Lecture 4: Case Studies of Search Operators and Representation

CSE5012: Evolutionary Computation and Its Applications

Xin Yao

CSE, SUSTech

6 March 2023

Review of Previous Lectures

- **Main ideas of EAs**
- **Different types of EAs**
- **Evolutionary operators**
  - **Crossover/recombination operators**
  - **Mutation operators**
- **Selection schemes**
  - **Parent selection**
  - **Survivor selection**

Outline of This Lecture

Global Optimisation by Mutation-Based EAs I

1. **Generate the initial population of μ individuals, each individual is a real-valued vector, $\mathbf{x}_i$ ($i = 1, \ldots, \mu$). And set $k = 1$.**
2. **Evaluate the fitness of each individual.**
3. **Each individual $\mathbf{x}_i$ ($i = 1, \ldots, \mu$) generates a single offspring $\mathbf{x}'_i$: for $j = 1, \cdots, n$,**

$$x'_i(j) \quad = \quad x_i(j) + \mathcal{N}_j(0, \sigma^2) \tag{1}$$

where

- **n is the dimensionality,**
- **$x_i(j)$ and $x'_i(j)$ denote the j^{th} component of the vectors $\mathbf{x}_i$ and $\mathbf{x}'_i$, respectively,**
- **$\mathcal{N}(0, \sigma^2)$ denotes a Gaussian distributed one-dimensional random number with mean 0 and standard deviation σ,**
- **$\mathcal{N}_j(0, \sigma^2)$ indicates that the random number is generated anew for each value of j.**

Global Optimisation by Mutation-Based EAs II

4. **Evaluate the fitness of each offspring.**
5. **Apply round robin tournament selection: For each individual, q opponents are chosen randomly from all the parents and offspring with an equal probability. For each comparison, if the individual's fitness is no greater than the opponent's, it receives a "win".**
6. **Select the μ best individuals (from 2μ) that have the most wins to be the next generation.**
7. **Stop if the stopping criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.**

Example Visualisation of $\mathcal{N}(0, \sigma^2)$

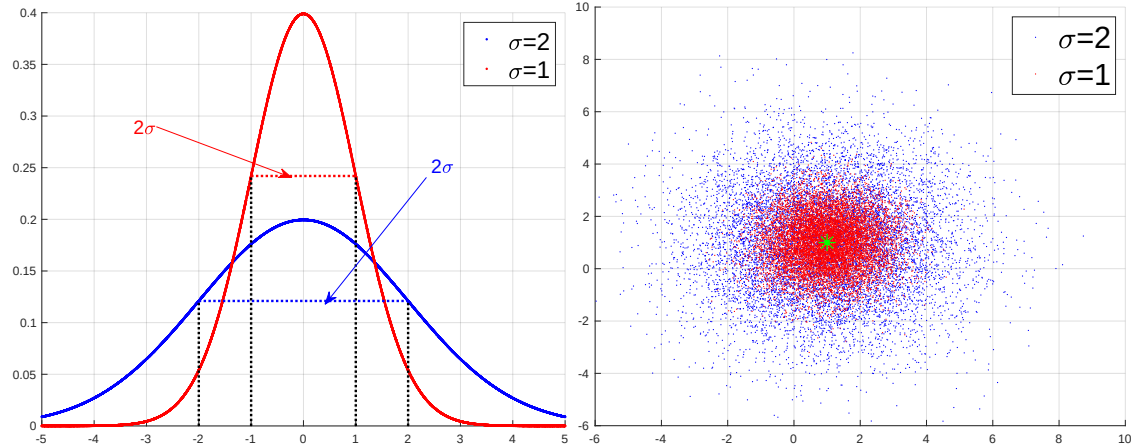


Figure 1: Left: Gaussian distributions $\mathcal{N}(0, 1)$ and $\mathcal{N}(0, 2)$.
Right: 10000 offspring generated using Eq. (1) with perturbation $\sim \mathcal{N}(0, 1)$ and perturbation $\sim \mathcal{N}(0, 2)$ around the parent $\mathbf{x}$.

Why $\mathcal{N}(0, \sigma^2)$? How to Determine σ?

1. **The standard deviation of the Gaussian distribution determines the search step-size of the mutation. It is a crucial parameter.**
2. **Unfortunately, the optimal search step size is problem-dependent.**
3. **Even for a single problem, different search stages require different search step sizes.**
4. **Self-adaptation can be used to get around this problem partially.**
   Self-adaptation is one of the key features of EAs, which usually implies that parameters are encoded as part of chromosomes.

Function Optimisation by Classical EP (CEP)

Each individual $(\mathbf{x}_i, \eta_i)$**,** $i = 1, \cdots, \mu$**, creates a single offspring** $(\mathbf{x}_i{}', \eta_i{}')$ **by: for** $j = 1, \cdots, n$**,**

$$x_i{}'(j) = x_i(j) + \eta_i(j)\mathcal{N}_j(0,1), \quad (2)$$

$$\eta_i{}'(j) = \eta_i(j)\exp(\tau'\mathcal{N}(0,1) + \tau\mathcal{N}_j(0,1)) \quad (3)$$

where

- $x_i(j)$**,** $x_i{}'(j)$**,** $\eta_i(j)$ **and** $\eta_i{}'(j)$ **denote the** j**-th components of the vectors** $\mathbf{x}_i$**,** $\mathbf{x}_i{}'$**,** η_i **and** $\eta_i{}'$**, respectively,**
- $\mathcal{N}(0,1)$ **denotes a normally distributed one-dimensional random number with mean zero and standard deviation one,**
- $\mathcal{N}_j(0,1)$ **indicates that the random number is generated anew for each value of** j**,**
- **The parameters** τ **and** τ' **have commonly been set to** $\frac{1}{\sqrt{2\sqrt{n}}}$ **and** $\frac{1}{\sqrt{2n}}$**.**

Fast EP

- **The idea comes from fast simulated annealing.**
- **Use a Cauchy, instead of Gaussian, random number in Eq.(2) to *generate* a new offspring. That is,**

$$x_i'(j) = x_i(j) + \eta_i(j)\delta_j \tag{4}$$

where δ_j is an Cauchy random number with the scale parameter $t = 1$, and is generated anew for each value of j.

