Fast Evolution Strategies*

Xin Yao and Yong Liu
Computational Intelligence Group, School of Computer Science
University College, The University of New South Wales
Australian Defence Force Academy, Canberra, ACT, Australia 2600
Email: {xin,liuy}@csadfa.cs.adfa.oz.au, URL: http://www.cs.adfa.oz.au/˜xin

Abstract

Evolution strategies are a class of general optimisation algorithms which are applicable to functions that are multimodal, nondifferentiable, or even discontinuous. Although recombination operators have been introduced into evolution strategies, the primary search operator is still mutation. Classical evolution strategies rely on Gaussian mutations. A new mutation operator based on the Cauchy distribution is proposed in this paper. It is shown empirically that the new evolution strategy based on Cauchy mutation outperforms the classical evolution strategy on most of the 23 benchmark problems tested in this paper. The paper also shows empirically that changing the order of mutating the objective variables and mutating the strategy parameters does not alter the previous conclusion significantly, and that Cauchy mutations with different scaling parameters still outperform the Gaussian mutation with self-adaptation. However, the advantage of Cauchy mutations disappears when recombination is used in evolution strategies. It is argued that the search step size plays an important role in determining evolution strategies' performance. The large step size of recombination plays a similar role as Cauchy mutation.

Keyword — Evolution strategies, function optimisation, Cauchy mutation.

1 Introduction

Among three major branches of evolutionary computation, i.e., genetic algorithms (GAs), evolutionary programming (EP) and evolution strategies (ESs), ESs are the only one which was originally proposed for numerical optimisation and is still mainly used in optimisation [2, 3]. The primary search operator in ESs is mutation although recombinations have been used. The state-of-the-art of ESs is (μ, λ)-ES [4, 5], where $\lambda > \mu \geq 1$. (μ, λ) means that μ parents generate λ offspring through recombination and mutation in each generation. The best μ offspring are selected deterministically from the λ offspring and replace the parents. Elitism and probabilistic selection are not used. This paper first considers a simplified version of ESs, i.e., ESs without any recombination. Then ESs with recombination and a different order of mutating objective variables and strategy parameters are investigated.

ESs can be regarded as a population-based variant of generate-and-test algorithms [6]. They use search operators such as mutation to **generate** new solutions and use a selection scheme to **test** which of the newly generated solutions should survive to the next generation. The advantage of viewing ESs (and other evolutionary algorithms, EAs) as a variant of generate-and-test search algorithms is that the relationships between ESs and other search algorithms, such as simulated annealing (SA), tabu search (TS), hill-climbing, etc., can be made clearer and thus easier to explore. In addition, the generate-and-test view of ESs makes it obvious that "genetic" operators, such as crossover (recombination) and mutation, are really stochastic search operators which are used to generate new search points in a search space. The effectiveness of a search operator would be best described by its ability to produce promising new points which have higher probabilities of finding a global optimum, rather than by some biological analogy. The role of test in a generate-and-test algorithm or selection in ESs is to evaluate how "promising" a new point is. Such evaluation can be either deterministic or probabilistic.

The (μ, λ)-ESs use Gaussian mutation to *generate* new offspring and deterministic selection to *test* them. There has been a lot of work on different selection schemes for ESs [7]. However, work on mutations has

*An earlier short version of this paper [1] was presented at the Sixth Annual Conference on Evolutionary Programming, Indianapolis, USA, 13-16 April 1997. This work is partially supported by a University College Special Research Grant. The paper has been accepted by the journal *Control and Cybernetics*.

been concentrated on self-adaptation [2, 5] rather than on new mutations. Gaussian mutations seem to be the only choice [2, 5]. Recently, Cauchy mutation has been proposed as a very promising search operator due to its higher probability of making long jumps [8, 9, 10]. In [8, 9], a fast EP based on Cauchy mutation was proposed. It compares favourably to the classical EP on 23 benchmark functions (up to 30 dimensions). In [10], the idea of using Cauchy mutation in EAs was independently studied by Kappler. An $(1+1)$ EA without self-adaptation and recombination was investigated. Both analytical and numerical results on 3 one- or two- dimension functions were presented. It was pointed out that "in one dimension, an algorithm working with Cauchy distributed mutations is both more robust and faster. This result cannot easily be generalized to higher dimensions, ..." [10].

This paper continues the work of fast EP [8] and studies fast ESs which use Cauchy mutations. The idea of Cauchy mutation was originally inspired by fast simulated annealing [11, 12]. The relationship between the classical ESs (CES) using Gaussian mutation and the fast ESs (FES) using Cauchy mutation is analogous to that between classical simulated annealing and fast simulated annealing. This paper investigates multi-membered ESs, i.e., (μ, λ)-ESs with self-adaptation. Extensive experimental studies on 23 benchmark problems (up to 30 dimensions) have been carried out. The results have shown that FES outperforms CES on most of the 23 benchmark problems.

The rest of this paper is organised as follows. Section 2 formulates the global optimisation problem considered in this paper and describes the implementation of CES. Section 3 describes the implementation of FES. Section 4 presents and discusses the experimental results on CES and FES using 23 benchmark problems. Section 5 investiagtes different ES variants. Finally, Section 6 concludes with a few remarks.

2 Function Optimisation By Classical Evolution Strategies

A global minimisation problem can be formalised as a pair (S, f), where $S \subseteq R^n$ is a bounded set on R^n and $f : S \mapsto R$ is an n-dimensional real-valued function. The problem is to find a point $\mathbf{x}_{min} \in S$ such that $f(\mathbf{x}_{min})$ is a global minimum on S. More specially, it is required to find an $\mathbf{x}_{min} \in S$ such that

$$\forall \mathbf{x} \in S : f(\mathbf{x}_{min}) \leq f(\mathbf{x})$$

Here f does not need to be continuous, but it must be bounded. We only consider unconstrained function minimisation in this paper. Function maximisation can be converted to a minimisation problem easily by taking a negative sign.

According to the description by Bäck and Schwefel [3], the (μ, λ)-CES is implemented as follows in our studies:

1. Generate the initial population of μ individuals, and set $k = 1$. Each individual is taken as a pair of real-valued vectors, $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$.

2. Evaluate the fitness value for each individual $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, of the population based on the objective function, $f(\mathbf{x}_i)$.

3. Each parent $(\mathbf{x}_i, \eta_i)$, $i = 1, \cdots, \mu$, creates λ/μ offspring on average, so that a total of λ offspring are generated: for $i = 1, \cdots, \mu$, $j = 1, \cdots, n$, and $k = 1, \cdots, \lambda$,

$$\begin{aligned} \mathbf{x}_k{}'(j) &= \mathbf{x}_i(j) + \eta_i(j) N(0,1), & (1)\\ \eta_k{}'(j) &= \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) & (2) \end{aligned}$$

   where $\mathbf{x}_i(j)$, $\mathbf{x}_k{}'(j)$, $\eta_i(j)$ and $\eta_k{}'(j)$ denote the j-th component of the vectors $\mathbf{x}_i$, $\mathbf{x}_k{}'$, η_i and $\eta_k{}'$, respectively. $N(0,1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0,1)$ indicates that the random number is generated anew for each value of j. The factors τ and τ' are usually set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$ [3].

4. Evaluate the fitness of each offspring $(\mathbf{x}_i{}', \eta_i{}')$, $\forall i \in \{1, \cdots, \lambda\}$, according to $f(\mathbf{x}_i{}')$.

5. Sort offspring $(\mathbf{x}_i{}', \eta_i{}')$, $\forall i \in \{1, \cdots, \lambda\}$ in a non-descending order according to their fitness values, and select the μ best offspring out of λ to be parents of the next generation.

6. Stop if the stopping criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

It is worth mentioning that swapping the order of Eq.(1) and Eq.(2) and using $\eta_k{}'(j)$ to generate $\mathbf{x}_k{}'(j)$ may give better performance for some problems [13]. However, no definite conclusion can be drawn yet.

3 Fast Evolution Strategies

The one-dimensional Cauchy density function centred at the origin is defined by:

$$f_t(x) = \frac{1}{\pi} \frac{t}{t^2 + x^2}, \quad -\infty < x < \infty,$$

where $t > 0$ is a scale parameter [14](pp.51). The corresponding distribution function is

$$F_t(x) = \frac{1}{2} + \frac{1}{\pi} arctan \left(\frac{x}{t} \right).$$

The shape of $f_t(x)$ resembles that of the Gaussian density function but approaches the axis so slowly that an expectation does not exist. As a result, the variance of the Cauchy distribution is infinite. Figure 1 shows the difference between Cauchy and Gaussian functions by plotting them in the same diagram. It is obvious that the Cauchy function is more likely to generate a random number far away from the origin because of its long flat tails. This implies that Cauchy mutation in FES is more likely to escape from a local minimum or move away from a plateau.

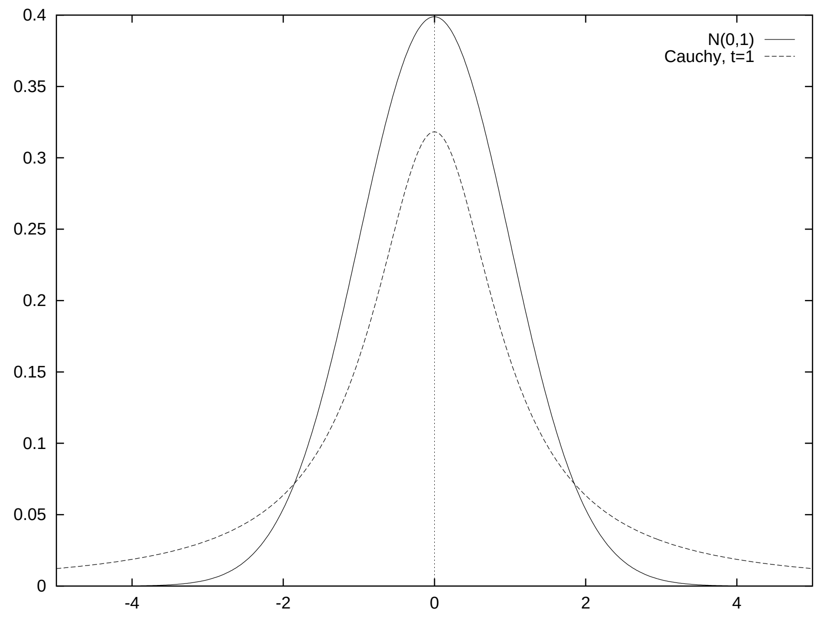


Figure 1: Comparison between Cauchy and Gaussian distributions.

