A Gentle Introduction to Evolutionary Computation

Xin Yao (x.yao@cs.bham.ac.uk)

1. What Is Evolutionary Computation
2. Different Evolutionary Algorithms
3. Major Areas Within Evolutionary Computation
4. Summary

What Is Evolutionary Computation

1. It is the study of computational systems which use ideas and get inspirations from natural evolution.
2. One of the principles borrowed is *survival of the fittest*.
3. Evolutionary computation (EC) techniques can be used in optimisation, learning and design.
4. EC techniques do not require rich domain knowledge to use. However, domain knowledge can be incorporated into EC techniques.

A Simple Evolutionary Algorithm

1. Generate the initial **population** $P(0)$ at random, and set $i \leftarrow 0$;
2. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) **Select** parents from $P(i)$ based on their fitness in $P(i)$;
   (c) **Generate** offspring from the parents using *crossover* and *mutation* to form $P(i+1)$;
   (d) $i \leftarrow i+1$;
3. UNTIL halting criteria are satisfied

EA as Population-Based Generate-and-Test

Generate: Mutate and/or recombine individuals in a population.

Test: Select the next generation from the parents and offsprings.

How Does the Simple EA Work

Let's use the simple EA to maximise the function $f(x) = x^2$ with x in the *integer* interval $[0, 31]$, i.e., $x = 0, 1, \cdots, 30, 31$.

The first step of EA applications is *encoding* (i.e., the representation of chromosomes). We adopt binary representation for integers. Five bits are used to represent integers up to 31. Assume that the population size is 4.

1. Generate initial population at random, e.g., 01101, 11000, 01000, 10011. These are *chromosomes* or *genotypes*.
2. Calculate fitness value for each individual.
   (a) Decode the individual into an integer (called *phenotypes*),

$$01101 \rightarrow 13, 11000 \rightarrow 24, 01000 \rightarrow 8, 10011 \rightarrow 19;$$

(b) Evaluate the fitness according to $f(x) = x^2$,

$$13 \to 169, 24 \to 576, 8 \to 64, 19 \to 361.$$

3. Select two individuals for crossover based on their fitness. If roulette-wheel selection is used, then

$$p_i = \frac{f_i}{\sum_j f_j}.$$

Two offspring are often produced and added to an intermediate population. Repeat this step until the intermediate population is filled. In our example,

$$p_1(13) = 169/1170 = 0.14 \quad p_2(24) = 576/1170 = 0.49$$
$$p_3(8) = 64/1170 = 0.06 \quad p_4(19) = 361.1170 = 0.31$$

Assume we have $crossover(01101, 11000)$ and $crossover(10011, 11000)$. We may obtain offspring 0110 0 and

1100 1 from $crossover(01101, 11000)$ by choosing a random crossover point at 4, and obtain 10 000 and 11 011 from $crossover(10011, 11000)$ by choosing a random crossover point at 2. Now the intermediate population is

$$01100, 11001, 10000, 11011$$

4. Apply mutation to individuals in the intermediate population with a *small* probability. A simple mutation is bit-flipping. For example, we may have the following new population $P(1)$ after random mutation:

$$0110\mathbf{1}, 11001, \mathbf{0}0000, 11011$$

5. Goto Step 2 if not stop.

Different Evolutionary Algorithms

There are several well-known EAs with different

- historical backgrounds,
- representations,
- variation operators, and
- selection schemes.

In fact, EAs refer to a whole family of algorithms, not a single algorithm.

Genetic Algorithms (GAs)

1. First formulated by Holland for adaptive search and by his students for optimisation from mid 1960s to mid 1970s.
2. Binary strings have been used extensively as individuals (chromosomes).
3. Simulate Darwinian evolution.
4. Search operators are only applied to the *genotypic* representation (chromosome) of individuals.
5. Emphasise the role of recombination (crossover). Mutation is only used as a background operator.
6. Often use roulette-wheel selection.

Evolutionary Programming (EP)

1. First proposed by Fogel *et al.* in mid 1960s for simulating intelligence.
2. Finite state machines (FSMs) were used to represent individuals, although real-valued vectors have always been used in numerical optimisation.
3. It is closer to Lamarckian evolution.
4. Search operators (mutations only) are applied to the *phenotypic* representation of individuals.
5. It does *not* use any recombination.
6. Usually use tournament selection.

Evolution Strategies (ES)

1. First proposed by Rechenberg and Schwefel in mid 1960s for numerical optimisation.
2. Real-valued vectors are used to represent individuals.
3. They are closer to Larmackian evolution.
4. They do have recombination.
5. They use self-adaptive mutations.

Genetic Programming (GP)

1. First used by de Garis to indicate the evolution of artificial neural networks, but used by Koza to indicate the application of GAs to the evolution of computer programs.
2. Trees (especially Lisp expression trees) are often used to represent individuals.
3. Both crossover and mutation are used.

Preferred Term: Evolutionary Algorithms

- EAs face the same fundamental issues as those classical AI faces, i.e., **representation**, and **search**.
- Although GAs, EP, ES, and GP are different, they are all different variants of population-based generate-and-test algorithms. They share more similarities than differences!
- A better and more general term to use is **evolutionary algorithms** (EAs).

Variation Operators and Selection Schemes

Crossover/Recombination: k-point crossover, uniform crossover, intermediate crossover, global discrete crossover, etc.

Mutation: bit-flipping, Gaussian mutation, Cauchy mutation, etc.

Selection: roulette wheel selection (fitness proportional selection), rank-based selection (linear and nonlinear), tournament selection, elitism, etc.

Replacement Strategy: generational, steady-state (continuous), etc.

Specialised Operators: multi-parent recombination, inversion, order-based crossover, etc.

Major Areas in Evolutionary Computation

1. Optimisation
2. Learning
3. Design
4. Theory

Evolutionary Optimisation

1. Numerical (global) optimisation.
2. Combinatorial optimisation (of NP-hard problems).
3. Mixed optimisation.
4. Constrained optimisation.
5. Multiobjective optimisation.
6. Optimisation in a dynamic environment (with a dynamic fitness function).

Evolutionary Learning

Evolutionary learning can be used in supervised, unsupervised and reinforcement learning.

1. Learning classifier systems (Rule-based systems).
2. Evolutionary artificial neural networks.
3. Evolutionary fuzzy logic systems.
4. Co-evolutionary learning.
5. Automatic modularisation of machine learning systems by speciation and niching.

Evolutionary Design

EC techniques are particularly good at exploring unconventional designs which are very difficult to obtain by hand.

1. Evolutionary design of artificial neural networks.
2. Evolutionary design of electronic circuits.
3. Evolvable hardware.
4. Evolutionary design of (building) architectures.

Summary

1. Evolutionary algorithms can be regarded as population-based generate-and-test algorithms.
2. Evolutionary computation techniques can be used in optimisation, learning and design.
3. Evolutionary computation techniques are flexible and robust.
4. Evolutionary computation techniques are definitely useful tools in your toolbox, but there are problems for which other techniques might be more suitable.

Global Optimisation

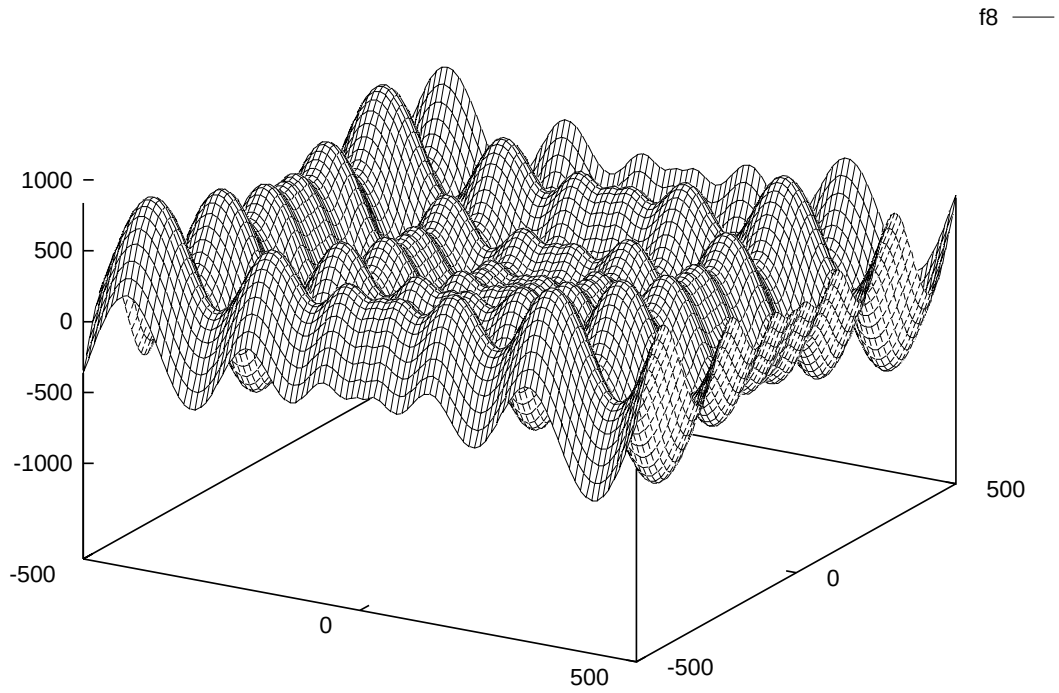


Figure 1: Function f_8.