- **Everything else, including Eq. (3), are kept unchanged in order to evaluate the impact of Cauchy random numbers.**

Cauchy Distribution

Its density function is

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

$t > 0$ **is a scale parameter. The corresponding distribution function is**

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

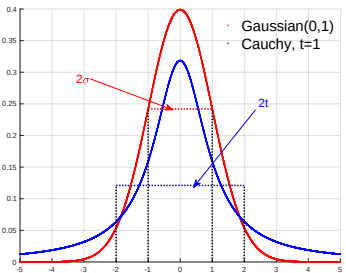


Figure 2: Gaussian and Cauchy density functions.

Benchmark Functions

- **Benchmark functions:**
  - **23 functions were used in our computational studies [9]. They have different characteristics.**
  - **Some have a relatively high dimension (e.g. 30).**
  - **Some have many local optima.**

- **Aim at minimising these functions here.**
  $\rightarrow$ Smaller values are better.

X. Yao, Y. Liu and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, 3(2):82-102, July 1999.

Benchmark Functions at a Closer Look.

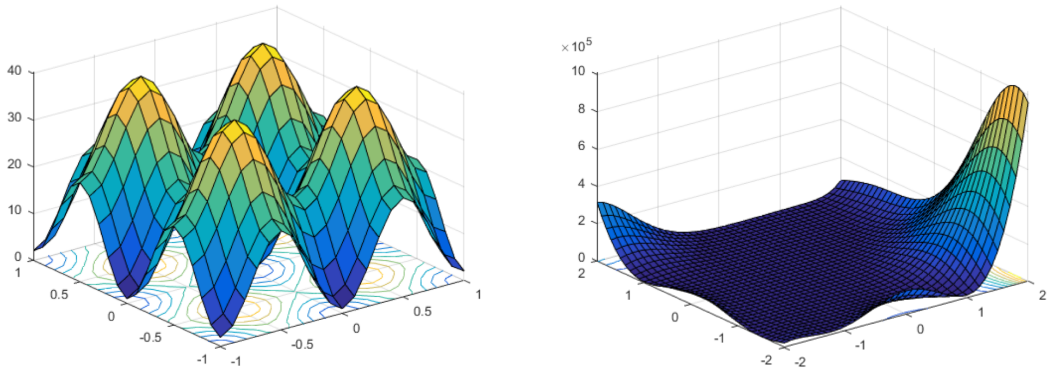


Figure 3: Functions f_9 (left) and f_{18} (right).

Experimental Setup

- **Population size 100.**
- **Tournament size 10 for selection.**
- **All experiments were run 50 times, i.e., 50 trials.**
- **Initial populations were the same for CEP and FEP.**

Questions

1. Do you remember what is a tournament size?
2. Why repeating the experiments many times?
3. Why using same initial populations for CEP and FEP?

Experiments on Unimodal Functions f_1-f_7

- **Unimodal functions: f_1-f_5**
- **f_6 is the step function (one minimum, discontinuous).**
- **f_7 is a noisy quartic function, where $random[0, 1)$ is a uniformly distributed random variable in [0, 1).**

Test function	n	S	f_{min}
$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	**30**	$[-100, 100]^n$	**0**
$f_2(\mathbf{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	**30**	$[-100, 100]^n$	**0**
$f_3(\mathbf{x}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	**30**	$[-10, 10]^n$	**0**
$f_4(\mathbf{x}) = \max_i \{\lvert x_i \rvert, 1 \leq i \leq n\}$	**30**	$[-100, 100]^n$	**0**
$f_5(\mathbf{x}) = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	**30**	$[-100, 100]^n$	**0**
$f_6(\mathbf{x}) = \sum_{i=1}^{n} (x_i + 0.5)^2$	**30**	$[-30, 30]^n$	**0**
$f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$	**30**	$[-1.28, 1.28]^n$	**0**

Experiments on Unimodal Functions f_1-f_7

Visualisation of Some Functions

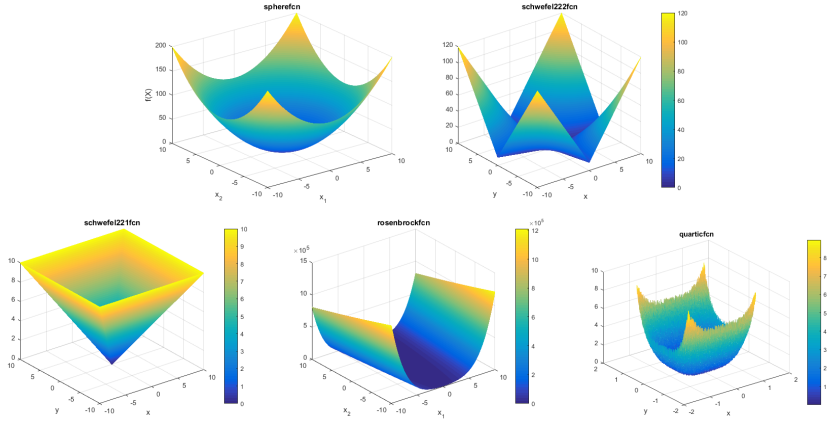


Figure 4: Functions f_1, f_2, f_4, f_5 and f_7. Sources: `http://benchmarkfcns.xyz/fcns`