In order to investigate the impact of Cauchy mutation on ESs, the minimal change has been made to the CES. The FES studied in this paper is kept exactly the same as the CES described in Section 2 except for Eq.(1) which is replaced by the following:

$$\mathbf{x}_k{}'(j) = \mathbf{x}_i(j) + \eta_i(j)\delta_j \tag{3}$$

where δ_j is an Cauchy random variable with the scale parameter $t = 1$ and is generated anew for each value of j. It is worth indicating that Eq.(2) is unchanged in FES in order to keep the modification of CES to a minimum. η in FES plays the role of the scale parameter t not the variance in the Cauchy distribution.

In our experiments, the Gaussian random number was generated according to the following function in FORTRAN [15] (pp.280).

```
      FUNCTION gasdev(idum)
      INTEGER idum
      REAL gasdev
C       Returns a normally distributed number with zero mean and unit var-
C       iance, using ran1(idum) as the source of uniform numbers.
      INTEGER iset
      REAL fac,gset,rsq,v1,v2,ran1
      SAVE iset,gset
      DATA iset/0/
      if(iset.eq.0)then
 1      v1=2.*ran1(idum)-1.
        v2=2.*ran1(idum)-1.
        rsq=v1**2+v2**2
        if(rsq.ge.1..or.rsq.eq.0.)goto 1
        fac=sqrt(-2.*log(rsq)/rsq)
        gset=v1*fac
        gasdev=v2*fac
        iset=1
      else
        gasdev=gset
        iset=0
      endif
      return
      END
```

The Cauchy random number was generated according to the following FORTRAN function [16] (pp.451).

```
      FUNCTION cauchy(idum)
      REAL cauchy
C      Returns a Cauchy random number with probability density function
C      f(x)=1/(pi*(1+x*x)).
      REAL v1,v2
      v1=gasdev(idum)
      v2=gasdev(idum)
      if(v2.ne.0.)then
        cauchy=v1/v2
      else
        cauchy=0.0
      endif
      return
      END
```

The uniform random generator was generated according to a FORTRAN function given by Press *et al.* [15] (pp.271).

4 Experimental Studies

4.1 Test Functions

A set of 23 well-known functions [17, 18, 4, 3, 19, 20] are used in our experimental studies. This relatively large set is necessary in order to reduce biases in evaluating algorithms. The 23 test functions are listed in Table 1. The detailed description of each function is given in the appendix. Functions f_1 to f_{13} are high dimensional problems. Functions f_1 to f_5 are unimodal functions. Function f_6 is the step function which has one minimum and is discontinuous. Function f_7 is a noisy quartic function, where $random[0, 1)$ is a uniformly distributed random variable in $[0, 1)$. Functions f_8 to f_{13} are multimodal functions where the number of local minima increases exponentially with the function dimension [18, 4]. Functions f_{14} to f_{23} are low-dimensional functions which have only a few local minima [18]. For unimodal functions, the convergence rate of FES and CES is more important than the final results of the optimisation in this paper, as there are other methods which are specifically designed to optimise unimodal functions. For multimodal functions, the important issue is whether an algorithm can find a better solution in a shorter time.

4.2 Experimental Setup

The experimental setup was based on Bäck and Schwefel's suggestion [3]. For all experiments, $(30, 200)$-ES with self-adaptive standard deviations, no correlated mutations, no recombination, the same initial standard

Table 1: The 23 test functions used in our experimental studies, where n is the dimension of the function, f_{min} is the minimum value of the function, and $S \subseteq R^n$. The detailed description of each function is given in the appendix.

Test function	n	S	f_{min}
$f_1(x) = \sum_{i=1}^n x_i^2$	30	$[-100, 100]^n$	0
$f_2(x) = \sum_{i=1}^n \lvert x_i \rvert + \prod_{i=1}^n \lvert x_i \rvert$	30	$[-10, 10]^n$	0
$f_3(x) = \sum_{i=1}^n (\sum_{j=1}^i x_j)^2$	30	$[-100, 100]^n$	0
$f_4(x) = \max_i \{\lvert x_i \rvert, 1 \leq i \leq n\}$	30	$[-100, 100]^n$	0
$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^n$	0
$f_6(x) = \sum_{i=1}^n (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100, 100]^n$	0
$f_7(x) = \sum_{i=1}^n i x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]^n$	0
$f_8(x) = \sum_{i=1}^n \left(-x_i \sin(\sqrt{\lvert x_i \rvert})\right)$	30	$[-500, 500]^n$	-12569.5
$f_9(x) = \sum_{i=1}^n [x_i^2 - 10\cos(2\pi x_i) + 10)]$	30	$[-5.12, 5.12]^n$	0
$f_{10}(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^n \cos 2\pi x_i\right)$	30	$[-32, 32]^n$	0
$+20 + e$			
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$	0
$f_{12}(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})]\right.$	30	$[-50, 50]^n$	0
$\left.+(y_n - 1)^2\right\} + \sum_{i=1}^n u(x_i, 10, 100, 4),$			
$y_i = 1 + \frac{1}{4}(x_i + 1)$			
$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \leq x_i \leq a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$			
$f_{13}(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})]\right.$	30	$[-50, 50]^n$	0
$\left.+(x_n - 1)[1 + \sin^2(2\pi x_n)]\right\} + \sum_{i=1}^n u(x_i, 5, 100, 4)$			
$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^2 (x_i - a_{ij})^6}\right]^{-1}$	2	$[-65.536, 65.536]^n$	1
$f_{15}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	$[-5, 5]^n$	0.0003075
$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$	-1.0316285
$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$	0.398
$f_{18}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2$	2	$[-2, 2]^n$	3
$+6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1$			
$+12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$			
$f_{19}(x) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^4 a_{ij}(x_j - p_{ij})^2\right]$	4	$[0, 1]^n$	-3.86
$f_{20}(x) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^6 a_{ij}(x_j - p_{ij})^2\right]$	6	$[0, 1]^n$	-3.32
$f_{21}(x) = -\sum_{i=1}^5 [(x - a_i)^T(x - a_i) + c_i]^{-1}$	4	$[0, 10]^n$	$-1/c_1$
$f_{22}(x) = -\sum_{i=1}^7 [(x - a_i)^T(x - a_i) + c_i]^{-1}$	4	$[0, 10]^n$	$-1/c_1$
$f_{23}(x) = -\sum_{i=1}^{10} [(x - a_i)^T(x - a_i) + c_i]^{-1}$	4	$[0, 10]^n$	$-1/c_1$
where $c_1 = 0.1$			

deviations 3.0, and the same initial population were used. All experiments were repeated for 50 runs. The initial population was generated uniformly at random in the ranges specified in Table 1. The number of generations for each function was determined after some limited preliminary runs which showed that an ES would have converged (either prematurely or globally) after certain number of generations. There is little point running the algorithm longer if it is unlikely to improve the performance further.

4.3 Experimental Results

4.3.1 Unimodal Functions (f_1–f_7)

Unimodal functions are not the most interesting and challenging test problems for global optimisation algorithms, such as ESs. There are more efficient algorithms than ESs, which are specifically designed to optimise them. The aim here is to use them to get a picture of the convergence rate of CES and FES. Figures 2 and 3 show the evolutionary process of CES and FES on unimodal functions f_1–f_7. The final results are summarised in Table 2.

Table 2: Comparison between CES and FES on f_1–f_7. The results were averaged over 50 runs. "Mean Best" indicates the mean best function values found in the last generation. "Std Dev" stands for the standard deviation.

Function	Number of	FES		CES		FEP−CEP
	Generations	Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_1	750	2.5×10^{-4}	6.8×10^{-5}	3.4×10^{-5}	8.6×10^{-6}	$22.07^\dagger$
f_2	1000	6.0×10^{-2}	9.6×10^{-3}	2.1×10^{-2}	2.2×10^{-3}	$27.96^\dagger$
f_3	2500	1.4×10^{-3}	5.3×10^{-4}	1.3×10^{-4}	8.5×10^{-5}	$16.53^\dagger$
f_4	2500	5.5×10^{-3}	6.5×10^{-4}	0.35	0.42	$-5.78^\dagger$
f_5	7500	33.28	43.13	6.69	14.45	$3.97^\dagger$
f_6	750	0	0	411.16	695.35	$-4.18^\dagger$
f_7	1500	1.2×10^{-2}	5.8×10^{-3}	3.0×10^{-2}	1.5×10^{-2}	$-7.93^\dagger$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

In terms of final results, FES performs better than CES on f_4, f_6 and f_7, but worse than CES on f_1–f_3 and f_5. No strong conclusion can be drawn here. However, a closer look at the evolutionary processes reveals some interesting facts. For example, FES performs far better than CES on f_6 (the step function). It has a very fast convergence rate and converges to the global minimum every time. This indicates that FES is very good at dealing with plateaus due to its long jumps. Such long jumps enable FES to move from one plateau to a lower one easily, while CES would have to wander about a plateau for a long time before it can reach a lower plateau.