Global Optimisation by Mutation-Based EAs

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

$$x_i'(j) \quad = \quad x_i(j) + N_j(0,1) \tag{1}$$

$$\tag{2}$$

where $x_i(j)$ denotes the j-th component of the vectors x_i. $N(0,1)$ 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.

4. Calculate the fitness of each offspring.
5. 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.

Why $N(0,1)$?

1. The standard deviation of the Normal 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.

Function Optimisation by Classical EP (CEP)

EP = Evolutionary Programming

1. Generate the initial population of μ individuals, and set $k = 1$. Each individual is taken as a pair of real-valued vectors, (x_i, η_i), $\forall i \in \{1, \cdots, \mu\}$.
2. Evaluate the fitness score for each individual (x_i, η_i), $\forall i \in \{1, \cdots, \mu\}$, of the population based on the objective function, $f(x_i)$.
3. Each parent (x_i, η_i), $i = 1, \cdots, \mu$, creates a single offspring (x_i', η_i') by: for $j = 1, \cdots, n$,

$$x_i'(j) = x_i(j) + \eta_i(j) N_j(0,1), \quad (3)$$

$$\eta_i'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (4)$$

where $x_i(j)$, $x_i'(j)$, $\eta_i(j)$ and $\eta_i'(j)$ denote the j-th component of the vectors x_i, x_i', η_i and η_i', 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' have commonly set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$.

4. Calculate the fitness of each offspring (x_i', η_i'), $\forall i \in \{1, \cdots, \mu\}$.

5. Conduct pairwise comparison over the union of parents (x_i, η_i) and offspring (x_i', η_i'), $\forall i \in \{1, \cdots, \mu\}$. 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 individuals out of (x_i, η_i) and $(x_i{}', \eta_i{}')$, $\forall i \in \{1, \cdots, \mu\}$, that have the most wins to be parents of the next generation.

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

What Do Mutation and Self-Adaptation Do

Fast EP

- The idea comes from fast simulated annealing.
- Use a Cauchy, instead of Gaussian, random number in Eq.(3) to **generate** a new offspring. That is,

$$x_i'(j) = x_i(j) + \eta_i(j)\delta_j \tag{5}$$

  where δ_j is an Cauchy random number variable with the scale parameter $t = 1$, and is generated anew for each value of j.
- Everything else, including Eq.(4), are kept unchanged in order to evaluate the impact of Cauchy random numbers.

Cauchy Distribution

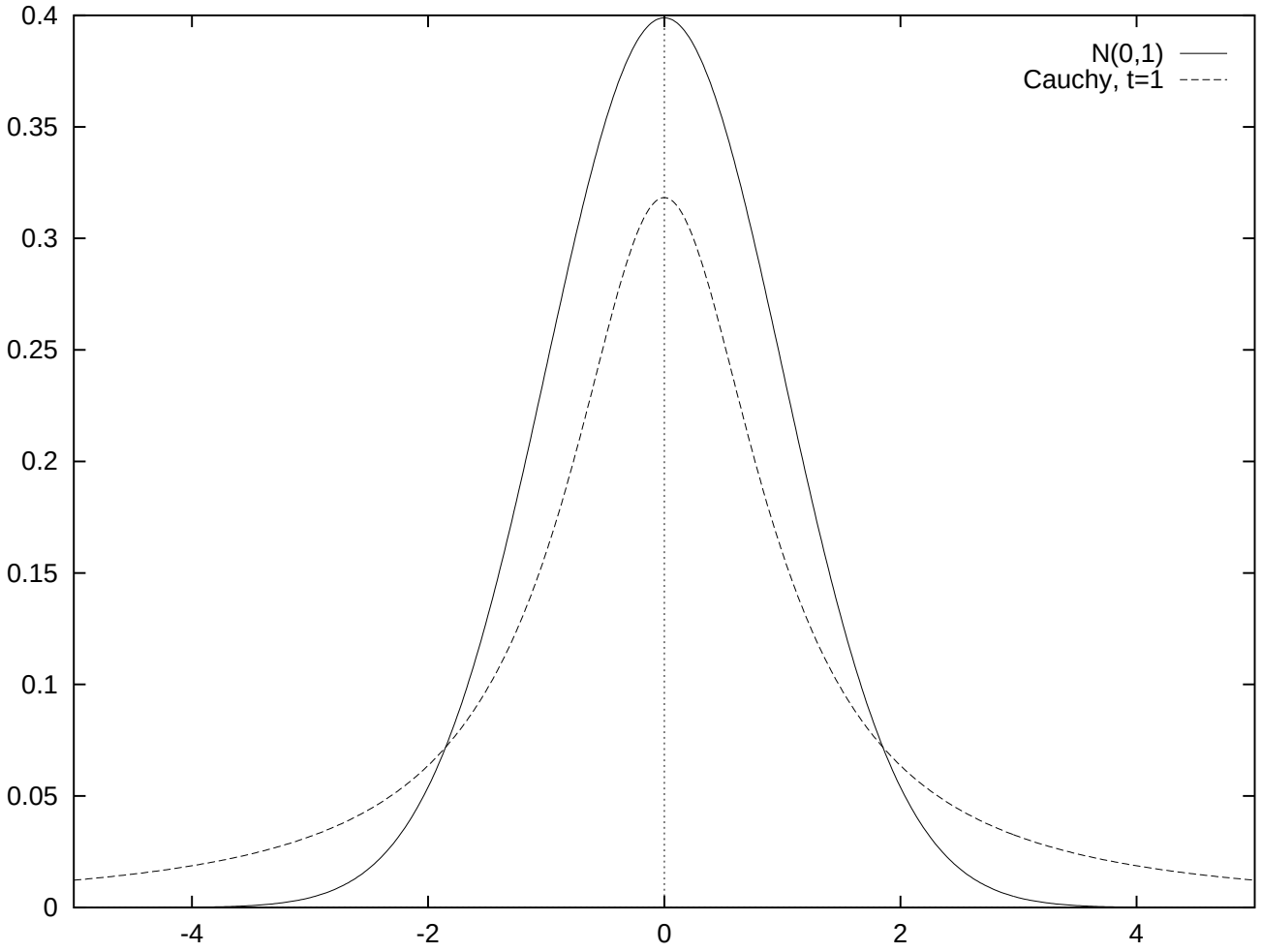
Its density function is

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

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

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

Gaussian and Cauchy Density Functions



Test Functions

- 23 functions were used in our computational studies. They have different characteristics.
- Some have a relatively high dimension.
- Some have many local optima.

