Lecture 3
Real-valued Representations and Operators

Dr. Philipp Rohlfshagen
p.rohlfshagen@cs.bham.ac.uk

based on slides by X. Yao

October 9, 2009

Introduction

Representation of a problem

- Each individual corresponds to a solution $\boldsymbol{x} = x_1 x_2 \dots x_n$
- We can modify the solution by means of crossover and mutation

Last lecture: Discrete representations

- Each gene has a value taken from a finite set
- E.g., $x_i \in \{0, 1\}$ or $x_i \in \{4, 5, 6, 7, 8\}$

This lecture: Real-valued (continuous) representations

- Each gene has a value taken from a continuous interval
- E.g., $x_i \in [-5, 5]$ or $x_i \in [0, 1]$

Problems with Discrete Representations

Q: Why do we need another type of representation?

A binary encoding can represent any integer

- n bits encode 2^n integers from 0 to $2^n - 1$
- E.g., $0000 = 0,\ 0001 = 1,\ 0010 = 2,\ 0011 = 3,\ \ldots$

Hamming Cliffs

- Locality may not be preserved
- $0111 = 7$ BUT $1000 = 8$ (think mutation operator)

Gray Coding partly overcomes this issue

- Converts the mapping from binary to integer
- Can reach any adjacent integer by single bit-flip

Gray Coding

Integer	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Gray Coding

Integer	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Gray Coding

Integer	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary-based Real-valued Representations

Can also represent most floating point numbers as binary

- Discretising the continuous domain

How? Given domain $[-2, 2]$ and precision of 6 decimal places

- Divide domain $[-2, 2]$ into $4 \cdot 1000000$ intervals
- We need 22 bits ($4 \times 1000000 < 2^{22}$)
- Convert binary string to integer x', then convert to real

$$x = -2 + x' \cdot \frac{4}{2^{22} - 1}$$

- But same problem as before

Using a real-valued representation overcomes these issues

- Real-valued representation may be a more natural choice

Discrete Recombination

Does not change actual (gene) values

- Very similar to the crossover operators on binary strings

Multi-point Recombination (1-point, 2-point, k-point crossover)

- Similar to that for the binary representation

Global Discrete Recombination

- Similar to uniform crossover for the binary representation

x_1: 0.21 1.87 3.66 | 1.11 2.25 x'_1: 0.21 1.87 3.66 | 2.56 0.11

x_2: 2.32 0.77 2.99 | 2.56 0.11 x'_2: 2.32 0.77 2.99 | 1.11 2.25

Discrete Recombination

Does not change actual (gene) values

- Very similar to the crossover operators on binary strings

Multi-point Recombination (1-point, 2-point, k-point crossover)

- Similar to that for the binary representation

Global Discrete Recombination

- Similar to uniform crossover for the binary representation

$\boldsymbol{x}_1$: 0.21 1.87 3.66 | 1.11 2.25 $\boldsymbol{x}'_1$: 0.21 1.87 3.66 | 2.56 0.11

$\boldsymbol{x}_2$: 2.32 0.77 2.99 | 2.56 0.11 $\boldsymbol{x}'_2$: 2.32 0.77 2.99 | 1.11 2.25

Intermediate Recombination

We can actually modify the genetic values

- In the discrete case, resulting value might not be valid
- E.g., average in binary ($x_{1i} = x_{2i} = 1$): $x'_i = (x_{1i} + x_{2i})/2 = 0.5 \notin \{0, 1\}$

For 2 parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and $i = 1, 2, \ldots, n$:

$$x'_i = \alpha x_{1i} + (1 - \alpha) x_{2i} \quad \text{where} \quad \alpha \in [0, 1]$$

Given $\alpha = 0.5$:

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$\boldsymbol{x}'$: 1.27 1.32 3.33 1.84 1.18

Intermediate Recombination

We can actually modify the genetic values

- In the discrete case, resulting value might not be valid
- E.g., average in binary ($x_{1i} = x_{2i} = 1$): $x'_i = (x_{1i} + x_{2i})/2 = 0.5 \notin \{0, 1\}$

For 2 parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and $i = 1, 2, \ldots, n$:

$$x'_i = \alpha x_{1i} + (1 - \alpha) x_{2i} \quad \text{where} \quad \alpha \in [0, 1]$$

Given $\alpha = 0.5$:

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$\boldsymbol{x}'$: 1.27 1.32 3.33 1.84 1.18