FES's behaviour on f_1 is also very interesting. According to Figure 2, f_1's value decreases much faster for FES than for CES in the beginning. This is probably caused by FES's long jumps, which take it to the center of the sphere more rapidly. When FES approaches the center, i.e., the minimum, long jumps are less likely to generate better offspring and FES has to depend on small steps to move towards the minimum. The smaller central part of the Cauchy distribution, as shown by Figure 1, implies Cauchy mutation is weaker than Gaussian one at fine-grained neighbourhood (local) search. Hence the decrease of f_1's value for FES slows down considerably in the vicinity of the minimum, i.e., when f_1 is smaller than 10^{-3}. CES, on the other hand, improves f_1's value steadily throughout the evolution and eventually overtakes FES.

The behaviour of FES and CES on other functions can be explained in a similar way. The probability of making long jumps by a mutation plays an important role in determining the behaviour of ESs.

4.3.2 Multimodal Functions With Many Local Minima (f_8–f_{13})

Functions f_8–f_{13} are multimodal functions with many local minima. The number of local minima increases exponentially as the function dimension increases [18, 4]. These functions appear to be very "rugged" and difficult to optimise. Figure 4 shows the 2-dimensional version of f_8.

The evolutionary processes of FES and CES for f_8–f_{13} are shown by Figures 5 and 6. The final results are summarised in Table 3. Somewhat surprisingly, FES outperforms CES consistently on these apparently

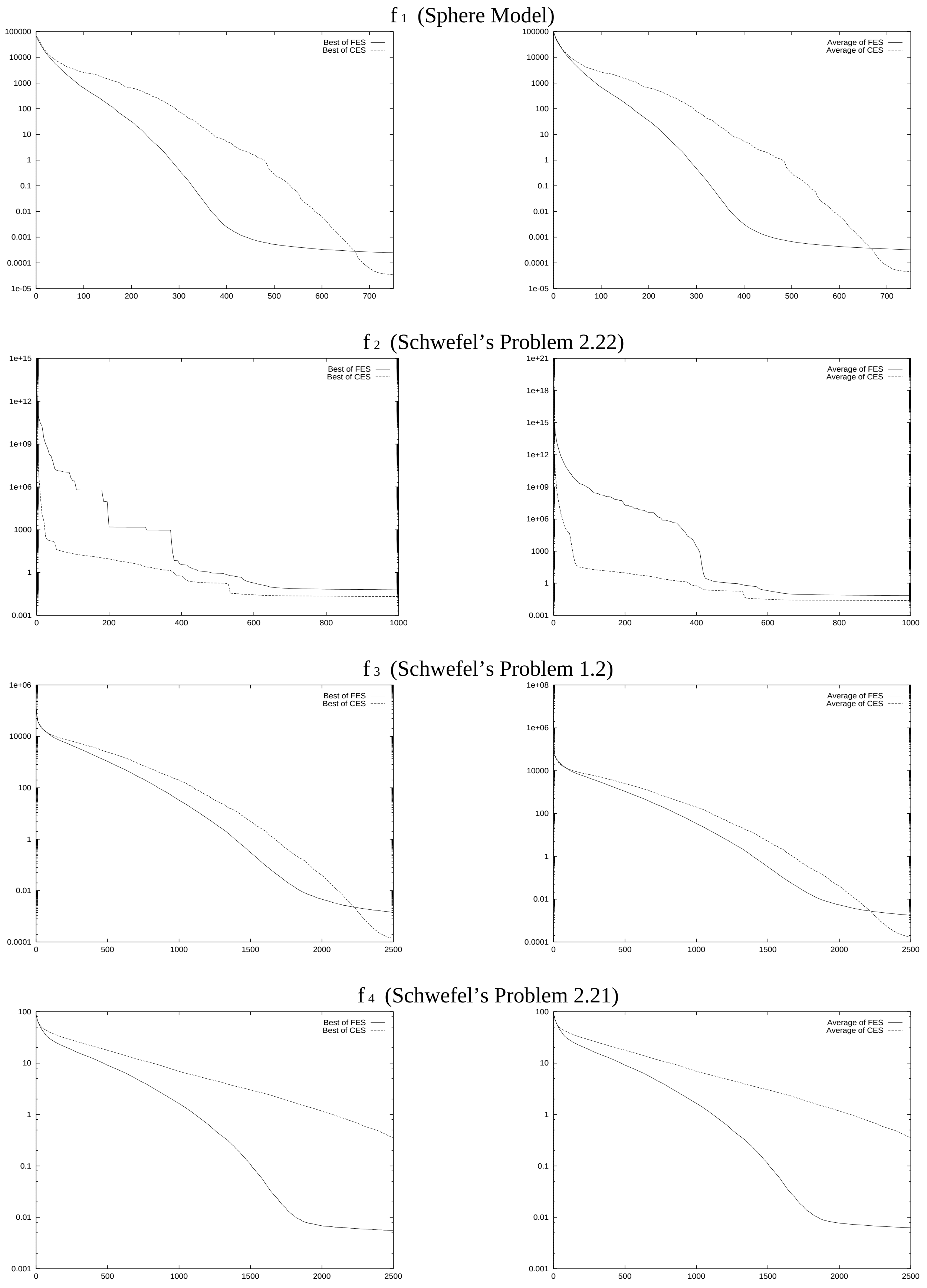
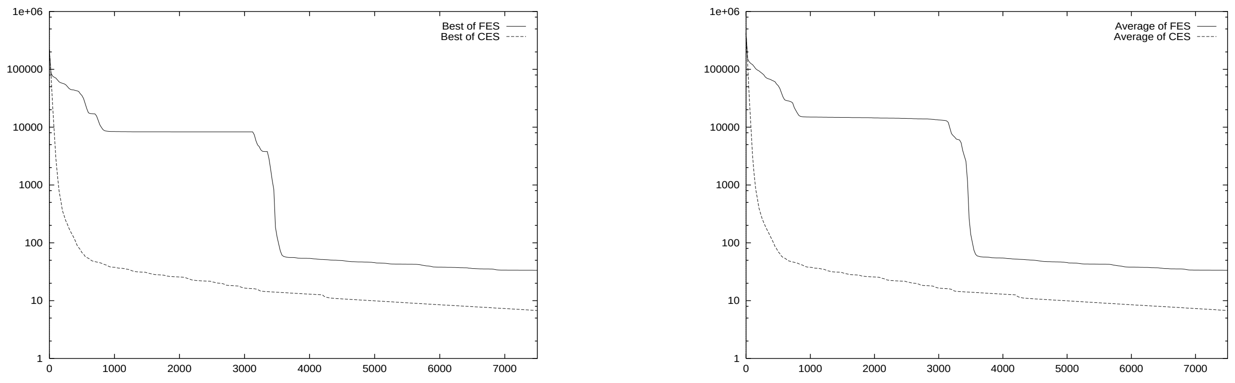
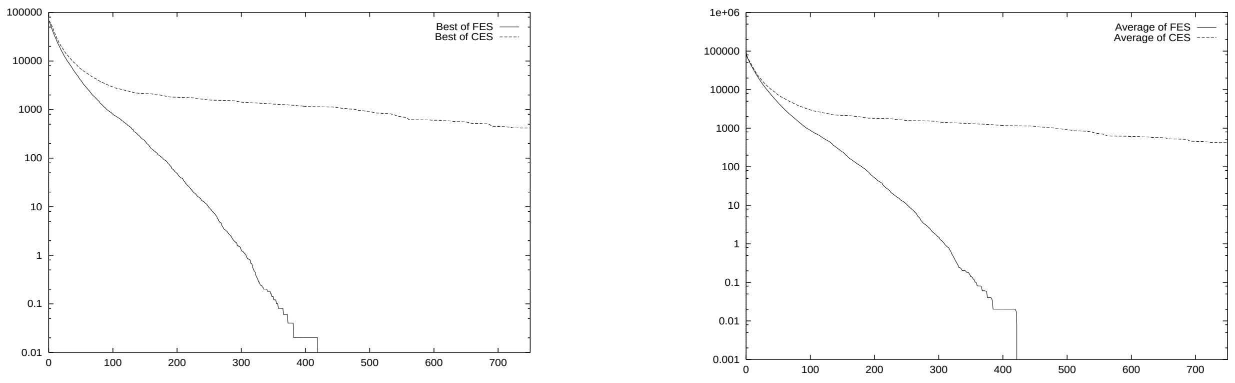


Figure 2: Comparison between CES and FES on f_1–f_4. The vertical axis is the function value and the horizontal axis is the number of generations.

f$_5$ (Generalized Rosenbrock's Function)



f$_6$ (Step Function)



f$_7$ (Quartic Function with Noise)

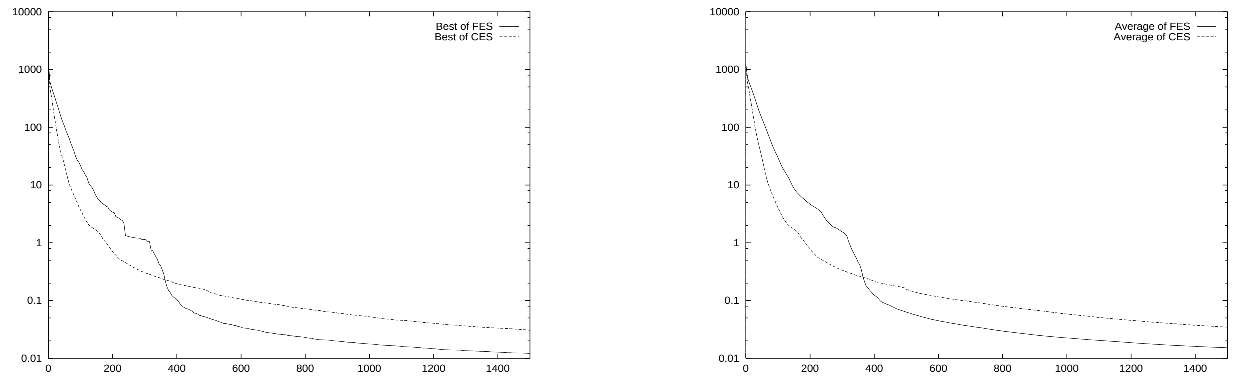


Figure 3: Comparison between CES and FES on f_5–f_7.The vertical axis is the function value and the horizontal axis is the number of generations.