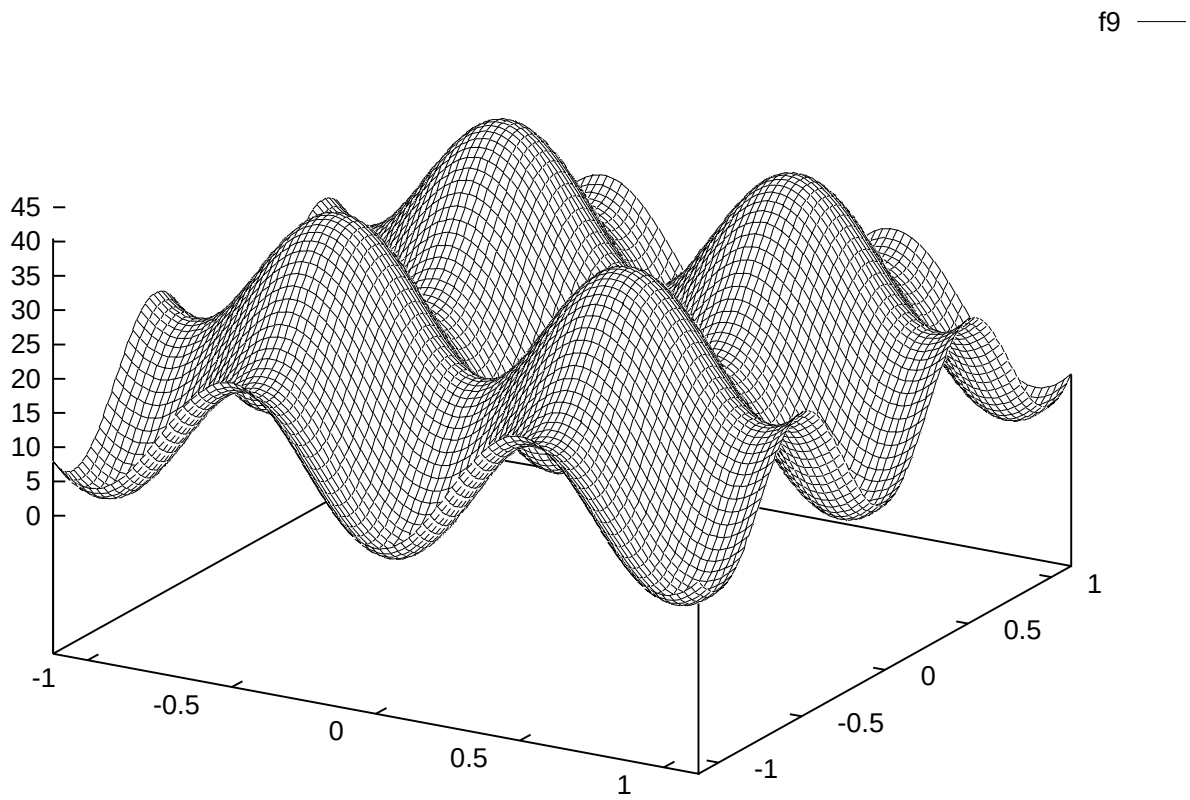


Figure 2: Function f_9 at a closer look.

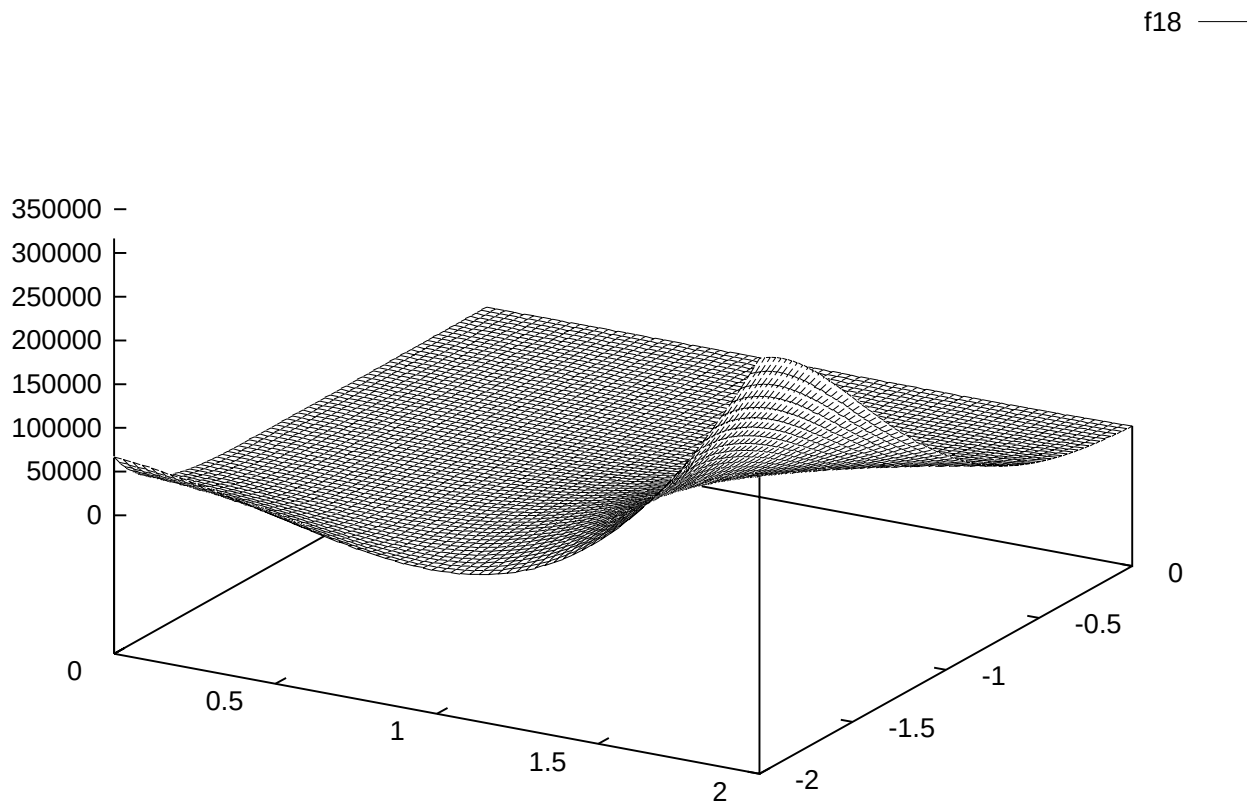


Figure 3: Function f_{18} at a closer look.

Experimental Setup

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

Experiments on Unimodal Functions

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
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.

Discussions on Unimodal Functions

- 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 Multimodal Functions f_8–f_{13}

F	No. of	FEP		CEP		FEP−CEP
	Gen's	Mean Best	Std Dev	Mean Best	Std Dev	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.

Discussions on Multimodal Functions f_8–f_{13}

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

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

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.

Discussions on Multimodal Functions f_{14}–f_{23}

- 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 so that CEP appeared to be better?

Experiments on Low-Dimensional f_8–f_{13}

F	No. of	FEP		CEP		FEP−CEP
	Gen's	Mean Best	Std Dev	Mean Best	Std Dev	t-test
f_8	500	-2061.74	58.79	-1762.45	176.21	$-11.17^{\dagger}$
f_9	400	0.14	0.40	4.08	3.08	$-8.89^{\dagger}$
f_{10}	400	8.6×10^{-4}	1.8×10^{-4}	8.1×10^{-2}	0.34	-1.67
f_{11}	1500	5.3×10^{-2}	4.2×10^{-2}	0.14	0.12	$-4.64^{\dagger}$
f_{12}	200	1.5×10^{-7}	1.2×10^{-7}	2.5×10^{-2}	0.12	-1.43
f_{13}	200	3.5×10^{-7}	1.8×10^{-7}	3.8×10^{-3}	1.4×10^{-2}	-1.89

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

Discussions on Low-Dimensional f_8–f_{13}

- FEP still converged faster to better solutions.
- Dimensionality does not play a major role in causing the difference between FEP and CEP.
- There must be something inherent in those functions which caused such difference.

The Impact of Parameter t on FEP — Part I

Table 1: The mean best solutions found by FEP using different scale parameter t in the Cauchy mutation for functions $f_1(1500)$, $f_2(2000)$, $f_{10}(1500)$, $f_{11}(2000)$, $f_{21}(100)$, $f_{22}(100)$ and $f_{23}(100)$. The values in "()" indicate the number of generations used in FEP. All results have been averaged over 50 runs.

Function	$t = 0.0156$	$t = 0.0313$	$t = 0.0625$	$t = 0.1250$	$t = 0.2500$
f_1	1.0435	0.0599	0.0038	1.5×10^{-4}	6.5×10^{-5}
f_2	3.8×10^{-4}	3.1×10^{-4}	5.9×10^{-4}	0.0011	0.0021
f_{10}	1.5627	0.2858	0.0061	0.0030	0.0050
f_{11}	1.0121	0.2237	0.1093	0.0740	0.0368
f_{21}	−6.9236	−7.7261	−8.0487	−8.6473	−8.0932
f_{22}	−7.9211	−8.3719	−9.1735	−9.8401	−9.1587
f_{23}	−7.8588	−8.6935	−9.4663	−9.2627	−9.8107

The Impact of Parameter t on FEP — Part II

Table 2: The mean best solutions found by FEP using different scale parameter t in the Cauchy mutation for functions $f_1(1500)$, $f_2(2000)$, $f_{10}(1500)$, $f_{11}(2000)$, $f_{21}(100)$, $f_{22}(100)$ and $f_{23}(100)$. The values in "()" indicate the number of generations used in FEP. All results have been averaged over 50 runs.