Arithmetic & Heuristic Recombination

Arithmetic crossover (for p parents):

$$x_i' = \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_3 x_{3i} + \ldots \quad \text{where} \quad \sum_{i=1}^{p} \alpha_i = 1$$

- Generalised (simple) intermediate crossover

Heuristic crossover (where $\mathbf{x_1}$ is no worse than $\mathbf{x_2}$):

$$\mathbf{x}' = u(\mathbf{x_1} - \mathbf{x_2}) + \mathbf{x_1} \quad \text{where} \quad u = rand([0,1])$$

- Partially reflected point

Arithmetic & Heuristic Recombination

Arithmetic crossover (for p parents):

$$x_i' = \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_3 x_{3i} + \ldots \quad \text{where} \quad \sum_{i=1}^{p} \alpha_i = 1$$

- Generalised (simple) intermediate crossover

Heuristic crossover (where $\mathbf{x_1}$ is no worse than $\mathbf{x_2}$):

$$\mathbf{x}' = u(\mathbf{x_1} - \mathbf{x_2}) + \mathbf{x_1} \quad \text{where} \quad u = rand([0,1])$$

- Partially reflected point

Heuristic Recombination

Heuristic crossover example: Assume $f(\boldsymbol{x}_1) \geq f(\boldsymbol{x}_2)$ and $x_i \in [0, 4]$

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$$0.13 \cdot (0.21 - 2.32) + 0.21 \,,\; 0.47 \cdot (1.87 - 0.77) + 1.87 \quad \ldots$$

$$\boldsymbol{x}': -0.06 \;\; 2.39 \quad \ldots$$

What is happening here?

What if resulting values are out of bound (e.g., -0.06)?

Heuristic Recombination

Heuristic crossover example: Assume $f(\boldsymbol{x}_1) \geq f(\boldsymbol{x}_2)$ and $x_i \in [0, 4]$

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$$0.13 \cdot (0.21 - 2.32) + 0.21 \;,\; 0.47 \cdot (1.87 - 0.77) + 1.87 \quad \ldots$$

$$\boldsymbol{x}': -0.06 \;\; 2.39 \quad \ldots$$

What is happening here?

What if resulting values are out of bound (e.g., -0.06)?

Simplex Recombination I

Randomly select a group (> 2) of parents from the population

Find the best $\boldsymbol{x}_{max}$ and worst $\boldsymbol{x}_{min}$ in the group

- The offspring $\boldsymbol{x}'$ will eventually replace $\boldsymbol{x}_{min}$

Compute the centroid, $\boldsymbol{x}_c$, of the group ignoring $\boldsymbol{x}_{min}$

$$\boldsymbol{x}_c = \frac{1}{n}\sum_{i=1}^{n-1} x_i$$

Generate a reflected point using $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_c$:

$$\boldsymbol{x}_r = \boldsymbol{x}_c + (\boldsymbol{x}_c - \boldsymbol{x}_{min})$$

Simplex Recombination II

If $f(\boldsymbol{x}_r) \geq f(\boldsymbol{x}_{max})$, *then* create expanded point:

$$\boldsymbol{x}_e = \boldsymbol{x}_r + (\boldsymbol{x}_r - \boldsymbol{x}_c)$$

If $f(\boldsymbol{x}_e) \geq f(\boldsymbol{x}_r)$, *then* $\boldsymbol{x}' \leftarrow \boldsymbol{x}_e$, else $\boldsymbol{x}' \leftarrow \boldsymbol{x}_r$

If $f(\boldsymbol{x}_r) \geq f(\boldsymbol{x}_{min})$, *then* $\boldsymbol{x}' \leftarrow \boldsymbol{x}_r$

If $f(\boldsymbol{x}_r) \leq f(\boldsymbol{x}_{min})$, *then* compute contracted point $\boldsymbol{x}_q$

$$\boldsymbol{x}_q = (\boldsymbol{x}_{min} + \boldsymbol{x}_c)/2$$

If $f(\boldsymbol{x}_q) \geq f(\boldsymbol{x}_{min})$, *then* $\boldsymbol{x}' \leftarrow \boldsymbol{x}_q$

Else, offspring is *contracted point* towards $\boldsymbol{x}_{max}$

$$\boldsymbol{x}' = (\boldsymbol{x}_{min} + \boldsymbol{x}_{max})/2$$

Geometric & Quadratic Recombination

Geometric recombination

- For 2 parents:

$$\mathbf{x}' = [(x_{11}x_{21})^{\frac{1}{2}}, \ldots, (x_{1n}x_{2n})^{\frac{1}{2}}]$$