Experiments on Unimodal Functions f_1-f_7

Numerical Results

F	No. of	FEP		CEP		FEP−CEP
	Gen's	Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_1	1500	5.7×10^{-4}	1.3×10^{-4}	2.2×10^{-4}	5.9×10^{-4}	$4.06^{\dagger}$
f_2	2000	8.1×10^{-3}	7.7×10^{-4}	2.6×10^{-3}	1.7×10^{-4}	$49.83^{\dagger}$
f_3	5000	1.6×10^{-2}	1.4×10^{-2}	5.0×10^{-2}	6.6×10^{-2}	$-3.79^{\dagger}$
f_4	5000	0.3	0.5	2.0	1.2	$-8.25^{\dagger}$
f_5	20000	5.06	5.87	6.17	13.61	$-0.52^{\dagger}$
f_6	1500	0	0	577.76	1125.76	$-3.67^{\dagger}$
f_7	3000	7.6×10^{-3}	2.6×10^{-3}	1.8×10^{-2}	6.4×10^{-3}	$-10.72^{\dagger}$

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

Observations:

- **FEP performed better than CEP on f_3-f_7.**
- **CEP was better for f_1 and f_2.**
- **FEP converged faster, even for f_1 and f_2 (for a long period).**

Experiments on Unimodal Functions f_1-f_7

Evolutionary Curves for f_1 and f_2 and f_7

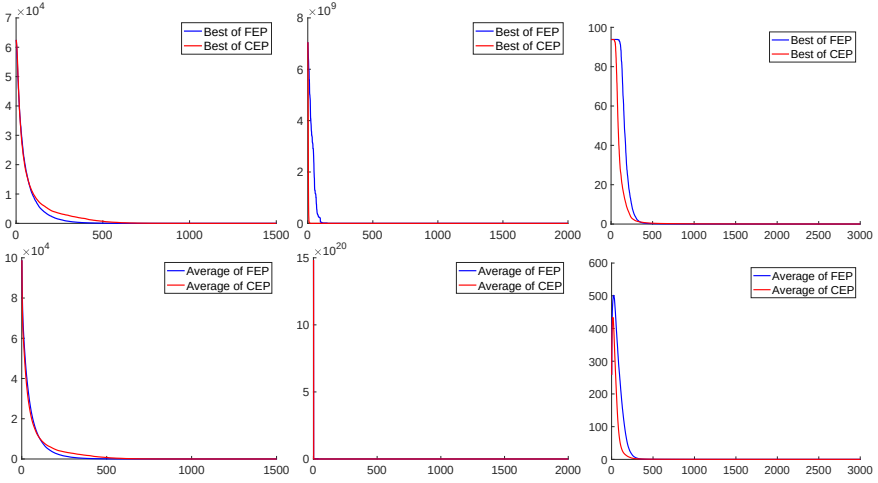


Figure 5: Evolutionary curves of CEP and FEP for f_1 (left), f_2 (middle) and f_7 (right). Plot from 1^{st} generation.

Experiments on Unimodal Functions f_1-f_7

Evolutionary Curves for f_1 and f_2 and f_7

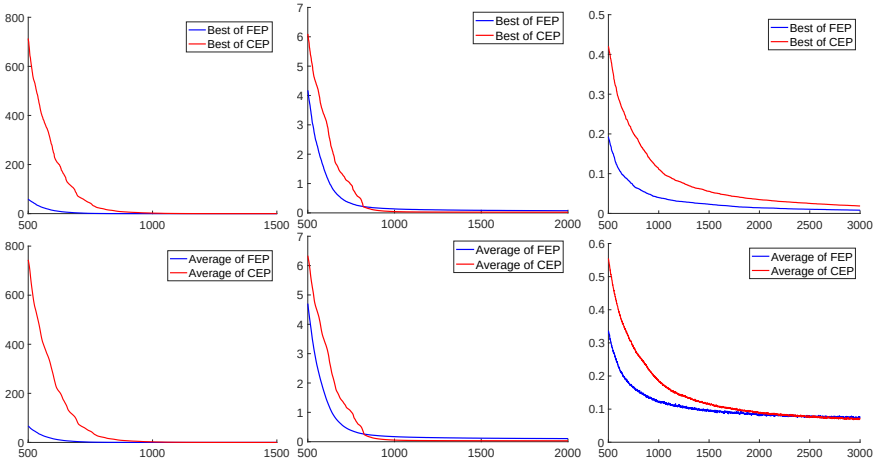


Figure 6: Evolutionary curves of CEP and FEP for f_1 (left), f_2 (middle) and f_7 (right). Plot from 500^{th} generation.

Statistical Tests and Comparisons

- **Student's T-test[1] is used to determine if two sets of data are significantly different from each other.**
- **T-test assumes that the data are normally distributed. This may not be true in this case.**
- **A better way to do statistical tests when comparing EAs is to use a non-parametric method, e.g., Wilcoxon signed-rank test.**

[1] Why it is called "Student's T-test"? An interesting history to know:)

Experiments on Multimodal Functions f_8-f_{13}

- **Multimodal functions**
- **#local minima increases exponentially with n**

Test function	n	S	f_{min}
$f_8(\mathbf{x}) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	**30**	$[-500, 500]^n$	**-12569.5**
$f_9(\mathbf{x}) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10)]$	**30**	$[-5.12, 5.12]^n$	**0**
$f_{10}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$	**30**	$[-32, 32]^n$	**0**
$f_{11}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	**30**	$[-600, 600]^n$	**0**
$f_{12}(\mathbf{x}) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$, $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$	**30**	$[-50, 50]^n$	**0**
$f_{13}(\mathbf{x}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	**30**	$[-50, 50]^n$	**0**

Experiments on Unimodal Functions f_8-f_{13}

Visualisation of Some Functions

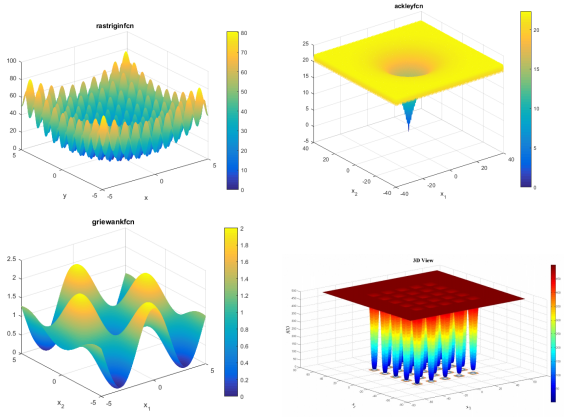


Figure 7: Functions f_9, f_{10}, f_{11} and f_{13}. Sources: `http://benchmarkfcns.xyz/fcns`, `https://www.al-roomi.org/benchmarks`