Function	$t = 0.5000$	$t = 0.7500$	$t = 1.0000$	$t = 1.2500$	$t = 1.5000$
f_1	1.8×10^{-4}	3.5×10^{-4}	5.7×10^{-4}	8.2×10^{-4}	0.0012
f_2	0.0041	0.0060	0.0081	0.0101	0.0120
f_{10}	0.0091	0.0136	0.0183	0.0227	9.1987
f_{11}	0.0274	0.0233	0.0161	0.0202	0.0121
f_{21}	−6.6272	−5.2845	−5.5189	−5.0095	−5.0578
f_{22}	−7.6829	−6.9698	−5.5194	−6.1831	−5.6476
f_{23}	−8.5037	−7.8622	−6.5713	−6.1300	−6.5364

Why Cauchy Mutation Performed Better

Given $G(0,1)$ and $C(1)$, 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

(Only 1-d case is considered here for convenience's sake.)

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

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

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

$$P_{G(0,\sigma^2)}(|x - x^*| \leq \epsilon) = \int_{x^*-\epsilon}^{x^*+\epsilon} f_{G(0,\sigma^2)}(x)dx \tag{6}$$

where $\epsilon > 0$ is the neighbourhood size and σ is often regarded as the step size of the Gaussian mutation. Figure 4 illustrates the situation.

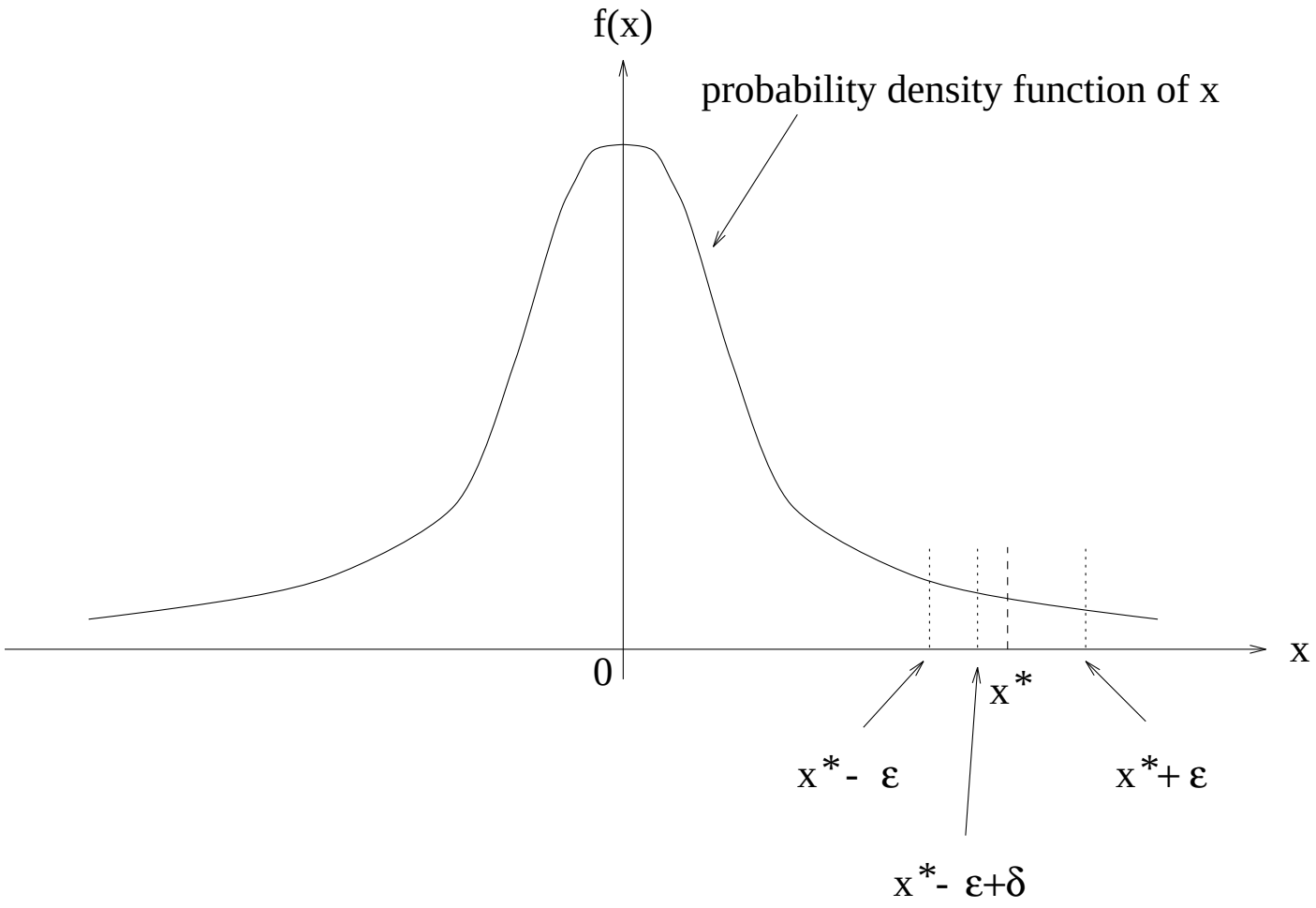


Figure 4: Evolutionary search as neighbourhood search, where x^* is the global optimum and $\epsilon > 0$ is the neighbourhood size.

An Analytical Result

It can be shown that

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

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

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

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

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

Empirical Evidence I

Table 3: Comparison of CEP's and FEP's final results on f_{21} when the initial population is generated uniformly at random in the range of $0 \leq x_i \leq 10$ and $2.5 \leq x_i \leq 5.5$. The results were averaged over 50 runs. The number of generations for each run was 100.

Initial Range	FEP		CEP		FEP−CEP
	Mean Best	Std Dev	Mean Best	Std Dev	t-test
$2.5 \leq x_i \leq 5.5$	−5.62	1.71	−7.90	2.85	$4.58^{\dagger}$
$0 \leq x_i \leq 10$	−5.57	1.54	−6.86	2.94	$2.94^{\dagger}$
t-test‡	−0.16		$-1.80^{\dagger}$		

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test. $^{\ddagger}\text{FEP(CEP)}_{small}-\text{FEP(CEP)}_{normal}$.

Empirical Evidence II

Table 4: Comparison of CEP's and FEP's final results on f_{21} when the initial population is generated uniformly at random in the range of $0 \leq x_i \leq 10$ and $0 \leq x_i \leq 100$ and a_i's were multiplied by 10. The results were averaged over 50 runs. The number of generations for each run was 100.

Initial Range	FEP		CEP		FEP−CEP
	Mean Best	Std Dev	Mean Best	Std Dev	t-test
$0 \leq x_i \leq 100$	−5.80	3.21	−5.59	2.97	−0.40
$0 \leq x_i \leq 10$	−5.57	1.54	−6.86	2.94	$2.94^{\dagger}$
t-test‡	−0.48		$2.10^{\dagger}$		

†The value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test. ‡FEP(CEP)$_{small}$−FEP(CEP)$_{normal}$.

Summary

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 more suitable parameters, instead of copying CEP's parameter setting.

Reference

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

Search Step Size and Search Bias

1. The search step size of mutation is crucial in deciding the search performance.
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

1. 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.
2. Rather than using either Gaussian or Cauchy mutations, we can use both. That is, two candidate offspring will be generated from every parent, one by Gaussian mutation and one by Cauchy mutation. The fitter one will survive as the single child.
3. The experimental results show that the improved fast EP (IFEP) is capable of performing as well as or better than the better one of FEP and CEP for most of the chosen test functions. This is achieved through a minimal change to the existing FEP and CEP.