- Can be generalised to k parents:

$$\mathbf{x}' = [(x_{11}^{\alpha_1}x_{21}^{\alpha_2}x_{k1}^{\alpha_k}), \ldots, (x_{1n}^{\alpha_1}x_{2n}^{\alpha_2}x_{kn}^{\alpha_k})]$$

where $\sum_{i=1}^{k} \alpha_k = 1$

Quadratic recombination

- From 3 parents, generate offspring using quadratic interpolation:

$$x_j' = \frac{1}{2} \cdot \frac{(x_{2j}^2 - x_{3j}^2)f(x_1) + (x_{3j}^2 - x_{1j}^2)f(x_2) + (x_{1j}^2 - x_{2j}^2)f(x_3)}{(x_{2j} - x_{3j})f(x_1) + (x_{3j} - x_{1j})f(x_2) + (x_{1j} - x_{2j})f(x_3)}$$

Geometric & Quadratic Recombination

Geometric recombination

- For 2 parents:

$$\mathbf{x}' = [(x_{11}x_{21})^{\frac{1}{2}}, \ldots, (x_{1n}x_{2n})^{\frac{1}{2}}]$$

- Can be generalised to k parents:

$$\mathbf{x}' = [(x_{11}^{\alpha_1} x_{21}^{\alpha_2} x_{k1}^{\alpha_k}), \ldots, (x_{1n}^{\alpha_1} x_{2n}^{\alpha_2} x_{kn}^{\alpha_k})]$$

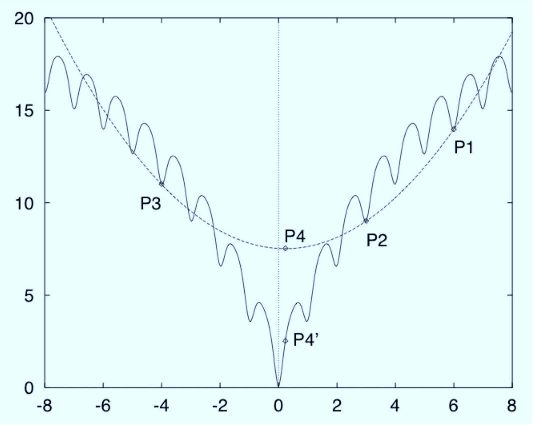
where $\sum_{i=1}^{k} \alpha_k = 1$

Quadratic recombination

- From 3 parents, generate offspring using quadratic interpolation:

$$x_j' = \frac{1}{2} \cdot \frac{(x_{2j}^2 - x_{3j}^2)f(x_1) + (x_{3j}^2 - x_{1j}^2)f(x_2) + (x_{1j}^2 - x_{2j}^2)f(x_3)}{(x_{2j} - x_{3j})f(x_1) + (x_{3j} - x_{1j})f(x_2) + (x_{1j} - x_{2j})f(x_3)}$$

What Does Quadratic Recombination Mean



A Hybrid EA With Local Search

1. Initialize μ individuals at random
2. Perform local search on each individual
3. REPEAT
   - 3.1 Generate P_1, P_2, P_3 by global discrete recombination
   - 3.2 Perform quadratic approximation using P_1, P_2, P_3 to produce P_4
   - 3.3 Perform local search from P_4
   - 3.4 Place P_1, P_2, P_3, P_4 into population
   - 3.5 Perform a $(\mu + 4)$ truncation selection
4. UNTIL termination criteria are met

Local Search With Random Memorising I

Local search in continuous domain

- Usually differs from local search in the discrete domain
- Use techniques like *Simplex method*, *quasi Newton procedures*, etc.
- Even with restarts, not good for finding global optimum of rugged functions

Combine local search with global search method (e.g., EA)

- Local search is expensive, especially if same local optimum is found multiple times

Use random memorising to improve efficiency:

- Store best solutions in a sequential memory (up to certain depth)
- Retrieve a random one when a new best solution is found
- Search along the direction of old $\longrightarrow$ new

Local Search With Random Memorising II

Search along the direction of old → new

First, compute direction

$$s = \frac{x_{new} - x_{old}}{\|x_{new} - x_{old}\|}$$

Second, perform local search with increasing step size

$$\boldsymbol{x}_{c_b} = \boldsymbol{x}_{c_b - 1} + s \cdot b_a \cdot d_g$$

- where s is the direction
- b_a the search step multiplier (which increases over time)
- d_g is the original, global step size