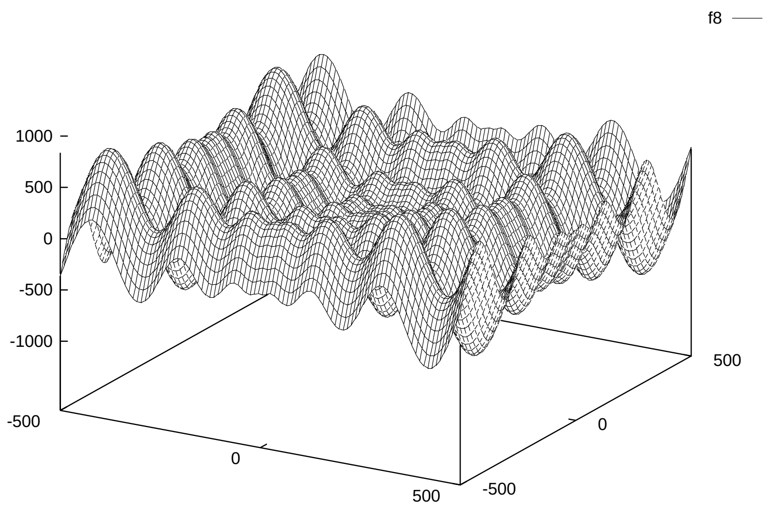


Figure 4: The 2-dimensional version of f_8.

difficult functions. Figures 5 and 6 show that CES stagnates rather early in search and makes little progress thereafter, while FES keeps finding better function values throughout the evolution. It appears that CES is trapped in one of the local minima and is unable to get out due to its more localised Gaussian mutation. FES, on the other hand, has a much higher probability of making long jumps and thus is easier to get out of a local minimum when trapped. A good near (global) minimum is more likely to be found by FES.

Table 3: Comparison between CES and FES on f_8–f_{13}. The results were averaged over 50 runs. "Mean Best" indicates the mean best function values found in the last generation. "Std Dev" stands for the standard deviation.

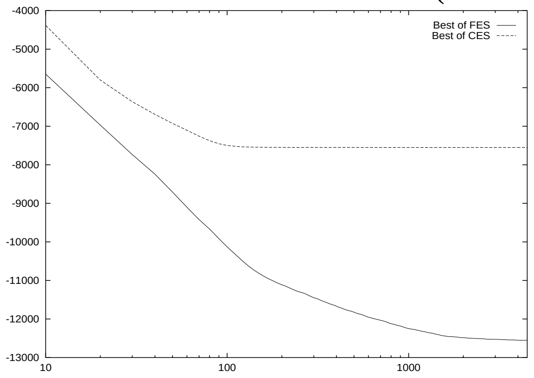
Function	Number of	FES		CES		FES−CES
	Generations	Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_8	4500	-12556.4	32.53	-7549.9	631.39	$-56.10^{\dagger}$
f_9	2500	0.16	0.33	70.82	21.49	$-23.19^{\dagger}$
f_{10}	750	1.2×10^{-2}	1.8×10^{-3}	9.07	2.84	$-22.51^{\dagger}$
f_{11}	1000	3.7×10^{-2}	5.0×10^{-2}	0.38	0.77	$-3.11^{\dagger}$
f_{12}	750	2.8×10^{-6}	8.1×10^{-7}	1.18	1.87	$-4.45^{\dagger}$
f_{13}	750	4.7×10^{-5}	1.5×10^{-5}	1.39	3.33	$-2.94^{\dagger}$

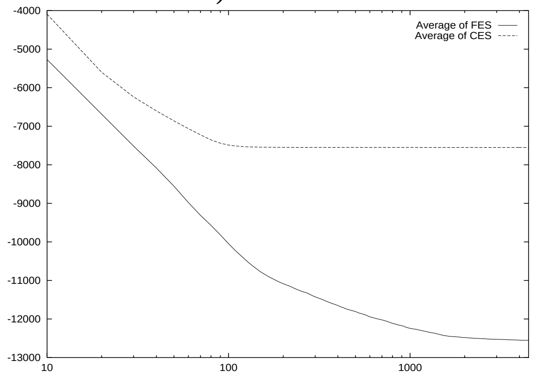
†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

4.3.3 Multimodal Functions With a Few Local Minima (f_{14}–f_{23})

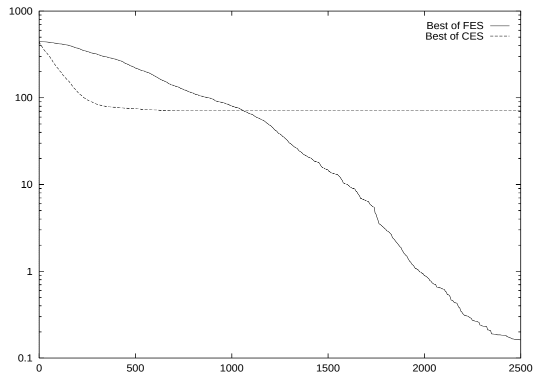
The evolutionary processes of FES and CES on functions f_{14}–f_{23} are shown by Figures 6, 7 and 8. The final results are summarised in Table 4. Although these functions are also multimodal functions, the behaviour of FES and CES on them are rather different from that on multimodal functions with many local minima. There is no consistent winner here. For functions f_{14} and f_{15}, FES outperforms CES. However, FES is outperformed by CES on functions f_{21} and f_{22}. No statistically significant difference has been detected between FES's and CES's performance on other functions. In fact, the final results of FES and CES were exactly the same for f_{16}, f_{17} and f_{18} although the initial behaviours were different.

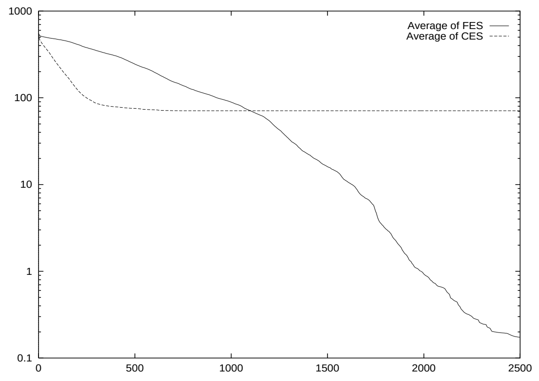
At the beginning, it was suspected that the low dimensionality of functions f_{14}–f_{23} might contribute to the similar performance of FES and CES. Hence another set of experiments were carried out using the f$_8$ (Generalized Schwefel's Problem 2.26)



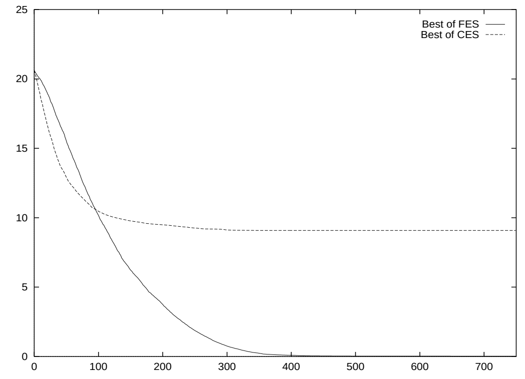


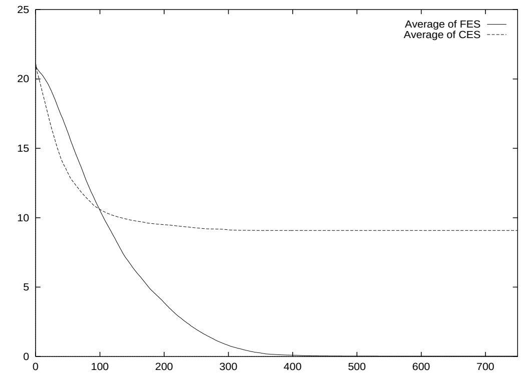
f$_9$ (Generalized Rastrigin's Function)



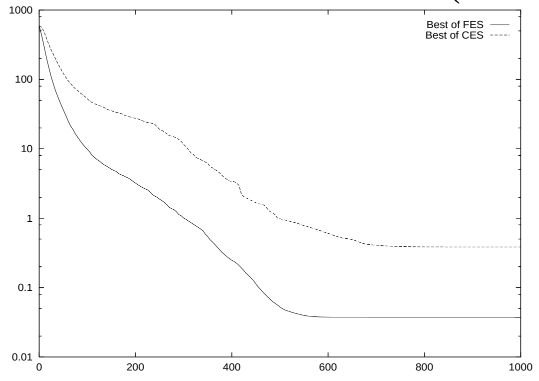


f$_{10}$ (Ackley's Function)





f$_{11}$ (Generalized Griewank Function)