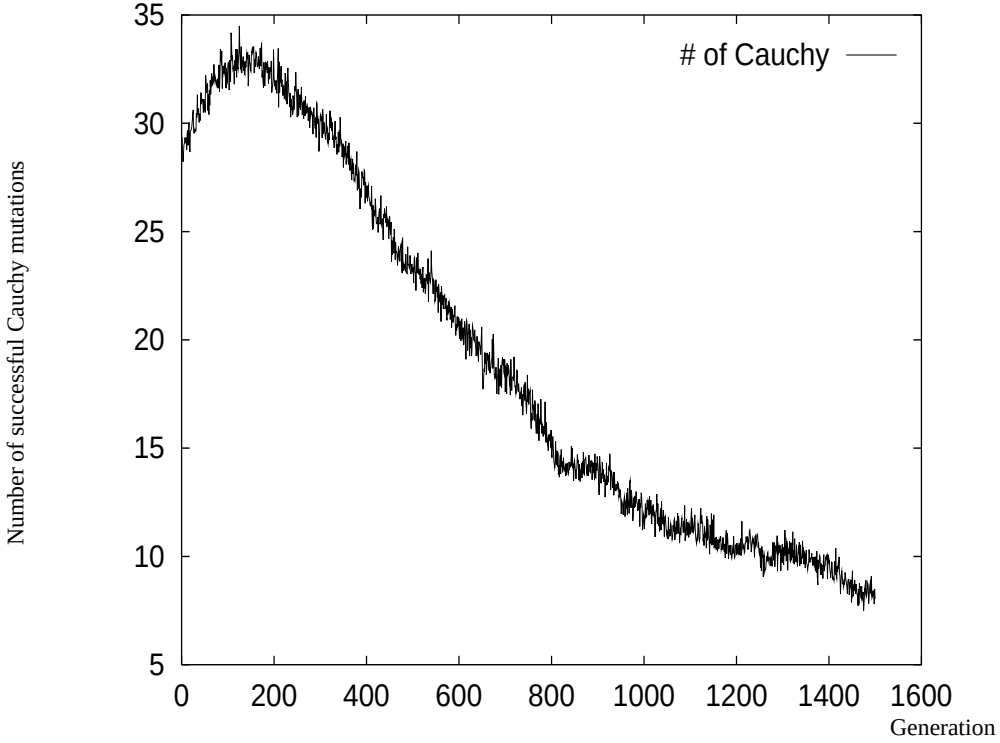


Figure 5: Number of successful Cauchy mutations in a population when IFEP is applied to function f_{10}. The vertical axis indicates the number of successful Cauchy mutations in a population and the horizontal axis indicates the number of generations. The results have been averaged over 50 runs.

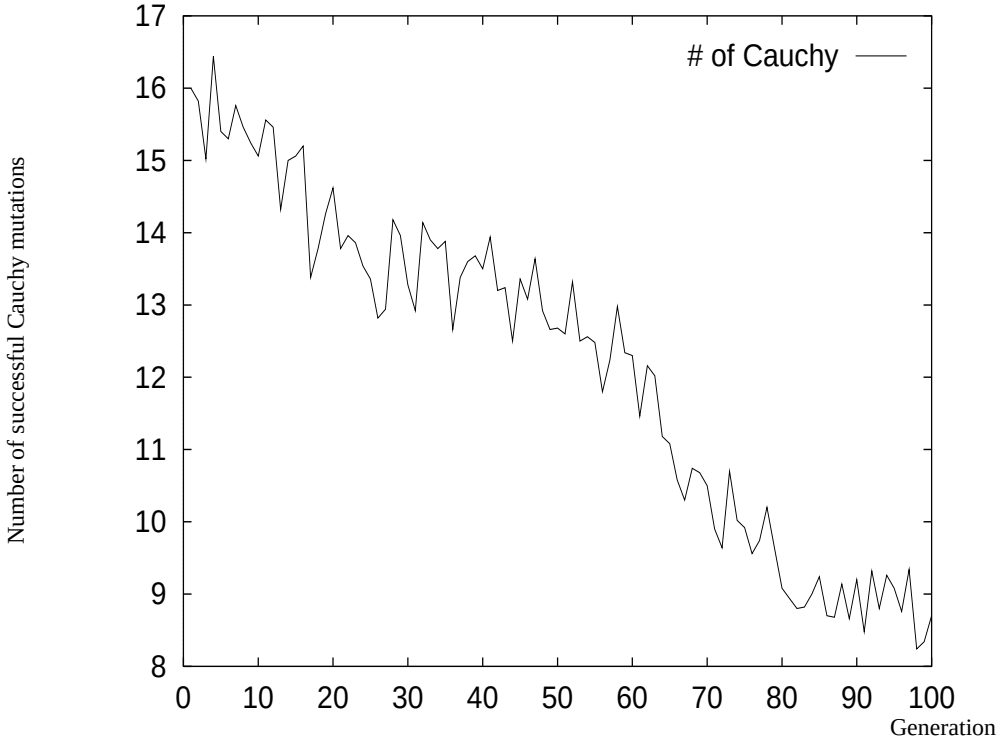


Figure 6: Number of successful Cauchy mutations in a population when IFEP is applied to function f_{21}. The vertical axis indicates the number of successful Cauchy mutations in a population and the horizontal axis indicates the number of generations. The results have been averaged over 50 runs.

Other Mixing Methods

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

$$x'_i(j) = x_i(j) + \eta_i(j)\,(0.5(N_j(0,1) + C_j(1)))$$

where $N_j(0,1)$ is a normally distributed number while $C_j(1)$ follows Cauchy distribution with parameter 1.

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

$$x'_i(j) = x_i(j) + \eta_{1i}(j)N_j(0,1) + \eta_{2i}(j)C_j(1)$$

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

A More General Self-Adaptive Method

1. The idea of mixing can be generalised to Lévy mutation.
2. Lévy probability distribution can be tuned to generate any distribution between the Gaussian and Cauchy probability distributions.
3. Hence we can use Lévy mutation with different parameters in EAs and let evolution to decide which one to use.

An Anomaly of Self-adaptation in EP

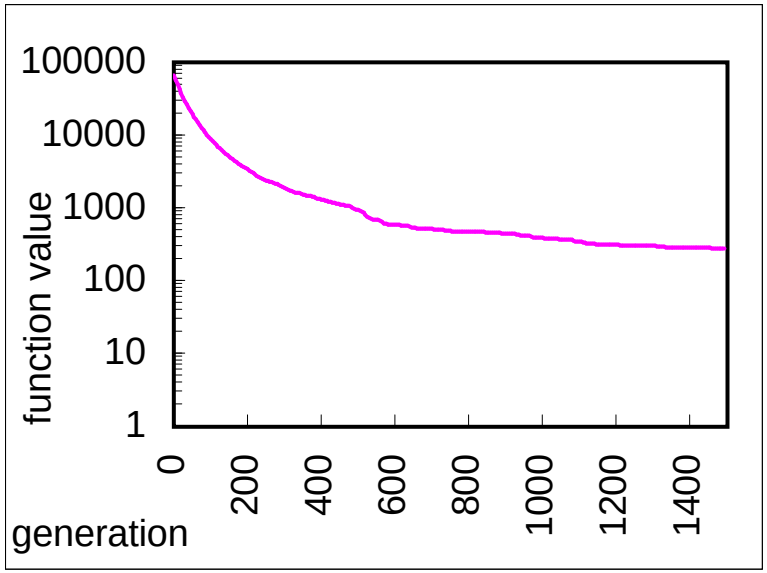


Figure 7: The 30-d sphere model stagnates early from mean of 50 runs.

Why EP Stagnates Early

Table 5: The 19-th component and the fitness of the best individual in a typical run.

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

Getting Around the Anomaly

Setting a lower bound! For example, set a fixed lower bound, e.g., 10^{-3}.

Use Success Rate to Adjust Lower Bounds

$$\eta_-^{t+1} = \eta_-^t \left(\frac{S_t}{A}\right), \tag{7}$$

where S_t is the success rate at generation t and A is a reference parameter, which has been set between 0.25 and 0.45 in our experiments. The success rate S_t is obtained by first computing the number of offspring selected for the next generation and then taking the ratio of successes to all offspring.

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 $\delta_i(j) = \eta_i'(j)N_j(0,1)$. We first calculate the average mutation step size from all accepted (successful) offspring:

$$\overline{\delta}(j) = \frac{1}{m}\sum_{v=1}^{m}\delta_v(j), j = 1, \cdots, n,$$

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

$$\eta_-^{t+1} = \text{median}\{\overline{\delta}(j), j = 1, 2, \ldots, n\}. \tag{8}$$

Getting Around the Anomaly — Recombination

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

Representation Is Important

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

1. 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*.
2. Gray coding can help, but does not solve the problem entirely. A better representation is to use integers or real numbers themselves.

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

1. 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.
2. The idea of self-adaptation can also be used in representations, where the most suitable representation will be evolved rather than fixed in advance.
3. For example, Cartesian and polar representations can be mixed adaptively in an EAs so that evolution can choose which representation is the best in the current stage of evolutionary search.

Summary

1. Search step size is a crucial factor in determining EA's performance.
2. Different operators, and EAs in general, have different search biases.
3. Mixing different operators and representations adaptively can lead to better performance for many (but not all) problems.
4. However, cares must be taken as self-adaptation does not always work as claimed.

References

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

2. 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.

3. 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.

4. K. Chellapilla, "Combining mutation operators in evolutionary programming," IEEE Transactions on Evolutionary Computation, 2(3):91-96, Sept. 1998.

The first three are downloadable from my web pages.

Two Approaches to Evolutionary Learning

Michigan Approach: Holland-style learning classifier systems (LCS), where each individual is a rule. The whole population is a complete (learning) system.