Experiments on Multimodal Functions f_8-f_{13}

Numerical Results

F	No. of Gen's	FEP Mean Best	FEP Std Dev	CEP Mean Best	CEP Std Dev	FEP−CEP t-test
f_8	**9000**	-12554.5	52.6	-7917.1	634.5	$-51.39^{\dagger}$
f_9	**5000**	4.6×10^{-2}	1.2×10^{-2}	89.0	23.1	$-27.25^{\dagger}$
f_{10}	**1500**	1.8×10^{-2}	2.1×10^{-3}	9.2	2.8	$-23.33^{\dagger}$
f_{11}	**2000**	1.6×10^{-2}	2.2×10^{-2}	8.6×10^{-2}	0.12	$-4.28^{\dagger}$
f_{12}	**1500**	9.2×10^{-6}	3.6×10^{-6}	1.76	2.4	$-5.29^{\dagger}$
f_{13}	**1500**	1.6×10^{-4}	7.3×10^{-5}	1.4	3.7	$-2.76^{\dagger}$

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

Observations:

- **FEP converged faster to a better solution.**
- **FEP seemed to deal with many local minima well.**

Experiments on Unimodal Functions f_8-f_{13}

Evolutionary Curves for f_9-f_{11}

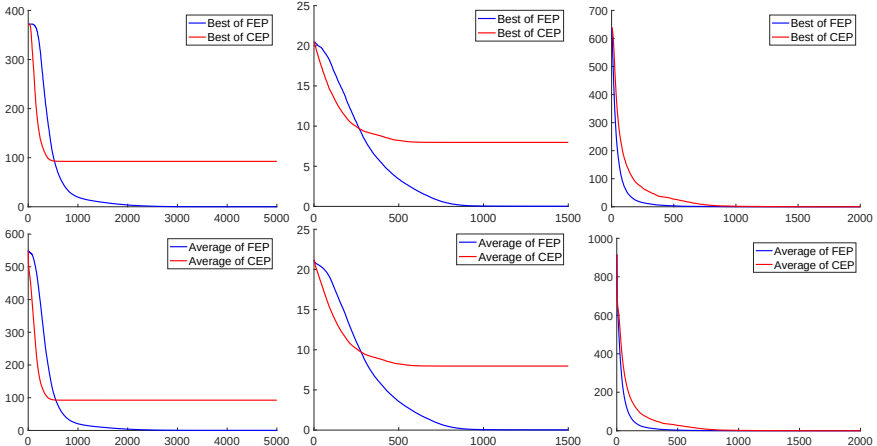


Figure 8: Evolutionary curves of CEP and FEP for f_9 (left), f_{10} (middle) and f_{11} (right).

Experiments on Multimodal Functions f_{14}-f_{23}

- **Multimodal functions**
- **Low-dimensional**
- **Only a few local minima**

Test function	n	S	f_{min}
$f_{14}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	**2**	$[-65.536, 65.536]^n$	**1**
$f_{15}(\mathbf{x}) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$	**4**	$[-5, 5]^n$	**0.0003075**
$f_{16}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	**2**	$[-5, 5]^n$	**-1.0316285**
$f_{17}(\mathbf{x}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	**2**	$[-5, 10] \times [0, 15]$	**0.398**
$f_{18}(\mathbf{x}) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	**2**	$[-2, 2]^n$	**3**
$f_{19}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{4} a_{ij}(x_j - p_{ij})^2\right]$	**4**	$[0, 1]^n$	**-3.86**
$f_{20}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right]$	**6**	$[0, 1]^n$	**-3.32**
$f_{21}(\mathbf{x}) = -\sum_{i=1}^{5}[(x - a_i)^T(\mathbf{x} - a_i) + c_i]^{-1}$	**4**	$[0, 10]^n$	$-1/c_1$
$f_{22}(\mathbf{x}) = -\sum_{i=1}^{7}[(x - a_i)^T(\mathbf{x} - a_i) + c_i]^{-1}$	**4**	$[0, 10]^n$	$-1/c_1$
$f_{23}(\mathbf{x}) = -\sum_{i=1}^{10}[(x - a_i)^T(\mathbf{x} - a_i) + c_i]^{-1}$	**4**	$[0, 10]^n$	$-1/c_1$
where $c_1 = 0.1$			

Experiments on Unimodal Functions f_{14}-f_{23}

Visualisation of Function

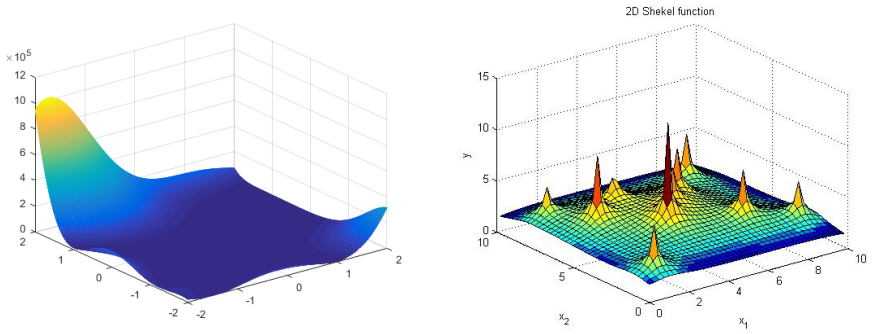


Figure 9: Functions f_{17} (Goldstein-Price, left) and $-f_{23}$ (Shekel, right). Sources: `http://benchmarkfcns.xyz/fcns`, `https://en.wikipedia.org/wiki/Shekel_function`