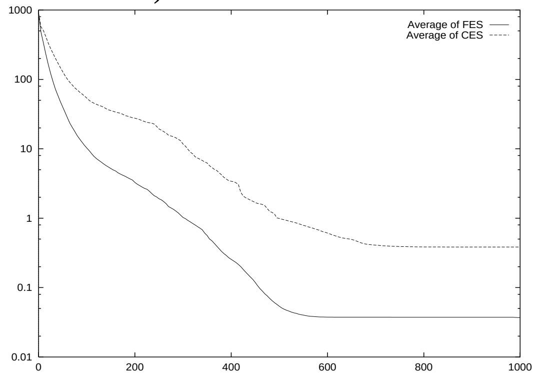
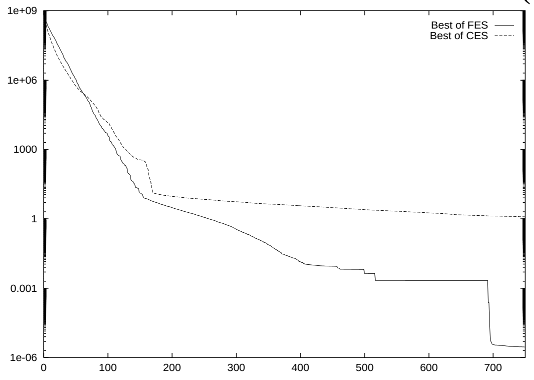
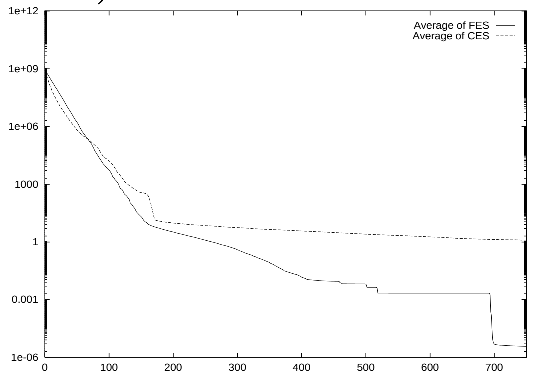


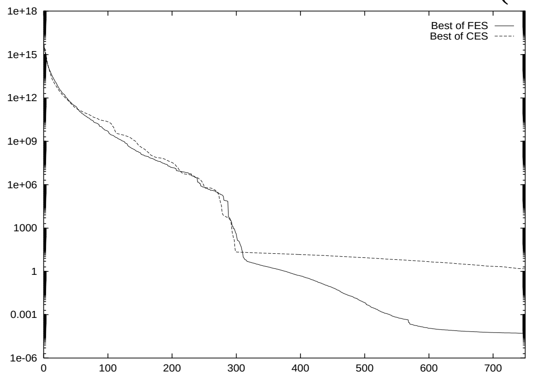
Figure 5: Comparison between CES and FES on f_8–f_{11}. The vertical axis is the function value and the horizontal axis is the number of generations.

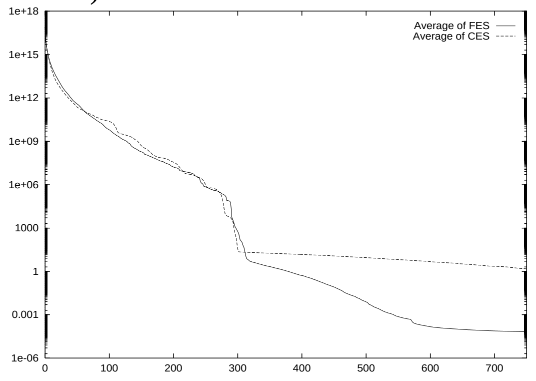
f_{12} (Penalised Function P8)



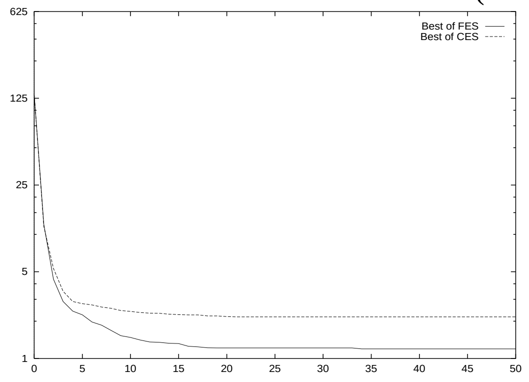


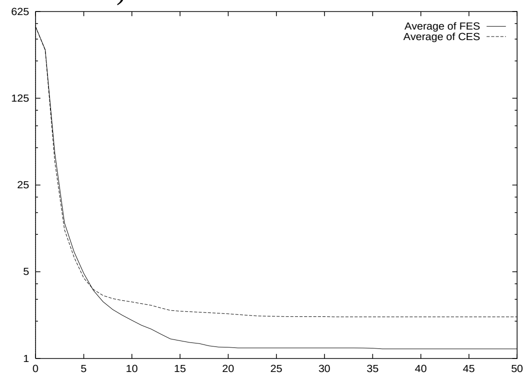
f_{13} (Penalised Function P16)



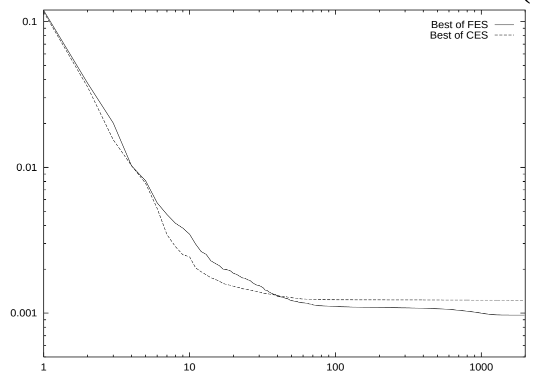


f_{14} (Shekel's Foxholes Function)





f_{15} (Kowalik's Function)



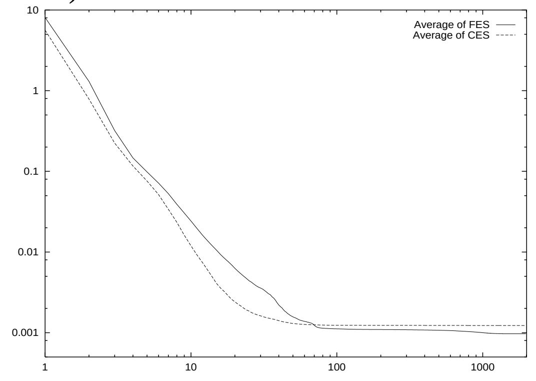


Figure 6: Comparison between CES and FES on f_{12}–f_{15}. The vertical axis is the function value and the horizontal axis is the number of generations.

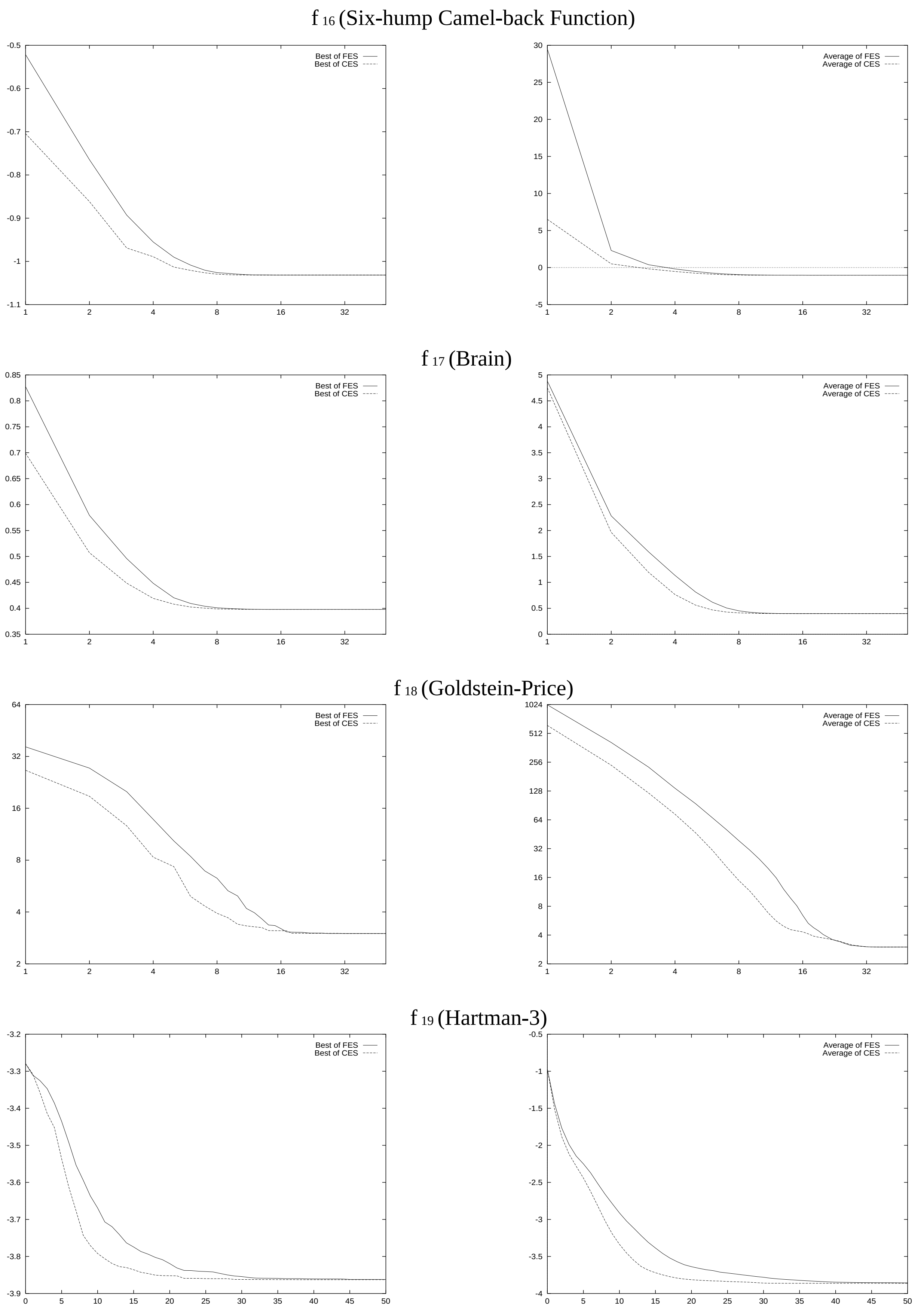


Figure 7: Comparison between CES and FES on f_{16}–f_{19}. The vertical axis is the function value and the horizontal axis is the number of generations.