New point added to memory as used for EA to continue search

Experimental Studies

18 multimodal benchmark functions

- $f_8 - f_{25}$ (a set of functions)
- $n \in \{2, 4, 6, 30\}$, domains differ for each problem

Population size is $N = 30$

Maximum function evaluation $F = 500000$

50 independent runs for each function

independent: Different random seeds, starting from different initial populations

Results on f_8–f_{25}

f	F	μ	σ	%
8	199244	-11834.65	298.78	1/50
9	306280	2.67	1.58	5/50
10	370686	2.08e-12	2.11e-12	50/50
11	173592	1.87e-10	1.31e-9	50/50
12	476245	5.84e-9	2.16e-8	50/50
13	504212	1.19e-2	3.01e-2	33/50
14	3052	1.04	0.20	48/50
15	111748	3.0749e-4	2.45e-10	50/50
16	2817	-1.031628	2.70e-8	50/50
17	5496	0.3979	8.26e-9	50/50
18	4676	3	0	50/50
19	6852	-3.86	0	50/50
20	17504	-3.32	2.35e-7	50/50
21	13790	-10.1532	0	50/50
22	13354	-10.4029	2.16e-7	50/50
23	14312	-10.5364	4.58e-7	50/50
24	10754	3.89e-4	1.90e-3	48/50
25	15614	1.07e-4	7.51e-4	50/50

Results on f_8 and f_9 with $N \in \{50, 60\}$

f	F	μ	σ	%
8	199244	-11834.65	298.78	1/50
9	306280	2.67	1.58	5/50

f	N	F	μ	σ	%
8	50	327434	-12296.68	119.27	1/50
8	60	391634	-12358.64	166.33	12/50
9	50	508021	2.98e-1	4.97e-1	36/50
9	60	610163	2.19e-1	4.12e-1	39/50

Some Observations

Different problems require different operators and selection schemes

- There is no universally best one
- Important to understand what to use when!

Real valued representations may be more appropriate than binary

- A more natural choice for function optimisation
- Some guidelines exists for choosing good representations (neighbourhood, etc.)

Many search operators are heuristics-based

- Domain knowledge can often be incorporated into operators and representation

Global Optimisation by Mutation-Based EAs I

1. Generate initial population of μ individuals, and set $k = 1$
   - Each individual is a real-valued vector ($x_i \in [l, u]$)
2. Evaluate the fitness of each individual
3. Each individual creates a single offspring: For $j = 1, \cdots, n$

$$x_{ij}' = x_{ij} + N_j(0, 1)$$

4. Calculate the fitness of each offspring

Global Optimisation by Mutation-Based EAs II

5. For each individual, q opponents are chosen from parents and offspring
   - If the individual's fitness is no less than the opponent's, it receives a *win*
6. Select the μ best individuals (from 2μ) that have the most wins
   - They constitute the next generation.
7. Stop if the stopping criterion is satisfied;
   - Otherwise, $k = k + 1$ and go to Step 3

Analysis of mutation operator

$N(0,1)$ denotes a normally distributed random number

- The mean is $\mu = 0$ and the standard deviation is $\sigma = 1$ (i.e., $N(\mu, \sigma)$)

$N_j(0,1)$ means a newly sampled value for each index j

μ determines the search step size of the mutation.

- It is a crucial parameter

Unfortunately, the optimal search step size is problem-dependent

- Even for a single function, different search step sizes may be optimal
- Self-adaptation can be used to get around this problem partially

Function Optimisation by Classical EP (CEP)

Each individual (x_i, η_i) creates a single offspring $({x_i}', {\eta_i}')$

For $j = 1, \cdots, n$

$$\begin{aligned} {\eta_{ij}}' &= \eta_{ij} \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (1) \\ {x_{ij}}' &= x_{ij} + \eta_{ij} N_j(0,1) \quad (2) \end{aligned}$$

The factors τ' and τ are commonly set to:

- $\left(\sqrt{2n}\right)^{-1}$ (global step-size)
- $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ (local step-size)

Fast EP

The idea comes from fast simulated annealing.