Experiments on Multimodal Functions f_{14}-f_{23}

Numerical Results

F	No. of	FEP		CEP		FEP−CEP
	Gen's	Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_{14}	100	1.22	0.56	1.66	1.19	$-2.21^{\dagger}$
f_{15}	4000	5.0×10^{-4}	3.2×10^{-4}	4.7×10^{-4}	3.0×10^{-4}	0.49
f_{16}	100	−1.03	4.9×10^{-7}	−1.03	4.9×10^{-7}	0.0
f_{17}	100	0.398	1.5×10^{-7}	0.398	1.5×10^{-7}	0.0
f_{18}	100	3.02	0.11	3.0	0	1.0
f_{19}	100	−3.86	1.4×10^{-5}	−3.86	1.4×10^{-2}	−1.0
f_{20}	200	−3.27	5.9×10^{-2}	−3.28	5.8×10^{-2}	0.45
f_{21}	100	−5.52	1.59	−6.86	2.67	$3.56^{\dagger}$
f_{22}	100	−5.52	2.12	−8.27	2.95	$5.44^{\dagger}$
f_{23}	100	−6.57	3.14	−9.10	2.92	$4.24^{\dagger}$

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

Observations:

- **The results are mixed!**
  - **FEP and CEP performed equally well on f_{16} and f_{17}. They are comparable on f_{15} and f_{18}-f_{20}.**
  - **CEP performed better on f_{21}-f_{23} (Shekel functions).**
- **Is it because the dimension was low ($n = 2, 4, 6$) so that CEP appeared to be better?**

Experiments on Unimodal Functions f_{14}-f_{23}

Evolutionary Curves for f_{21}

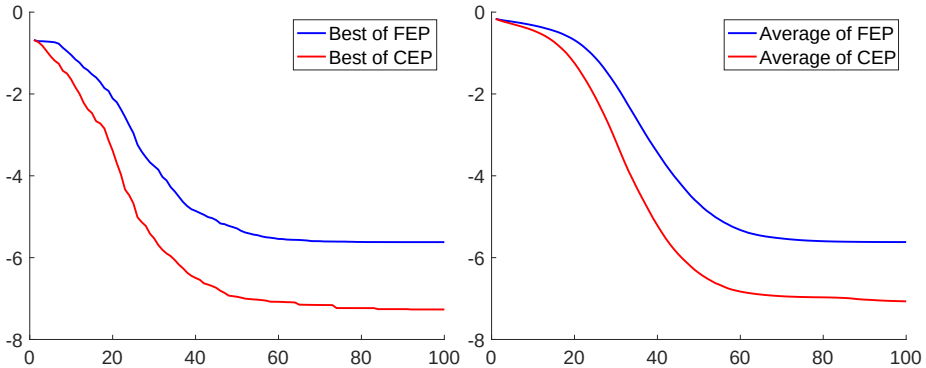


Figure 10: Evolutionary curves of CEP and FEP for f_{21}.

- **How can we gain a deeper understanding of these results?**
- **How can one line of code, i.e., a simple mutation operator, make such a big difference?**

Why Cauchy Mutation Performed Better (or Worse)

Given $\mathcal{N}(0,1)$ and $\mathcal{C}(1)$, the expected length of Gaussian and Cauchy jumps are:

$$E_{Gaussian}(x) = \int_0^{+\infty} x \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = \frac{1}{\sqrt{2\pi}} = 0.399$$

$$E_{Cauchy}(x) = \int_0^{+\infty} x \frac{1}{\pi(1+x^2)} dx = +\infty$$

It is obvious that Gaussian mutation is much localised than Cauchy mutation.

Why and When Large Jumps are Beneficial?

Take the $n = 1$ case and Gaussian mutation with $\mathcal{N}(0, \sigma^2)$ as an example, i.e.,

$$PDF_{\mathcal{N}(0,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}, \quad -\infty < x < +\infty,$$

the probability of generating a point in the vicinity of the global optimum x^* is given by

$$Prob_{\mathcal{N}(0,\sigma^2)}(|x - x^*| \leq \epsilon) = \int_{x^*-\epsilon}^{x^*+\epsilon} f_{\mathcal{N}(0,\sigma^2)}(x)dx \tag{5}$$

where

- $\epsilon > 0$ **and** σ **is often regarded as the step size of the Gaussian mutation.**

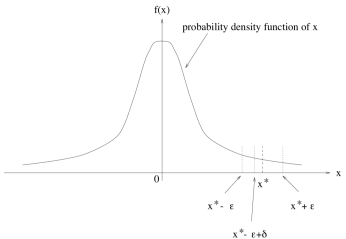


Figure 11: x^* is the global optimum and $\epsilon > 0$.

An Analytical Result

It can be shown that

$$\frac{\partial}{\partial \sigma} P_{\mathcal{N}(0,\sigma^2)}(|x - x^*| \leq \epsilon) > 0$$

when $|x^* - \epsilon + \delta| > \sigma$. That is, the larger σ is, the larger $P_{\mathcal{N}(0,\sigma^2)}(|x - x^*| \leq \epsilon)$ if $|x^* - \epsilon + \delta| > \sigma$.

On the other hand, if $|x^* - \epsilon + \delta| < \sigma$, then

$$\frac{\partial}{\partial \sigma} P_{\mathcal{N}(0,\sigma^2)}(|x - x^*| \leq \epsilon) < 0,$$

which indicates that $P_{\mathcal{N}(0,\sigma^2)}(|x - x^*| \leq \epsilon)$ decreases, as σ increases.