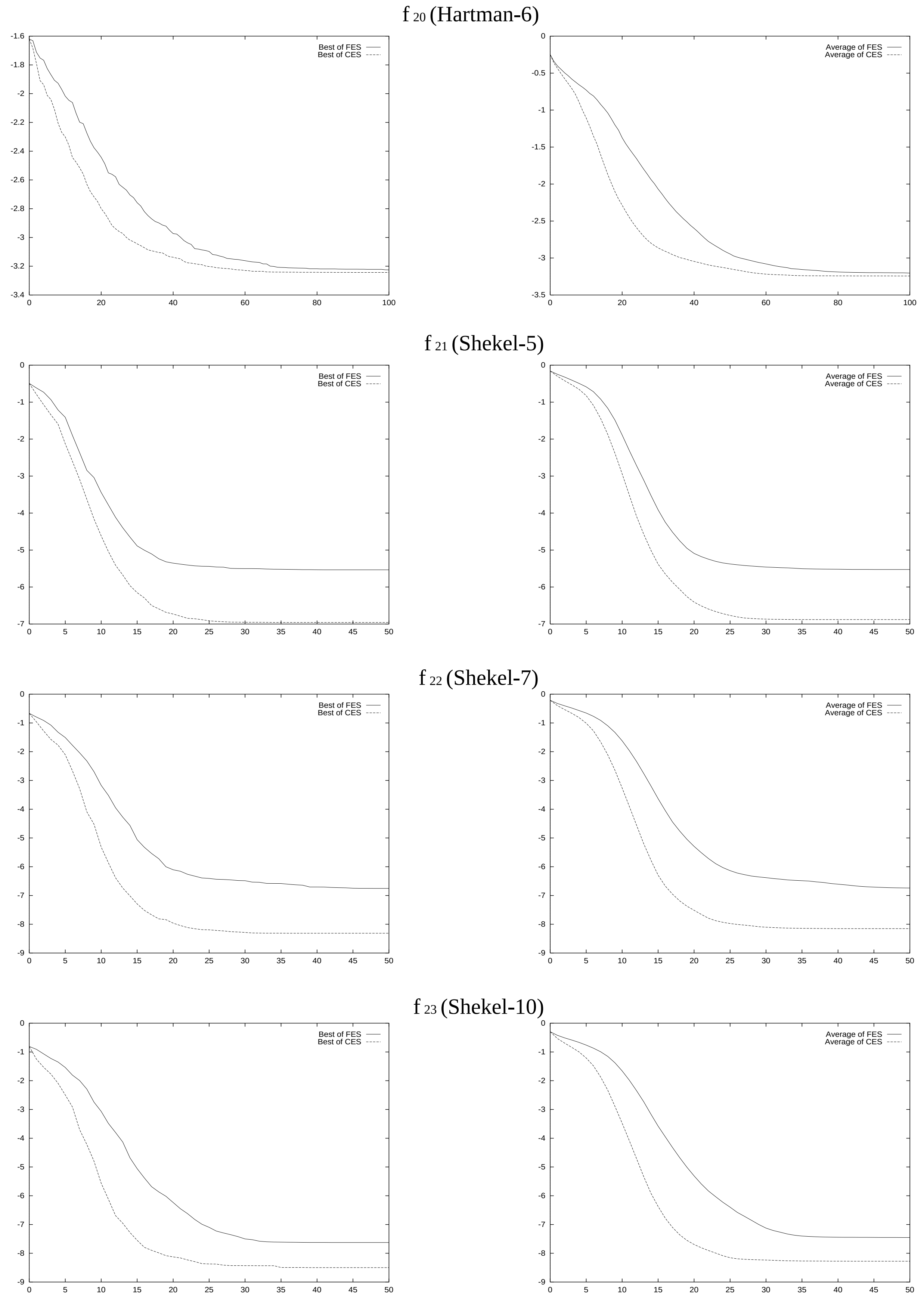


Figure 8: Comparison between CES and FES on f_{20}–f_{23}. The vertical axis is the function value and the horizontal axis is the number of generations.

Table 4: Comparison between CES and FES on f_{14}–f_{23}. The results were averaged over 50 runs. "Mean Best" indicates the mean best function values found in the last generation. "Std Dev" stands for the standard deviation.

Function	Number of	FES		CES		FES−CES
	Generations	Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_{14}	50	1.20	0.63	2.16	1.82	$-3.91^\dagger$
f_{15}	2000	9.7×10^{-4}	4.2×10^{-4}	1.2×10^{-3}	1.6×10^{-5}	$-4.36^\dagger$
f_{16}	50	−1.0316	6.0×10^{-7}	−1.0316	6.0×10^{-7}	0
f_{17}	50	0.398	6.0×10^{-8}	0.398	6.0×10^{-8}	0
f_{18}	50	3.0	0	3.0	0	0
f_{19}	50	−3.86	4.0×10^{-3}	−3.86	1.4×10^{-5}	1.30
f_{20}	100	−3.23	0.12	−3.24	5.7×10^{-2}	0.93
f_{21}	50	−5.54	1.82	−6.96	3.10	$2.81^\dagger$
f_{22}	50	−6.76	3.01	−8.31	3.10	$2.50^\dagger$
f_{23}	50	−7.63	3.27	−8.50	3.14	1.25

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

5-dimensional version of functions f_8–f_{13}. The same pattern as that shown by Figures 5 and 6 was observed. This result shows that dimensionality is not one of the factors which affect FES's and CES's performance on functions f_{14}–f_{23}. The characteristics of these functions are the factors. One of such characteristics might be the number of local minima. Unlike functions f_8–f_{13}, all these functions have just a few local minima. The advantage of FES's long jumps might be weakened in this case since there are not many local minima to escape. Also, fewer local minima imply that most of the optimisation time would be spent on searching in one of the local minima's "basin of attractions," where there is only one minimum. Hence, CES's performance would be very close to or even better than FES's.

Since the goal of FES is to minimise multimodal not unimodal functions, FES's worse performance on functions f_{21} and f_{22} warrants a closer examination. Among 16 multimodal functions tested in this paper, these two were the only cases where FES was outperformed by CES. (Only statistically significant difference is considered in this paper.) Figure 9 shows the 2-dimensional version of function f_{21}. The shape of f_{22} is similar. It can be seen from the figure that f_{21} is rather spiky with some small but deep local minima scattered on a relatively flat area. These small but deep "spikes" cause some difficulties to FES. Neither FES's nor CES's result was close to the global minimum. Both of them seemed to be trapped in some local minimum. However, FES suffered more. This fact appears to contradict our previous discussion which says FEP's long jumps are beneficial, but it does not. Recall the analysis of FEP's and CEP's behaviours on f_1 in Section 4.3.1. It is not difficult to see that long jumps are not always beneficial. It is detrimental when the search points are already close to the global minimum. This turns out to be the case with functions f_{21} and f_{22}.

For functions f_{21} and f_{22}, the range of x_i's are relatively small. Some of the points in the initial populations are already very close to the global minimum. After a few generations, the whole population will be close to the global minimum. In such a situation, long jumps will no longer be beneficial. This can be verified both analytically and empirically. The detailed results were presented elsewhere [9].

4.4 Related Work on Fast Evolutionary Programming

Similar to FES, fast evolutionary programming (FEP) [8, 9] also uses Cauchy mutation. FEP has been tested on the same 23 benchmark functions as described by Table 1. Comparing those results [8, 9] with the results obtained from the current study, it is clear that the difference between FES and CES is very similar to the difference between FEP and CEP. Similar evolutionary patterns were observed from FEP and CEP for the three function categories. The only exceptions were f_3, f_5, f_{15} and f_{23}. For f_3, FES performed worse than CES, while FEP performed better than CEP. For f_5, FES also performed worse than CES, while there was no statistically significant difference between FEP and CEP. For f_{15}, FES performed better than CES, while there was no statistically significant difference between FEP and CEP either. For f_{23}, there was no statistically significant difference between FES and CES, but FEP performed worse than CEP. In

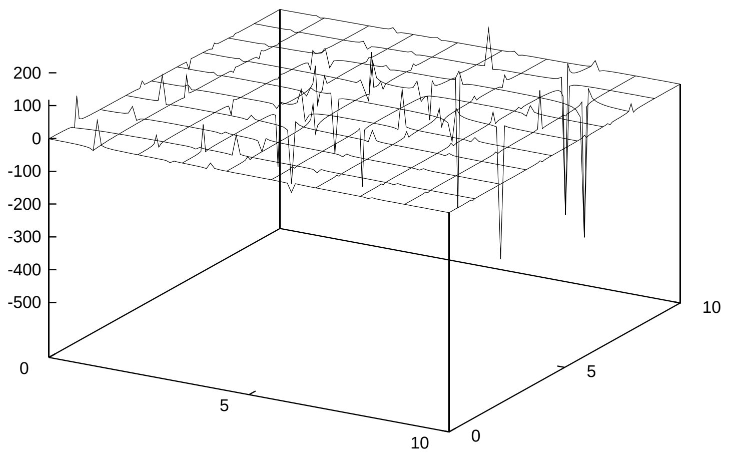


Figure 9: The 2-dimensional version of function f_{21} (Shekel-5).

general, the relationship between FES and CES is very similar to that between FEP and CEP. Since the major difference between EP and ES is their selection schemes, the results of FES and FEP indicate that Cauchy mutation is a very robust search operator which can work with different selection schemes. In fact, FES's performance can be further improved by mixing Cauchy and Gaussian mutations. Such improvement has been proven to be very successful in the case of FEP [9].