Pitt Approach: Each individual is a *complete* system.

This talk deals only with the Pitt-style evolutionary learning since it is more widely used.

Current Practice in Evolutionary Learning

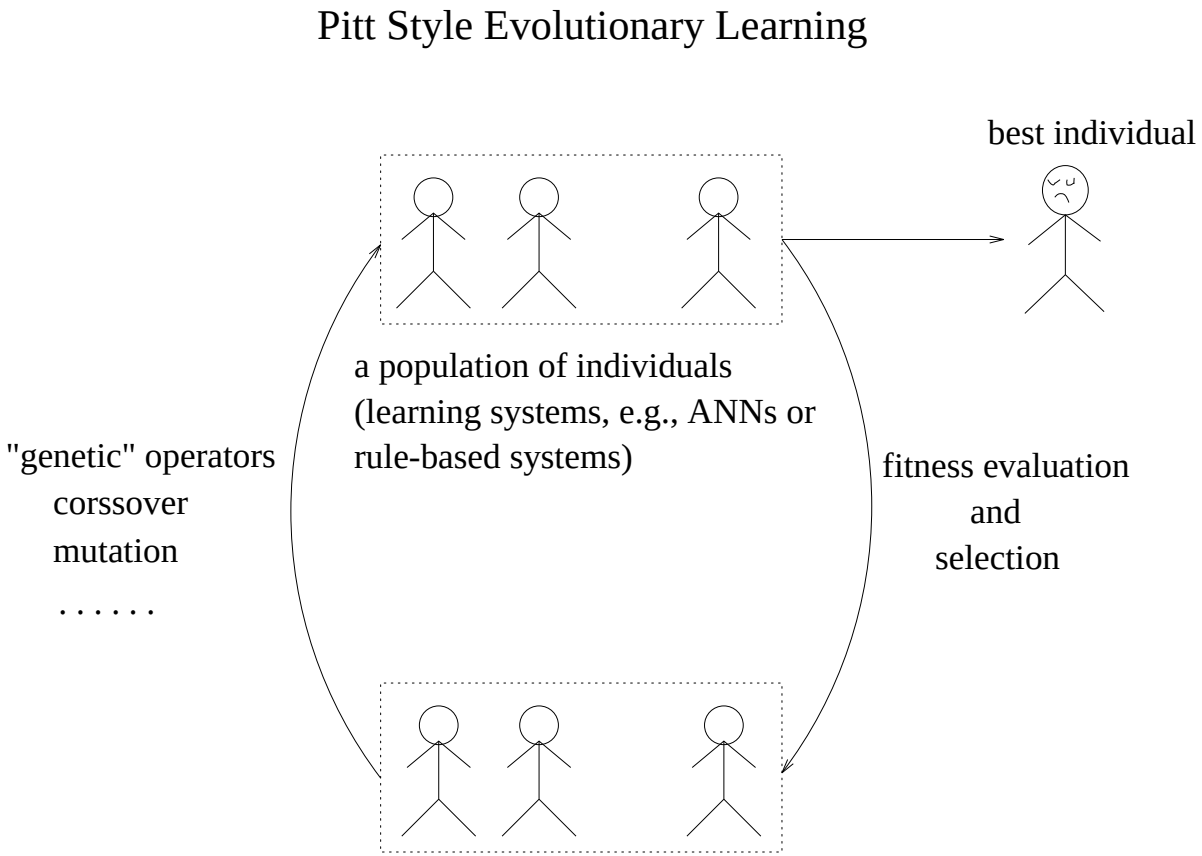


Figure 8: A general framework for Pitt style evolutionary learning.

Fitness Evaluation

1. Based on the training error.
2. Based on the training error and complexity (regularisation), i.e.,

$$\frac{1}{fitness} \propto error + \alpha * complexity$$

Evolutionary Learning and Optimisation

- Learning has often been formulated as an optimisation problem.
- However, learning is different from optimisation.

1. In optimisation, the fitness function reflects what is needed. The optimal value is always better than the second optimal one.
2. In learning, there is no way to quantify generalisation exactly. A system with minimum training error may not be the one with the best generalisation ability. *Why select the "best" individual in a population as the final output?*

Exploit Useful Information in a Population

- Since an individual with the minimum training error may not be the one with best generalisation, it makes sense to exploit useful information in a population rather than any single individual.
- All in all, it is a *population* that is evolving, not a single individual.
- Two types of experiments have been carried out to show that population does contain more information than any individuals and such information can be utilised effectively in evolutionary learning.

  Experiment 1: Each individual is an ANN.

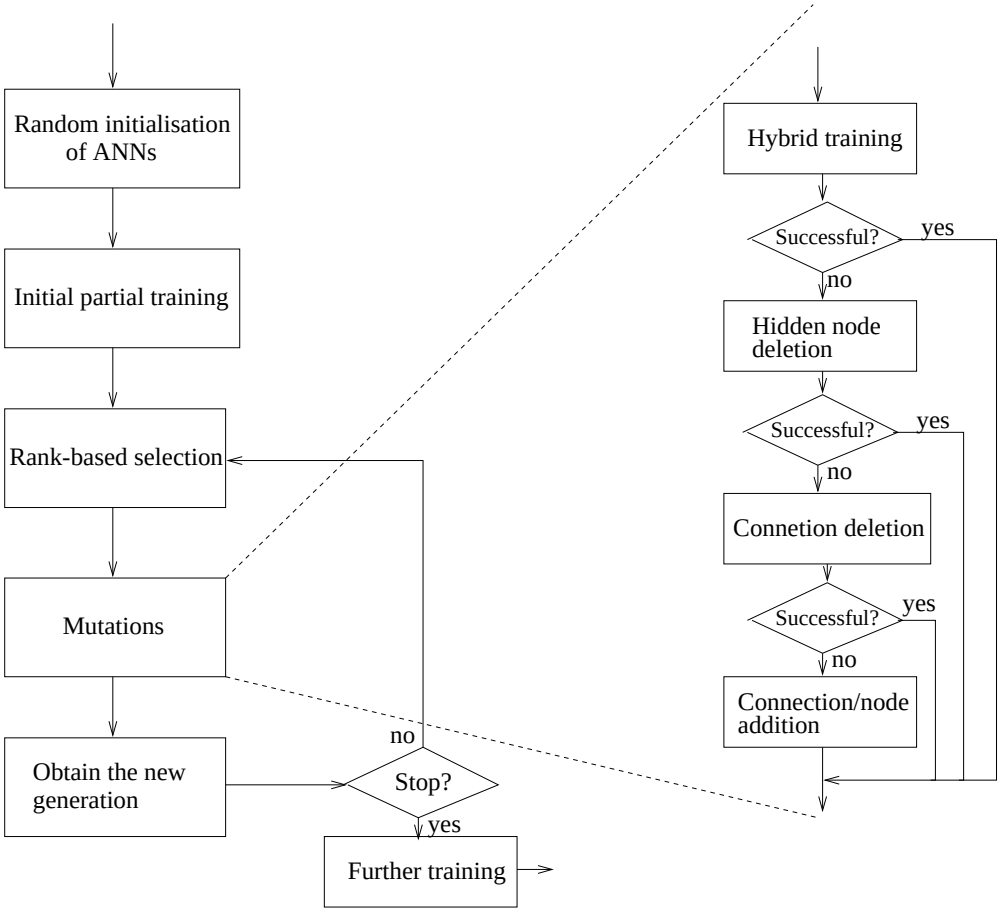
  Experiment 2: Each individual is a rule-based system.

Evolutionary Artificial Neural Networks

A simple idea to show the usefulness of population:

1. Use the "usual" evolutionary learning process to evolve NNs.
2. Instead of using the best individual as the final learned system, an integrated system which combines all the individuals in the final population is used as the final learned system.
3. This approach actually treats individuals as "modules" of an integrated system.
4. The final output from the integrated system is a linear combination of individuals' outputs.

The EANN System — EPNet



Experimental Studies on Modular EANNs

Three data sets were used in the experiments.

Australian Credit Card Data Set This two class problem has 690 examples in total. There are 14 attributes include 6 numeric values and 8 discrete ones.

Diabetes Data Set There are 500 examples of class 1 and 268 of class 2 for the two class problem. There are 8 attributes.

Heart Disease Data Set This database contains 13 attributes and 270 examples. There are two classes.