Remarks

1. **Cauchy mutation performs well when the global optimum is far away from the current search location. Its behaviour can be explained theoretically and empirically.**
2. **An optimal search step size can be derived if we know where the global optimum is. Unfortunately, such information is unavailable for real-world problems.**
3. **The performance of FEP can be improve by a set of more suitable parameters, instead of copying CEP's parameter setting.**
4. **The previous analysis considers only the probability of reaching the global optimum in one mutation. The global optimum can of course be reached in multiple mutations.**

Search Step Size and Search Bias

1. **The search step size of mutation can be defined by the mean step size that it takes in one operation.**
2. **In general, different search operators have different search step sizes, and thus appropriate for different problems as well as different evolutionary search stages for a single problem.**
3. **Search bias of an evolutionary search operator includes its step size and search directions. Search bias of a search operator determines how likely an offspring will be generated from a parent(s).**

Mixing Search Biases by Self-adaptation

- **Since the global optimum is *unknown* in real-world applications, it is impossible to know *a priori* what search biases we should use in EAs.**
- **One way to get around this problem is to use a variety of different biases and allow evolution to find out which one(s) are more promising than others.**
- **Rather than using either Gaussian or Cauchy mutations, we can mix them.**
- **That is, two candidate offspring can be generated from every parent, one by Gaussian mutation and one by Cauchy mutation. The fitter one will survive as the single child.**
- **This is the essence of IFEP (Improved Fast Evolutionary Programming) algorithm. This is one of the earliest examples of automatic operator selection in EAs.**

Example: Automatic Operator Selection

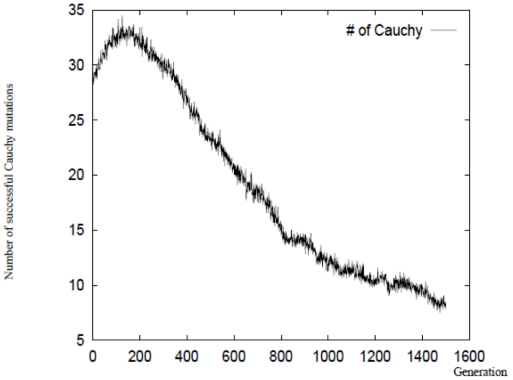


Figure 12: Number of successful Cauchy mutations in a population when an improve FEP based on mixing different mutation operators is applied to function f_{10}. The vertical axis indicates the number of successful Cauchy mutations in a population and the horizontal axis indicates the number of generations. The results have been averaged over 50 runs.

Other Mixing Methods

- **Mean mutation operator:** **Takes the average of the two mutations.**

$$x_i'(j) = x_i(j) + \eta_i(j)\left(0.5\mathcal{N}_j(0,1) + 0.5\mathcal{C}_j(1)\right)$$

where $\mathcal{N}_j(0,1)$ is a normally distributed number while $\mathcal{C}_j(1)$ follows Cauchy distribution with parameter 1.

- **Adaptive mutation operator:** **It's actually a self-adaptive method.**

$$x_i'(j) = x_i(j) + \eta_{1i}(j)\mathcal{N}_j(0,1) + \eta_{2i}(j)\mathcal{C}_j(1)$$

where both $\eta_{1i}(j)$ and $\eta_{2i}(j)$ are self-adaptive parameters.

A More General Self-Adaptive Method

- **The idea of mixing can be generalised to Lévy mutation.**
- **Lévy probability distribution can be tuned to generate any distribution between the Gaussian and Cauchy probability distributions.**
- **Hence we can use Lévy mutation with different parameters in EAs and let evolution to decide which one to use.**
- **This is the core idea behind Lévy Evolutionary Programming (LEP) [5].**

C. Y. Lee and X. Yao, "Evolutionary programming using the mutations based on the Lévy probability distribution," IEEE Transactions on Evolutionary Computation, 8(1):1-13, January 2004.

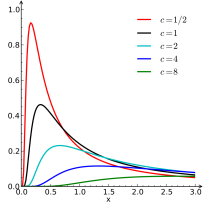


Figure 13: Probability density function of unshifted Lévy function (`https://commons.wikimedia.org/wiki/File:Levy0_distributionPDF.svg`).

- **Form of Lévy probability distribution:**

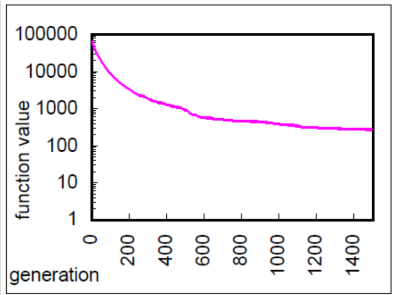
$$L_{\alpha,c}(x) = \frac{1}{\pi}\int_0^\infty e^{-cq^\alpha} cos(qx)dq$$

- $c > 0$ **is the scaling factor.**
- $\alpha \in (0, 2)$ **controls the shape of the distribution.**
- **Equivalent to Gaussian distribution when** $\alpha \to 2$**.**
- **Equivalent to Cauchy distribution when** $\alpha = 1$**.**

Self-adaptation is great.
It removes the need to tune parameters such as η, σ.

However, . . .

An Anomaly of Self-adaptation in EP



Generation	$(x_1(19), \eta_1(19))$	$f(x_1)$	$\frac{1}{\mu}\sum f(x_i)$
	:		
300	(-14.50, 4.52E−3)	812.85	846.52
	:		
600	(-14.50, 8.22E−6)	547.05	552.84
	:		
1000	(-14.50, 1.33E−8)	504.58	504.59
	:		
1500	(-14.50, 1.86E−12)	244.93	244.93

Figure 14: Left: The 30-dimensional sphere model stagnates early, from mean of 50 runs. Right: The 19-th component and the fitness of the best individual in a run.

- **Parasites, junk genes, etc.**
- **Getting Around the Anomaly: Setting a lower bound!**
  For example, set a fixed lower bound, e.g., 10^{-3}.

Use Mutation Step Size to Adjust Lower Bounds

Use the median of the mutation step size from all accepted (successful) offspring as the new lower bound for the next generation. Let $\delta_i(j) = \eta_i'(j)\mathcal{N}_j(0,1)$. We first calculate the average mutation step size from all accepted (successful) offspring:

$$\bar{\delta}(j) = \frac{1}{m}\sum_{k=1}^{m}\delta_k(j), j = 1, \ldots, n,$$

where m is the number of the accepted offspring.
Then, the lower bound of η for the next generation is

$$\eta_-^{t+1} = median\ \bar{\delta}(j), j = 1, \ldots, n.$$

K. H. Liang, X. Yao and C. S. Newton, "Adapting self-adaptive parameters in evolutionary algorithms," Applied Intelligence, 15(3):171-180, November/December 2001.