5 Other Variants of Evolution Strategies

The previous sections only present some results with a simple version of evolution strategies. This section investigates

1. whether changing the order of mutating objective variables and strategy parameters would make much difference between CES's and FES's performance,
2. whether FES still performs better if a different scale parameter t is used in the Cauchy distribution, and
3. whether FES still performs better if recombination is used.

5.1 The Order of Mutations

We have run the experiments with a different order of mutating objective variables and strategy parameters. Table 5 shows the results of CES and FES, where the strategy parameter (Eq.2) was mutated first, for the three representative functions. No recombination was used in CES and FES.

For f_1, which is a typical function in the first group of the 23 benchmark functions, FES was outperformed by CES significantly. For f_{10}, which is a typical function in the second group, FES performed significantly better than CES. For f_{23}, which is a typical function in the third group, FES was again outperformed by CES. These observations are the same as what we observed when we mutated the objective variables first. That is, changing the order of mutation has little impact on the observations we made in Section 4 about CES's and FES's relative performance, although their absolute (i.e., individual) performance may have changed slightly.

5.2 Cauchy Mutation with a Different Scale Parameter

All the previous experiments assumed scale parameter $t = 1$ in the Cauchy distribution. Tables 6 and 7 show the results of CEP and FEP on f_{10} when different values of the scale parameter were used. Table 6

Table 5: Comparison between CES and FES with no recombination (only changing the order of Eq.1 and Eq.2) on f_1, f_{10} and f_{23}. The results were averaged over 50 runs.

F	Gen's	FES		CES		FES−CES
		Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_1	750	2.0×10^{-4}	2.3×10^{-5}	2.4×10^{-5}	2.8×10^{-6}	$52.47^{\dagger}$
f_{10}	750	1.0×10^{-2}	9.4×10^{-4}	8.50	2.89	$-20.75^{\dagger}$
f_{23}	50	-8.86	2.92	-9.75	2.19	$1.76^{\dagger}$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

shows the results when the objective variables were mutated first, while Table 7 shows the results when the strategy parameters were mutated first. It is interesting to note that FES still outperforms CES for both $t = 0.5$ and $t = 1.5$. However, the performance of FES deteriorates as t increases for this particular problem. A general conclusion about the relationship between the scale parameter and the algorithm's performance is difficult to draw because it is problem-dependent.

Table 6: Comparison between CES and FES for different scale parameters with no recombination. Objective variables were mutated first. The experiment was based on 50 runs using f_{10}.

Scaling	Gen's	FES		CES		FES−CES
		Mean Best	Std Dev	Mean Best	Std Dev	t-test
0.5	750	5.9×10^{-3}	7.5×10^{-4}	9.72	2.75	$-25.01^{\dagger}$
1.0	750	1.2×10^{-2}	1.8×10^{-3}	9.07	2.84	$-22.51^{\dagger}$
1.5	750	0.42	2.82	7.61	2.83	$-14.92^{\dagger}$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

Table 7: Comparison between CES and FES for different scale parameters with no recombination. Strategy parameters were mutated first. The experiment was based on 50 runs using f_{10}.

Scaling	Gen's	FES		CES		FES−CES
		Mean Best	Std Dev	Mean Best	Std Dev	t-test
0.5	750	5.2×10^{-3}	4.4×10^{-4}	8.47	3.07	$-19.52^{\dagger}$
1.0	750	1.0×10^{-2}	9.4×10^{-4}	8.50	2.89	$-20.75^{\dagger}$
1.5	750	0.81	3.95	6.79	2.74	$-9.29^{\dagger}$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

5.3 Evolution Strategies with Recombination

Although evolution strategies emphasise mutation, they do use recombination. The current wisdom is to use discrete recombination on the objective variables and global intermediate recombination on the strategy parameters. Table 8 shows the results of CES and FES with aforementioned recombinations. The same recombinations were implemented for both algorithms.

The results in Table 8 reveal that FES performed poorly against CES for all three functions when recombination was used. The introduction of recombination to FES has significantly worsened FES's performance, while CES's performance improved greatly using the recombinations. Our preliminary analysis of such phenomena indicates that the search step size of different operators plays an important role in determining the performance of an algorithm. As pointed out earlier [9], Cauchy mutation has a much larger search step size than Gaussian mutation. A large search step size is beneficial when the current search point is far away from

Table 8: Comparison between CES and FES with recombination (discrete recombination on the objective variables and global intermediate recombination on the strategy parameters). The strategy parameters were mutated first. All results were averaged over 50 runs.

F	Gen's	FES		CES		FES−CES
		Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_1	750	27.94	34.52	2.2×10^{-5}	2.4×10^{-6}	$5.72^{\dagger}$
f_{10}	750	4.64	1.49	3.4×10^{-3}	2.4×10^{-4}	$22.03^{\dagger}$
f_{23}	50	−10.34	0.63	−10.54	1.4×10^{-4}	$2.22^{\dagger}$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

the global optimum, which is often the case at the beginning of search. When the current search point is close to the global optimum, which is likely towards the end of search, large search step sizes are detrimental to search.

The two recombinations implemented in our experiments have very large search step sizes, especially the global intermediate recombination. Using both Cauchy mutation and these recombinations imply a huge search step size which would be undesirable for the functions we studied. That is why the introduction of recombination into FES brought no benefit at all. On the other hand, Gaussian mutation's search step size is relatively small. The introduction of recombination into CES greatly increased CES's search step size and thus its performance. In a sense, introducing recombination to CES has a similar effect as replacing Gaussian mutation by Cauchy mutation. Both increase the algorithm's search step size.

To support our arguments and preliminary analysis, another set of experiments were carried out where only the discrete recombination was used on both objective variables and strategy parameters in FES. (The search step size of the discrete recombination is much smaller than the global intermediate recombination.) CES was kept the same as before. Table 9 shows the results of the experiment. It is clear that FES's performance has improved dramatically after this minor change. The results demonstrate that the search step size of Cauchy mutation is sufficiently large. There might not be any benefit of using recombination on the strategy parameters.

Table 9: Comparison between CES and FES with recombination (discrete recombination on the objective variables and global intermediate recombination on the strategy parameters for CES, and discrete recombination on both objective variables and strategy parameters for FES). The strategy parameters were mutated first. All results were averaged over 50 runs.

F	Gen's	FES		CES		FES−CES
		Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_1	750	1.3×10^{-4}	1.8×10^{-5}	2.2×10^{-5}	2.4×10^{-6}	$39.67^{\dagger}$
f_{10}	750	8.3×10^{-3}	6.6×10^{-4}	3.4×10^{-3}	2.4×10^{-4}	$49.87^{\dagger}$
f_{23}	50	−10.22	1.03	−10.54	1.4×10^{-4}	$2.15^{\dagger}$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

5.3.1 The Impact of Different Scale Parameters

Table 10 shows the impact of the scale parameter in Cauchy distribution on FES's performance when recombination is used. It indicates that different scale parameters did not change the global picture very much, although it did affect FES's performance slightly.

6 Conclusions

This paper proposes a new (μ, λ)-ES algorithm (i.e., FES) using Cauchy mutation. Extensive empirical studies on 23 benchmark problems (up to 30 dimensions) were carried out to evaluate the performance of

Table 10: Comparison between CES and FES with recombination (discrete recombination on the objective variables and global intermediate recombination on the strategy parameters for CES, and discrete recombination on both objective variables and strategy parameters for FES), when a different scale parameter is used. The strategy parameters were mutated first. All results were averaged over 50 runs on f_{10}.

Scaling	Gen's	FES		CES		FES−CES
		Mean Best	Std Dev	Mean Best	Std Dev	t-test
0.5	750	4.2×10^{-3}	3.0×10^{-4}	1.7×10^{-3}	9.7×10^{-5}	$59.03^{\dagger}$
1.0	750	8.3×10^{-3}	6.6×10^{-4}	3.4×10^{-3}	2.4×10^{-4}	$49.87^{\dagger}$
1.5	750	1.21	4.78	5.3×10^{-3}	3.2×10^{-4}	$1.78^{\dagger}$

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

FES. For multimodal functions with many local minima, FES outperforms CES consistently. For unimodal functions, CES appears to perform slightly better. However, FES is much better at dealing with plateaus. For multimodal functions with only a few local minima, the performance of FES and CES is very similar.

The main reason for the difference in performance between FES and CES is due to the difference in their probabilities of making long jumps. Long jumps are beneficial when the current search points are far away from the global minimum, while detrimental when the current search points get close to the global minimum. Recent analytical results and further empirical studies [9] support the preliminary analyses presented in this paper.

According to recent work on analysing EAs using step sizes of search operators [21], the impact of a search operator on the algorithm's search depends heavily on its search step size. It may be conjectured that recombination would play a major role in FES only if its search step size is larger than that of Cauchy mutation.

