 5: 96-101, 1994.

[8] J. Suzuki, "A Markov Chain Analysis on Simple Genetic Algorithms", *IEEE Trans. Syst. Man. Cybern.* 25:655-659, 1995

[9] A.E. Eiben and G. Rudolph, "Theorey of Evolutionary Algorithms: a Bird view", *Theor. Comput. Sci*, 229(1/2): 3-9, 1999.

[10] J. He and X. H. Yu, "Conditions for the convergence of evolutionary algorithms", *Journal of Systems Architecture*, 47(7):601-612, 2001.

[11] D.B. Fogel, L.J. Fogel and E. Atmar, "Meta-evolutionary Programming", in *Proc. 25th Asilomer Conf. Sigals, Systems and Computers*, R. Chen, Ed., pp540-545, 1991.

[12] D.B. Fogel, *Evolving Artificial Intelligence*, Ph.D. Dissertation, Univ. California, 1992.

[13] N. Saravanan, D.B. Fogel, "Learning of Strategy Parameters in Evolutionary Programming: an Empirical Study", in *Proc. 3rd Annual Conf. Evolutionary Programming*, A. Sebald and L. Fogel, Eds., pp269-280, 1994.

[14] X. Yao, Y. Liu and G.M. Lin, "Evolutionary Programming Made Faster", *IEEE Trans. Evol. Comput.*, vol3, no.2, pp82-102, 1999.

[15] C.Y. Lee and X. Yao, "Evolutionary Programming Using Mutations Based on the Levy Probability Distribution", *IEEE Trans. Evol. Comput.*, vol8, no.1, pp1-13, 2004.

[16] B. Selman, H.J. Levesque and D.G. Mitchell, "A New Method for Solving Hard Satisfiability Problems", in *Proc. of the AAAI'92*, pp440-446, San Jose, CA, 1992.

[17] T. Jansen and I. Wegenr, "Evolutionary Algorithms-How to Cope with Plateaus of Constant Fitness and When to Reject Strings of Same Fitness", *IEEE Trans. Evol. Comput.*, vol5, no.6, pp589-599, 2001.

[18] A.B. Simoes and E. Costa, "Transposition versus Crossover: an Emirical Study", in *Proc. of the Genetic and Evol. Compu. Conf.*, vol1, pp612-619, 1999.

[19] D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization", *IEEE Trans. Evol. Comput.*, vol1, no.1, pp67-82, 1997.

[20] D. Whitley, S. Rana, J. Dzubera and E. Mathias, "Evaluateing Evolutionary Algorithms", *Artif. Intell.*, vol85, pp245-276, 1996.

[21] D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, New York: IEEE Press, 1995.

[22] I.De Falco, A. Della Cioppa and E. Tarantino, "Mutation-based Genetic Algorithm: Performance Evaluation", *Applied Soft Computing* 1: pp285-299, 2002.

[23] Y. Davidor, "A Naturally Occurring Niche Species Phenomenon: The Model and First Results", in *Proc. 4th Int. Conf. Genetic Algorithms*, R. Belew and L.B. Booker Ed., pp257-263, 1991.

[24] L.J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination", *Foundations of Genetic Algorithms* 1, G.J.E. Rawlin Ed., pp265-283, 1991.

[25] S.W. Mahfoud, "Crowding and Preselection Revised", *Parallel Problem Solving from Nature 2*, R. Manner and B. Manderich Ed., pp27-36, 1992.

[26] T.Kuo and S.Y. Huwang, "A Genetic Algorithm with Disruptive Selection", *IEEE Trans. Syst., Man and Cybern.*, vol26, no.2, pp299-307, 1996.

[27] T. Schnier and X. Yao, "Using Multiple Representations in Evolutionary Algorithms", in *Proc. 2000 Congress on Evolutionary Computation*, IEEE press, pp479-486, 2000.

[28] L. Devroye, "Non-uniform Random Variate Generation", New York: Springer-Verlag, 1996.

[29] X. Yao and Y. Liu, "Fast Evolution Strategies", *Contr. Cybern.*, vol26, no.3, pp467-496, 1997.

[30] Q. Zhou and Y.D. Li, "Directed Variation in Evolution Strategies ", *IEEE Trans. Evol. Comput.*, vol7, no.4, pp356- 366, 2003

[31] X.C. Zhao and X.S. Gao, "A Micro Evolutionary Programming for Optimization of Continuous Space", in A. Tiwari and R. Roy (Eds.), *PPSN VIII Workshop on Chanlledges in Real World Optimization Using Evolutionary Computing*, pp17-23, 2004.