Experimental Results

Data set	Method	Testing Error Rate
Card	EPNet	0.100
	Ensemble	0.093
Diabetes	EPNet	0.232
	Ensemble	0.226
Heart	EPNet	0.154
	Ensemble	0.151

Automatic Modularisation

- The previous encouraging results were achieved without modifying the evolutionary learning process. It is hoped that more improvements could be obtained if some techniques were employed to take the modular approach into account. For example, different modules should deal with different aspects of a complex problem.
- Speciation by fitness sharing is one of such techniques which encourage *automatic* formation of species (i.e., modules).
- The following experiments demonstrate the effectiveness of automatic modularisation by speciation. In order to show the generality of the approach, a rule-based system was represented as an individual in the following experiments.
- The experiment was designed to learn strategies for playing the 2-player iterated prisoner's dilemma game.

The 2-Player Iterated Prisoner's Dilemma

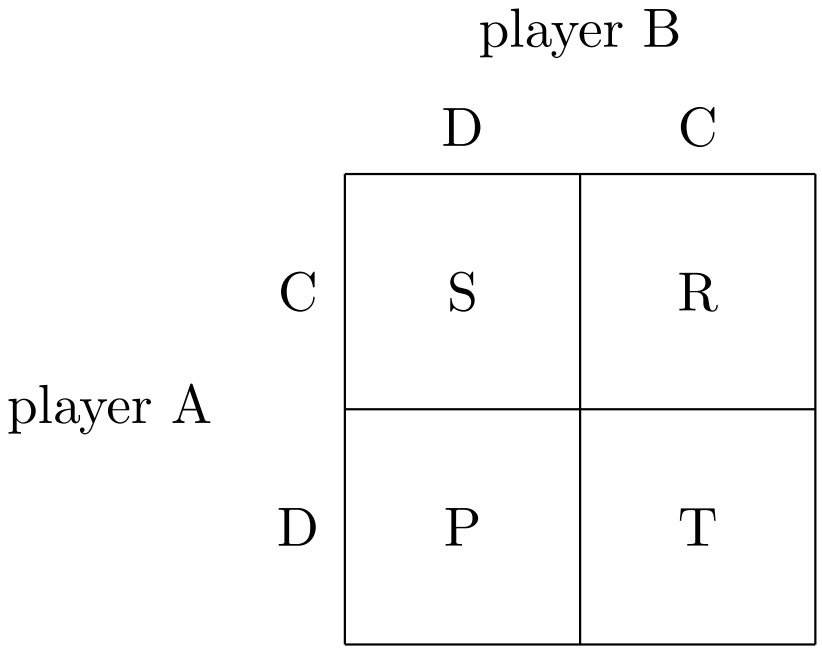


Figure 9: Two conditions must be satisfied: (1) $T > R > P > S$; and (2) $R > (S + T)/2$.

Speciation by Implicit Fitness Sharing

For each strategy i in the GA population, do the following C times:

1. From the GA population, select a sample of σ strategies.
2. Find the strategy in that sample which achieves the highest score (or the largest winning margin, if you prefer) against the single test strategy i.
3. The best in the sample receives payoff. In the case of a tie, payoff is shared equally among the tie-breakers.

Figure 10: Payoff function for implicit fitness sharing.

A Combination Method — the Gating Algorithm

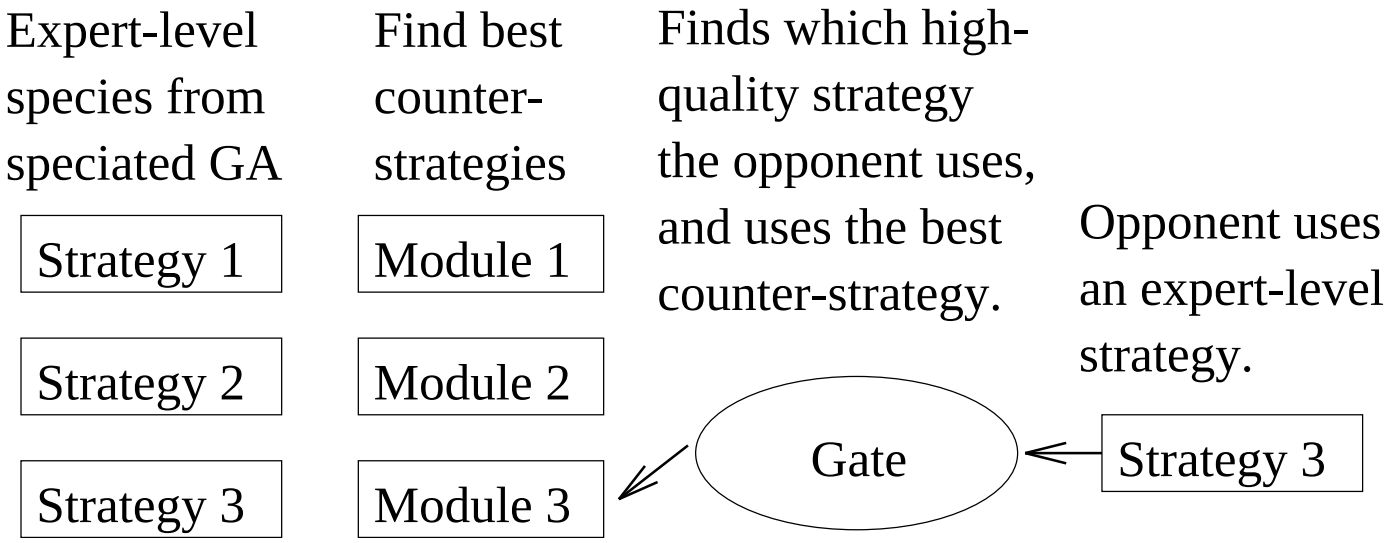


Figure 11: A gating algorithm for combining different expertise in a population together.

Experimental Results

	$l = 4$			
Strategy	Wins	Ties	Average Score	
	%	%	Own	Other's
best.ns	0.343	0.057	1.235	1.595
best.sr	0.360	0.059	1.322	1.513
gate.sr	0.643	0.059	1.520	1.234

Table 6: Results against the best 25 strategies from the partial enumerative search, for 2IPD with remembered history $l = 4$. The results were averaged over 30 runs.

NN Ensembles — Negatively Correlated Learning

The idea of designing different cooperative specialists is *not* limited to EAs. It can be used by gradient descent algorithms too.

1. Making individuals different:

$$E_i = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}(F_i(n) - d(n))^2 + \lambda p_i(n)\right)$$

where

$$p_i(n) = (F_i(n) - F(n))\sum_{j\neq i}(F_j(n) - F(n))$$

$F(n)$ is the ensemble output.

2. All individuals are learned *simultaneously*.

Ensemble Learning

We consider estimating $g(\mathbf{x}) = E[d|\mathbf{x}]$ by forming a simple average of a set of outputs of individual networks which are trained using the same training data set D

$$F(\mathbf{x}, D) = \frac{1}{M}\Sigma_{i=1}^{M} F_i(\mathbf{x}, D) \tag{9}$$

where $F_i(\mathbf{x}, D)$ is the actual response of network i and M is the number of neural network estimators.

Bias-Variance-Covariance Trade-off

Taking expectations with respect to the training set D, the expected mean-squared error of the combined system can be written in terms of individual network output

$$E_D\left[(E[d|x] - F(\mathbf{x}, D))^2\right] = (E_D[F(\mathbf{x}, D)] - E[d|x])^2 + E_D\left[\frac{1}{M^2}\Sigma_{i=1}^{M}\left(F_i(\mathbf{x}, D) - E_D[F_i(\mathbf{x}, D)]\right)^2\right] + E_D\left[\frac{1}{M^2}\Sigma_{i=1}^{M}\Sigma_{j\neq i}\left(F_i(\mathbf{x}, D) - E_D[F_i(\mathbf{x}, D)]\right)\right. \left.\left(F_j(\mathbf{x}, D) - E_D[F_j(\mathbf{x}, D)]\right)\right] \quad (10)$$

The expectation operator E_D represents the average over all the patterns in the training set D.

How to Choose the Correlation Penalty

The purpose of minimising p_i is to negatively correlate each individual's error with errors for the rest of the ensemble.

- The function p_i for regression problems can be chosen as

$$p_i(n) = (F_i(n) - d(n)) \, \Sigma_{j \neq i} \, (F_j(n) - d(n)) \tag{11}$$

  for noise free data, or

$$p_i(n) = (F_i(n) - F(n)) \, \Sigma_{j \neq i} \, (F_j(n) - F(n)) \tag{12}$$

  for noisy data.