Replace Gaussian distribution with a Cauchy one in Eq.(2)

$$x_{ij}' = x_{ij} + \eta_{ij} C_j(1) \tag{3}$$

$C_j(1)$ is an Cauchy random number variable with the scale parameter $t = 1$

- It is generated anew for each value of j

Everything else, including Eq.(1), is kept unchanged

Cauchy Distribution

Its density function is

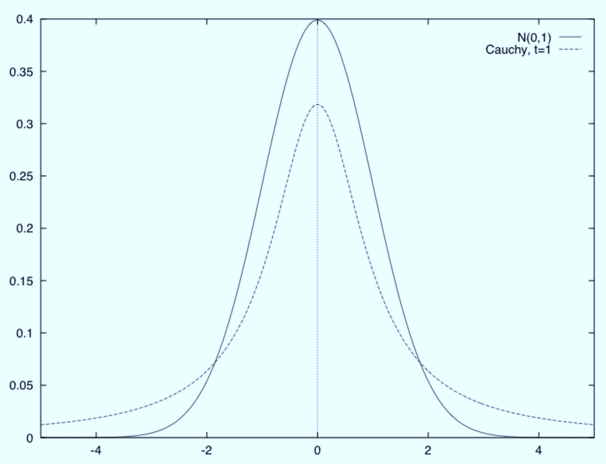
$$f_t(x) = \frac{1}{\pi} \frac{t}{t^2 + x^2}, \quad -\infty < x < \infty$$

where $t > 0$ is a scale parameter (step size)

The corresponding cumulative distribution function is

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

Gaussian and Cauchy Density Functions



Test Functions & Experimental Setup

23 different functions were used in the computational study

- Some have a relatively high dimension and some have many local optima

Population size is $N = 100$

Tournament size 10 for selection

All experiments were run 50 times, i.e., 50 trials

Each trials lasts F_g generations

Initial populations were the same for CEP and FEP

Experiments on Multimodal Functions f_8–f_{23}

f	F_g	FEP		CEP		FEP−CEP
		μ_{best}	σ	μ_{best}	σ	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	$-2.76^{\dagger}$
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.

Why Cauchy Mutation Performed Better (Or Not)

Given $G(0,1)$ and $C(1)$, the expected length of jumps are:

Gaussian:

$$E_{Gaussian}(x) = \int_0^{+\infty} x \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = \frac{1}{\sqrt{2\pi}} \approx 0.4$$

Cauchy:

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

Gaussian mutation is much more localised than Cauchy mutation

Cauchy mutation generates a longer jump than Gaussian with prob. 0.68

Why and When Large Jumps Are Beneficial

Take the Gaussian mutation with $G(0, \sigma^2)$ distribution as an example:

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

The probability of generating a point in the neighbourhood of x^* is given by

$$P_{G(0,\sigma^2)}(|x - x^*| \leq \epsilon) = \int_{x^*-\epsilon}^{x^*+\epsilon} f_{G(0,\sigma^2)}(x)dx$$

- $\epsilon > 0$ is the neighbourhood size
- σ is the step size of the Gaussian mutation

An Analytical Result

It can be shown that for some $0 < \delta < 2\epsilon$

$$\frac{\partial}{\partial\sigma} P_{G(0,\sigma^2)}(|x - x^*| \leq \epsilon) > 0 \quad \text{when} \quad |x^* - \epsilon + \delta| > \sigma$$

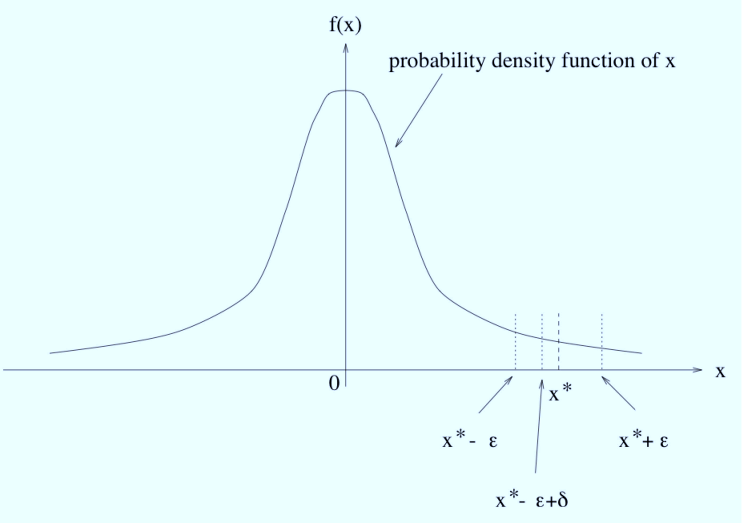
- The larger σ is, the larger $P_{G(0,\sigma^2)}(|x - x^*| \leq \epsilon)$