7 Appendix: Benchmark Functions

7.1 Sphere Model

$$f_1(x) = \sum_{i=1}^{30} x_i^2$$

$$-100 \leq x_i \leq 100, \quad \min(f_1) = f_1(0,\ldots,0) = 0$$

7.2 Schwefel's Problem 2.22

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$$

$$-10 \leq x_i \leq 10, \quad \min(f_2) = f_2(0,\ldots,0) = 0$$

7.3 Schwefel's Problem 1.2

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$$

$$-100 \leq x_i \leq 100, \quad \min(f_3) = f_3(0,\ldots,0) = 0$$

7.4 Schwefel's Problem 2.21

$$f_4(x) = \max_i\{|x_i|, 1 \le i \le 30\}$$

$$-100 \le x_i \le 100, \quad \min(f_4) = f_4(0,\ldots,0) = 0$$

7.5 Generalised Rosenbrock's Function

$$f_5(x) = \sum_{i=1}^{29}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

$$-30 \le x_i \le 30, \quad \min(f_5) = f_5(1,\ldots,1) = 0$$

7.6 Step Function

$$f_6(x) = \sum_{i=1}^{30}\left(\lfloor x_i + 0.5 \rfloor\right)^2$$

$$-100 \le x_i \le 100, \quad \min(f_6) = f_6(0,\ldots,0) = 0$$

7.7 Quartic Function with Noise

$$f_7(x) = \sum_{i=1}^{30} i x_i^4 + random[0,1)$$

$$-1.28 \le x_i \le 1.28, \quad \min(f_7) = f_7(0,\ldots,0) = 0$$

7.8 Generalised Schwefel's Problem 2.26

$$f_8(x) = -\sum_{i=1}^{30}\left(x_i \sin\left(\sqrt{|x_i|}\right)\right)$$

$$-500 \le x_i \le 500, \quad \min(f_8) = f_8(420.9687,\ldots,420.9687) = -12569.5$$

7.9 Generalised Rastrigin's Function

$$f_9(x) = \sum_{i=1}^{30}[x_i^2 - 10\cos(2\pi x_i) + 10)]$$

$$-5.12 \le x_i \le 5.12, \quad \min(f_9) = f_9(0,\ldots,0) = 0$$

7.10 Ackley's Function

$$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30}x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos 2\pi x_i\right) + 20 + e$$

$$-32 \le x_i \le 32, \quad \min(f_{10}) = f_{10}(0,\ldots,0) = 0$$

7.11 Generalised Griewank Function

$$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$-600 \le x_i \le 600, \quad \min(f_{11}) = f_{11}(0,\ldots,0) = 0$$

7.12 Generalised Penalised Functions

$$\begin{aligned} f_{12}(x) &= \frac{\pi}{30}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n-1)^2\right\} \\ &\quad + \sum_{i=1}^{30} u(x_i,10,100,4) \end{aligned}$$

$$-50 \le x_i \le 50, \quad \min(f_{12}) = f_{12}(1,\ldots,1) = 0$$

$$\begin{aligned} f_{13}(x) &= 0.1\left\{\sin^2(\pi 3x_1 + \sum_{i=1}^{29}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})] + (x_n-1)[1+\sin^2(2\pi x_{30})]\right\} \\ &\quad + \sum_{i=1}^{30} u(x_i,5,100,4) \end{aligned}$$

$$-50 \le x_i \le 50, \quad \min(f_{13}) = f_{13}(1,\ldots,1) = 0$$

where

$$u(x_i,a,k,m) = \begin{cases} k(x_i-a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i-a)^m, & x_i < -a. \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i+1)$$

7.13 Shekel's Foxholes Function

$$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$$

$$-65.536 \le x_i \le 65.536, \quad \min(f_{14}) = f_{14}(-32,-32) \approx 1$$

where

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

7.14 Kowalik's Function

$$f_{15}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$$

$$-5 \le x_i \le 5, \quad \min(f_{15}) \approx f_{15}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075$$

Table 11: Kowalik's Function f_{15}

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

7.15 Six-hump Camel-Back Function

$$f_{16} = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

$$-5 \le x_i \le 5$$

$$x_{min} = (0.08983, -0.7126),\ (-0.08983, 0.7126)$$

$$\min(f_{16}) = -1.0316285$$

7.16 Branin Function

$$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

$$-5 \le x_1 \le 10, \quad 0 \le x_2 \le 15$$

$$x_{min} = (-3.142, 12.275),\ (3.142, 2.275),\ (9.425, 2.425)$$

$$\min(f_{17}) = 0.398$$

7.17 Goldstein-Price Function

$$\begin{aligned} f_{18}(x) &= [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\ &\quad \times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned}$$

$$-2 \le x_i \le 2, \quad \min(f_{18}) = f_{18}(0, -1) = 3$$

7.18 Hartman's Family

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right]$$

with $n = 3, 6$ for $f_{19}(x)$ and $f_{20}(x)$, respectively, $0 \le x_j \le 1$. The coefficients are defined by Tables 12 and 13, respectively.

For $f_{19}(x)$ the global minimum is equal to -3.86 and it is reached at the point $(0.114, 0.556, 0.852)$. For $f_{20}(x)$ the global minimum is -3.32 at the point $(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)$.

Table 12: Hartman Function f_{19}

i	$a_{ij}, j=1,2,3$			c_i	$p_{ij}, j=1,2,3$		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.038150	0.5743	0.8828

Table 13: Hartman Function f_{20}

i	$a_{ij}, j=1,\cdots,6$						c_i	$p_{ij}, j=1,\cdots,6$					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

7.19 Shekel's Family

$$f(x) = -\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1}$$

with $m = 5, 7 10$ for $f_{21}(x)$, $f_{22}(x)$ and $f_{23}(x)$, respectively, $0 \le x_j \le 10$.

Table 14: Shekel Functions f_{21}, f_{22}, f_{23}

i	$a_{ij}, j=1,\cdots,4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

These functions have 5, 7 and 10 local minima for $f_{21}(x)$, $f_{22}(x)$, and $f_{23}(x)$, respectively. $x_{local\_opt} \approx a_i$, $f(x_{local\_opt}) \approx 1/c_i$ for $1 \le i \le m$. The coefficients are defined by Table 14.

Acknowledgement — The authors are grateful to Professors Zbigniew Michalewicz and Marc Schoenauer for their constructive comments on the earlier version of this paper.

References

[1] X. Yao and Y. Liu, "Fast evolution strategies," in *Evolutionary Programming VI: Proc. of the Sixth Annual Conference on Evolutionary Programming* (P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, eds.), vol. 1213 of *Lecture Notes in Computer Science*, (Berlin), pp. 151–161, Springer-Verlag, 1997.

[2] D. B. Fogel, "An introduction to simulated evolutionary optimisation," *IEEE Trans. on Neural Networks*, vol. 5, no. 1, pp. 3–14, 1994.

[3] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[4] H.-P. Schwefel, *Evolution and Optimum Seeking.* New York: John Wiley & Sons, 1995.

[5] T. Bäck and H.-P. Schwefel, "Evolutionary computation: an overview," in *Proc. of the 1996 IEEE Int'l Conf. on Evolutionary Computation (ICEC'96), Nagoya, Japan*, pp. 20–29, IEEE Press, New York, NY 10017-2394, 1996.

[6] X. Yao, "An overview of evolutionary computation," *Chinese Journal of Advanced Software Research (Allerton Press, Inc., New York, NY 10011)*, vol. 3, no. 1, pp. 12–29, 1996.

[7] T. Bäck, *Evolutionary Algorithms in Theory and Practice.* New York: Oxford University Press, 1996.

[8] X. Yao and Y. Liu, "Fast evolutionary programming," in *Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming* (L. J. Fogel, P. J. Angeline, and T. Bäck, eds.), (Cambridge, MA), pp. 451–460, The MIT Press, 1996.

[9] X. Yao, G. Lin, and Y. Liu, "An analysis of evolutionary algorithms based on neighbourhood and step sizes," in *Evolutionary Programming VI: Proc. of the Sixth Annual Conference on Evolutionary Programming* (P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, eds.), vol. 1213 of *Lecture Notes in Computer Science*, (Berlin), pp. 297–307, Springer-Verlag, 1997.

[10] C. Kappler, "Are evolutionary algorithms improved by large mutations?," in *Parallel Problem Solving from Nature (PPSN) IV* (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), vol. 1141 of *Lecture Notes in Computer Science*, (Berlin), pp. 346–355, Springer-Verlag, 1996.

[11] H. H. Szu and R. L. Hartley, "Nonconvex optimization by fast simulated annealing," *Proceedings of IEEE*, vol. 75, pp. 1538–1540, 1987.

[12] X. Yao, "A new simulated annealing algorithm," *Int. J. of Computer Math.*, vol. 56, pp. 161–168, 1995.

[13] D. K. Gehlhaar and D. B. Fogel, "Tuning evolutionary programming for conformationally flexible molecular docking," in *Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming* (L. J. Fogel, P. J. Angeline, and T. Bäck, eds.), pp. 419–429, MIT Press, Cambridge, MA, 1996.

[14] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 2. John Wiley & Sons, Inc., 2nd ed., 1971.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in FORTRAN, 2 edn.* Cambridge CB2 1RP, UK: Cambridge University Press, 1992.

[16] L. Devroye, *Non-Uniform Random Variate Generation.* New York, NY 10010: Springer-Verlag, 1986.

[17] D. B. Fogel, *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling.* Needham Heights, MA 02194: Ginn Press, 1991.

[18] A. Törn and A. Žilinskas, *Global Optimisation.* Berlin: Springer-Verlag, 1989. Lecture Notes in Computer Science, Vol. 350.

[19] L. Ingber and B. Rosen, "Genetic algorithms and very fast simulated reannealing: a comparison," *Mathl. Comput. Modelling*, vol. 16, no. 11, pp. 87–100, 1992.

[20] A. Dekkers and E. Aarts, "Global optimization and simulated annealing," *Math. Programming*, vol. 50, pp. 367–393, 1991.

[21] G. Lin and X. Yao, "Analysing crossover operators by search step size," in *Proc. of the 1997 IEEE Int'l Conf. on Evolutionary Computation (ICEC'97), Indianapolis, USA*, pp. 107–110, IEEE Press, New York, NY, April 1997.