Getting Around the Anomaly - Recombination

Intermediate recombination helps because it averages out extremely small step sizes.

Remarks on Mutation-based EAs

- **Search step size is a crucial factor in determining EA's performance.**
- **Different operators, and EAs in general, have different search biases.**
- **Mixing different operators adaptively can lead to better performance for many (but not all) problems.**
- **However, cares must be taken as self-adaptation may not work as claimed if some additional techniques are not used.**

Outline of This Lecture

Representation is Important

Search and representation are fundamental to evolutionary search. They go hand-in-hand.

- **Representation: mapping of genotypes to phenotypes.**
- **Binary strings have often been used to represent individuals, e.g., integers and real numbers. However, they may not be good representations, because binary encoding of an integer or real number can introduce so-called Hamming cliffs.**
- **Gray coding can help, but does not solve the problem entirely. A better way is to use integer representation, real-valued or float-point representation.**
- ***There are also permutation representation and tree representation. Will introduce next week.***

Binary Code vs. Gray Code

Integer	Binary code	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Adaptive Representation: Cartesian vs Polar

- **Although we have been using the Cartesian coordinates in all our examples so far, there are cases where a different representation would be more appropriate, e.g., polar coordinates.**
- **The idea of self-adaptation can also be used in representations, where the most suitable representation will be evolved rather than fixed in advance.**
- **For example, Cartesian and polar representations can be mixed adaptively in an EAs so that evolution can select which representation is the best in the current stage of evolutionary search.**

T. Schnier and X. Yao, "Using Multiple Representations in Evolutionary Algorithms," Proceedings of the 2000 Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, July 2000. pp.479-486.

Outline of This Lecture

EA Families

- **Genetic Algorithms (GAs) $\leftarrow$ We have seen previously**
- **Evolutionary Programming (EP) $\leftarrow$ We have seen previously**
- **Genetic Programming (GP)**
- **Differential Evolution (DE)**
- **Particle Swarm Optimisation (PSO)**
- **Evolution Strategies (ES)**
- **...**

Differential Evolution (DE)

- **Gradient-free method proposed by Storn and Price [8] in 1997.**
- **Good for**
  - **Continuous search space**
  - **Ill-conditioning functions (small changes in $\mathrm{x} \rightarrow$ large changes in $f(\mathrm{x})$)**
- **DE variants based on differential mutators:**
  - **DE/rand/1 [7] :** $\mathbf{x}_i' = \mathbf{x}_a + F(\mathbf{x}_b - \mathbf{x}_c)$
  - **DE/rand/2 [7] :** $\mathbf{x}_i' = \mathbf{x}_a + F(\mathbf{x}_b - \mathbf{x}_c) + F(\mathbf{x}_d - \mathbf{x}_e)$
  - **DE/best/1 [7] :** $\mathbf{x}_i' = \mathbf{x}_{best} + F(\mathbf{x}_b - \mathbf{x}_c)$
  - **DE/best/2 [7] :** $\mathbf{x}_i' = \mathbf{x}_{best} + F(\mathbf{x}_b - \mathbf{x}_c) + F(\mathbf{x}_d - \mathbf{x}_e)$

  where $\mathbf{x}_{best}$ is the best point in the current population, $\mathbf{x}_a$, $\mathbf{x}_b$, $\mathbf{x}_c$, $\mathbf{x}_d$ and $\mathbf{x}_e$ are distinct points randomly chosen in the current population.

Pseudo Code of DE/rand/2

Algorithm 1 DE/rand/2. For $j \in \{1, \ldots, n\}$, $x(j)$ denotes the j^{th} coordinate of a vector $\mathbf{x} \in \mathbb{R}^n$.

Input $F > 0$: Differential weight, $p_c \in [0, 1]$: Crossover probability, μ: Population size
Initialise $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$ uniformly in the bounded search space
while stopping criteria not met **do**
  for $i \in \{1, \ldots, \mu\}$ **do**
    Randomly draw a, b, c, d and e distinct in $\{1, \ldots, i-1, i+1, \ldots, \mu\}$
    Define $\mathbf{x}'_i = \mathbf{x}_a + F(\mathbf{x}_b - \mathbf{x}_c) + F(\mathbf{x}_d - \mathbf{x}_e)$
    Randomly draw $R \in \{1, \ldots, n\}$
    for $j \in \{1, \ldots, n\}$, **do**
      if $rand \leq \mathbf{x}_c$ or $j == R$ **then**
        $x''_i(j) = x'_i(j)$
      else
        $x''_i(j) = x_i(j)$
      end if
    end for
    $\mathbf{x}_i = best(\mathbf{x}_i, \mathbf{x}''_i)$ (keep $\mathbf{x}_i$ in case of tie)
  end for
end while

Particle Swarm Optimisation (PSO)

- **Proposed by Kennedy and Eberhart in 1995 [4]**
- **No crossover**
- **Mutation defined through a vector addition**
- **Consider an individual as a point in the solution space with a position $\mathbf{x}$ and a velocity $\mathbf{v}$ and use the latter to determine a new position (and a new velocity)**
- **Differences compared to DE: every candidate solution $\mathbf{x} \in \mathbb{R}^n$ carries its own perturbation vector $\mathbf{v} \in \mathbb{R}^n$**

Algorithm 2 PSO. For $j \in \{1, \ldots, n\}$, $\mathbf{x}(j)$ denotes the j^{th} coordinate of a vector $\mathbf{x} \in \mathbb{R}^n$.