On the other hand

$$\frac{\partial}{\partial\sigma} P_{G(0,\sigma^2)}(|x - x^*| \leq \epsilon) < 0 \quad \text{when} \quad |x^* - \epsilon + \delta| < \sigma$$

- The larger σ is, the smaller $P_{G(0,\sigma^2)}(|x - x^*| \leq \epsilon)$
- In fact, the probability decreases exponentially

An Analytical Result



Summary on Mutation

Cauchy mutation performs well when distance to global optimum is far away

- Its behaviour can be explained theoretically and empirically

Can derive optimal search step size if global optimum is known

- Unfortunately, such information is unavailable for real-world problems

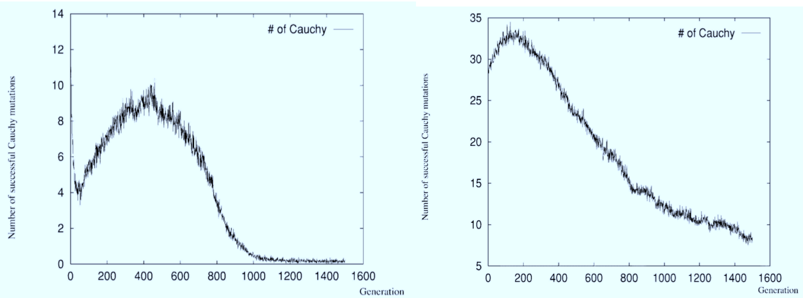
The performance of FEP can be improved by a set of more suitable parameters

- Instead of copying CEP's parameter setting (as done here for comparison)

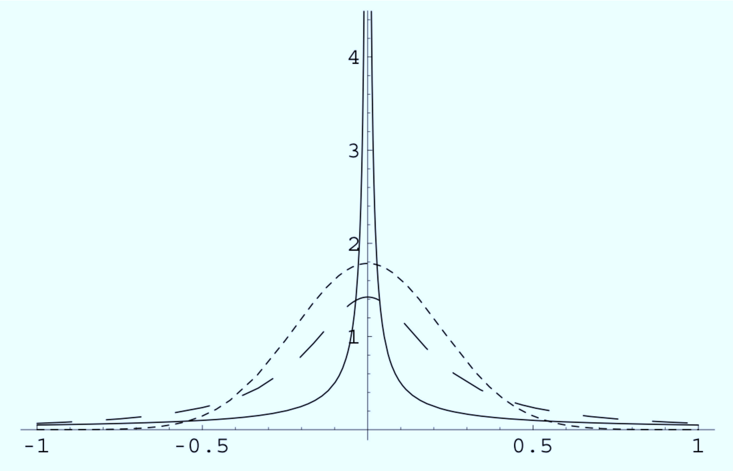
Many further possibilities

- Use both Gaussian and Cauchy (generate 2 offspring, keep the better)
- There are other distributions that may be used (see later slide)

Improved Fast Evolutionary Programming f_1 and f_{10}



Continuous Gray Coding Distribution



J. E. Rowe & D. Hidovic “An evolution strategy using a continuous version of the Gray-code neighbourhood distribution”

References (Essential Reading!)

1. H.-M. Voigt and J. Lange "Local evolutionary search enhancement by random memorizing" Proc. of the 1998 IEEE Int. Conf. on Evolutionary Computation. IEEE Press, Piscataway, NJ, USA, pp.547-552, 1998.

2. K.-H. Liang, X. Yao and C. S. Newton "Combining landscape approximation and local search in global optimization" Proc. of the 1999 Congress on Evolutionary Computation. Vol. 2, IEEE Press, Piscataway, NJ, USA, pp.1514-1520, July 1999.

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

4. **M-level**: Rowe, J.E. and Hidovic, D. (2004) An Evolution Strategy using a continuous version of the Gray-code neighbourhood distribution. Proceedings of GECCO 2004, Part 1 (Lecture Notes in Computer Science, vol. 3102), K. Deb et al. (eds). Springer-Verlag, pages 725-736.

Assessment

First of two course work assessments

- Only applicable to M-level students (Introduction to Evolutionary Computation)
- Worth 10% of your final mark
- Everyone else is encouraged to give it a go (we can give feedback)

All details are available online

- `www.cs.bham.ac.uk/~pkl/teaching/2009/ec`
- Submission is done online (as detailed on the web)