Input $\phi_1, \phi_2 > 0$: Differential weight, μ: Population size
Initialise $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$ uniformly in the bounded search space (LB, UB)
Initialise the particles' velocities uniformly in the range $(-|UB - LB|, |UB - LB|)$
for $i \in \{1, \ldots, \mu\}$ **do**
  $\mathbf{p}_i \leftarrow$ the particle $\mathbf{x}_i$
end for
$\mathbf{g} \leftarrow$ the global best position among $\mathbf{p}_1, \ldots, \mathbf{p}_\mu$
while stopping criteria not met **do**
  for $i \in \{1, \ldots, \mu\}$ **do**
    for $j \in \{1, \ldots, n\}$ **do**
      Pick random numbers $r_p, r_q \sim \mathbb{U}(0, 1)$
      $v_i'(j) = v_i(j) + \phi_1 \cdot r_p(p_i(j) - x_i(j)) + \phi_2 \cdot r_q(g(j) - x_i(j))$ and $x_i'(j) = x_i(j) + v_i'(j)$
    end for
  end for
  for $i \in \{1, \ldots, \mu\}$ **do**
    if $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **then**
      $\mathbf{p}_i = \mathbf{x}_i$ ▶ Update the particle's best position
    end if
    if $f(\mathbf{p}_i) < f(\mathbf{g})$ **then**
      $\mathbf{g} = \mathbf{p}_i$ ▶ Update the global best position
    end if
  end for
end while

Evolution Strategies (ES)

Almost the same as EP,

$$\mathbf{x}' = \mathbf{x} + \sigma\mathcal{N}(0, \mathbf{I})$$

as perturbation(s) of parent $\mathbf{x} \in \mathbb{R}^n$, where

- **σ is a step-size,**
- **$\mathcal{N}(0, \mathbf{I})$ denotes a n-dimensional normal distribution (isotropic).**

Self-Adaptive Evolution Strategy (SA-ES)

Algorithm 3 SA-ES. $mod(a, b)$ denotes a modulo b.

1: Parameters: $\lambda \geq \mu \in \mathbb{N}^+$, a dimension $n \in \mathbb{N}^+$
2: Input: initial population $\mathbf{x}_i \in \mathbb{R}^n$ and initial step-size $\sigma_i = 1$, $i \in \{1, \ldots, \mu\}$
3: **while** stopping criteria not met **do**
4: Generate λ individuals $\mathbf{x}'_i$, $i \in \{1, \ldots, \lambda\}$, independently using ▶ Reproduction

$$\begin{aligned} \sigma_i &= \sigma_{mod(i-1,\mu)+1} \times \exp\left(\frac{\mathcal{N}(0, \mathbf{I})}{2n}\right) \\ \mathbf{x}'_i &= \mathbf{x}_{mod(i-1,\mu)+1} + \sigma_i \mathcal{N}(0, \mathbf{I}) \end{aligned}$$

5: Evaluate each of the individuals once ▶ Evaluation
6: Define $k_1, \ldots, k_\lambda$ so that ▶ Rank

$$f(\mathbf{x}'_{k_1}) \leq f(\mathbf{x}'_{k_2}) \cdots \leq f(\mathbf{x}'_{k_\lambda})$$

7: $\mathbf{x}_i = \mathbf{x}'_{k_i}$, $i \in \{1, \ldots, \mu\}$ ▶ Replacement of population
8: $\sigma_i = \sigma_{k_i}$, $i \in \{1, \ldots, \mu\}$ ▶ Replacement of step-size
9: **end while**

Black-box Continuous Optimisation

Objective function $f : \mathbb{R}^n \mapsto \mathbb{R} \hookrightarrow$ Continuous optimization

$$\mathbf{x} \longrightarrow \boxed{\text{Black-box}} \longrightarrow f(\mathbf{x})$$

Goal: Find $\mathbf{x}^* = \text{argmin}_{x \in \mathbb{R}^n} f(\mathbf{x})$.

- **Do not use any internal property of f**
- **Randomised search: sample search points/solutions $\mathbf{x}_1, \mathbf{x}_2, \ldots$ respecting to a given distribution $\mathcal{P}$**
  - **In EAs, $\mathcal{P}$ is implicitly defined and updated via the search operators and selection schemes**

Outline of This Lecture

Summary

- **Search step size is a crucial factor in determining EA's performance.**
- **Mixing different operators adaptively can lead to better performance for many problems.**
- **Representation is important. Representations can be adaptive during a run.**
- **There are many popular EAs variants, select the variant according to your problem!**

Essential Reading for This Lecture I

[1] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. CRC Press, 1997.

[2] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Vol. 53. Springer, 2003.

[3] Nikolaus Hansen and Andreas Ostermeier. "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation". In: *Proceedings of IEEE international conference on evolutionary computation*. IEEE. 1996, pp. 312–317.

[4] J. Kennedy and R. Eberhart. "Particle Swarm Optimization". In: *Proceedings of IEEE International Conference on Neural Networks*. 1995, 1942–1948.

[5] C. Y. Lee and X. Yao. "Evolutionary programming using the mutations based on the Lévy probability distribution". In: *IEEE Transactions on Evolutionary computation* 8.1 (2004), pp. 1–13.

Essential Reading for This Lecture II

[6] Ko-Hsin Liang, Xin Yao, and Charles Newton. "Combining landscape approximation and local search in global optimization". In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*. Vol. 2. IEEE. 1999, pp. 1514–1520.

[7] Rainer Storn. "On the usage of differential evolution for function optimization". In: *Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American*. IEEE. 1996, pp. 519–523.

[8] Rainer Storn and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359.

[9] Xin Yao, Yong Liu, and Guangming Lin. "Evolutionary programming made faster". In: *IEEE Transactions on Evolutionary computation* 3.2 (1999), pp. 82–102.

Essential Reading for Next Lecture

1. **A. E. Eiben & J. E. Smith (2003). "Introduction to evolutionary computing," (Vol. 53, p. 18). Berlin: springer. (Second Edition)**
2. **X. Yao (1993b), "An empirical study of genetic operators in genetic algorithms," Microprocessing and Microprogramming, 38(1-5):707–714.**
3. **Liang, K. H., Yao, X., Newton, C., & Hoffman, D. (2002). "A new evolutionary approach to cutting stock problems with and without contiguity," Computers & Operations Research, 29(12), 1641-1659.**