- The function p_i for classification problem can be chosen as

$$p_i(n) = (F_i(n) - 0.5) \, \Sigma_{j \neq i} \, (F_j(n) - 0.5) \tag{13}$$

Experimental Studies

- The Australian credit card assessment problem was used.
- The whole data set is randomly partitioned into a training (518 cases) and a testing set (172 cases).
- The ensemble used in our experiment consisted of four strictly-layered feedforward neural networks. All individual networks had the same architecture. They had three layers and 5 hidden nodes in the hidden layer. The learning-rate η in BP is 0.1, and λ is 0.375. These parameters were chosen after limited preliminary experiments. They are not meant to be optimal.

Experiment Results: Error Rates

	Training set		Test set	
# epochs	500	1000	500	1000
Mean	0.1093	0.0846	0.1177	0.1163
SD	0.0092	0.0088	0.0182	0.0159
Min	0.0927	0.0676	0.0698	0.0756
Max	0.1255	0.1004	0.1628	0.1454

Table 7: Error rates for the Australian credit card assessment problem. The results were averaged over 25 runs.

Experiment Results: Evolutionary Process

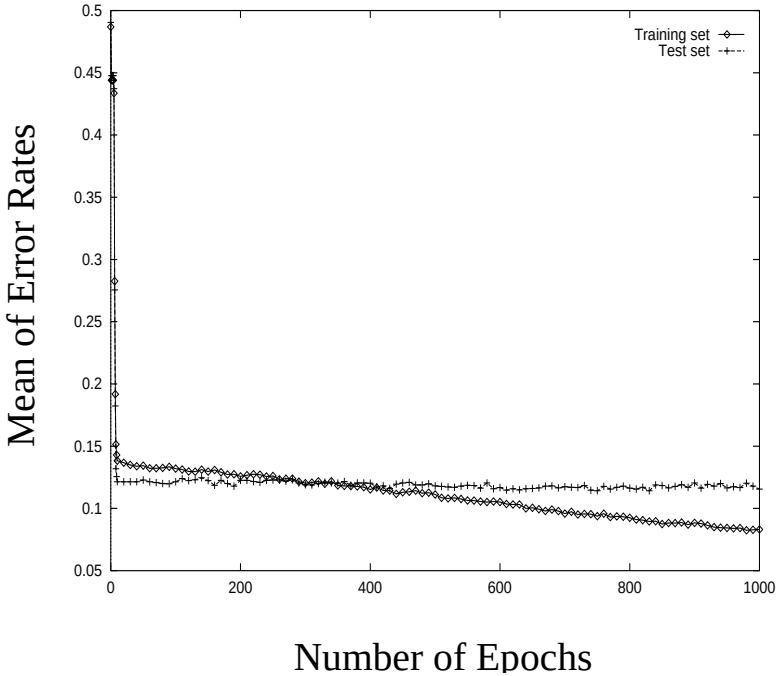


Figure 12: The average result over 25 runs of our ensemble learning algorithm. The horizontal axis indicates the number of training epochs. The vertical axis indicates the error rate.

Experiment Results: Being Different

$\Omega_1 = 146$	$\Omega_2 = 148$	$\Omega_3 = 149$	$\Omega_4 = 151$
$\Omega_{12} = 137$	$\Omega_{13} = 137$	$\Omega_{14} = 141$	$\Omega_{23} = 140$
$\Omega_{24} = 141$	$\Omega_{34} = 139$	$\Omega_{123} = 132$	$\Omega_{124} = 133$
$\Omega_{134} = 133$	$\Omega_{234} = 134$	$\Omega_{1234} = 128$	

Table 8: The sizes of the correct response sets of individual networks on the testing set and their intersections for the Australian credit card assessment problem.

Comparison with Other Work

Algorithm	TER	Algorithm	TER
NCNN	0.116	Logdisc	0.141
EPNet	0.115	CART	0.145
Evo-En-RLS	0.131	RBF	0.145
Cal5	0.137	CASTLE	0.148
ITrule	0.141	NaiveBay	0.151
DIPOL92	0.141	IndCART	0.152

Table 9: Comparison among the negative correlation NN (NCNN), EPNet, an evolutionary ensemble learning algorithm (Evo-En-RLS), and others in terms of the average testing error rate. TER stands for Testing Error Rate in the table.

Mackey-Glass Time Series Prediction Problem

Method	Testing RMS	
	$\Delta t = 6$	$\Delta t = 84$
NCNN	0.01	0.03
EPNet	0.02	0.06
BP	0.02	0.05
CC Learning	0.06	0.32

Table 10: The "Testing RMS" in the table refers to the normalised root-mean-square error on the testing set.

Comparision between NCNN and ME

Adding noises.

Method	E_{mse}	
	$\sigma^2 = 0.1$	$\sigma^2 = 0.2$
NCNN	0.012	0.023
ME	0.018	0.038

Table 11: Comparison between NCNN and the mixtures-of-experts (ME) architectures in terms of the integrated mean-squared error on the testing set for the moderate noise case and the large noise case.

Three Approaches of Ensemble Learning

- Independent Training
- Sequential Training
- Simultaneous Training

Two Classes of Gaussian-Distributed Patterns

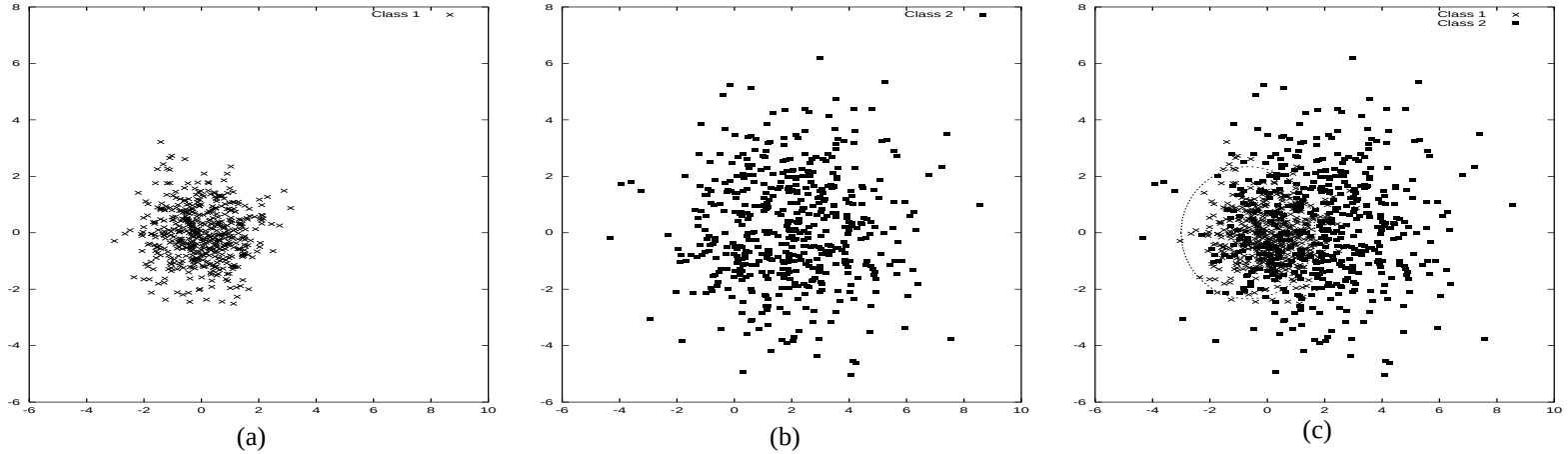


Figure 13: (a) Scatter plot of Class 1. (b) Scatter plot of Class 2. (c) Combined scatter plot of both classes. The circle represents the optimum Bayes solution.

Approach of Independent Training

No advantage to combine a set of identical neural networks. In order to create different neural networks, independent training approach trains a set of neural networks independently by

- varying initial random weights
- varying the architectures
- varying the learning algorithm used
- varying the data, such as using cross-validation
- ... ...

Approach of Sequential Training

In order to decorrelate the individual neural networks, sequential training approach trains a set of networks in a particular order, such as the boosting algorithm:

- Train the first neural network with randomly chosen N_1 patterns
- Select N_2 patterns on which the first neural network would have 50% error rate. Train the second neural network with the selected patterns.
- Select N_3 patterns on which the first two trained neural networks disagree. Train the third neural network with the selected patterns.

Approach of Simultaneous Training

- The mixtures-of-experts (ME) architectures: consists of two types of networks, i.e., a gating network and a number of expert networks . Each expert network makes individual decision on its covered region. The gating network weights the outputs of the expert networks to provide an overall best decision.
- Negative correlation learning (NCL): no gating network is needed in NCL. The idea of NCL is to introduce a correlation penalty term into the error function of each individual network so that the individual network can be trained simultaneously and interactively.

Decision Boundaries by the Independent Training

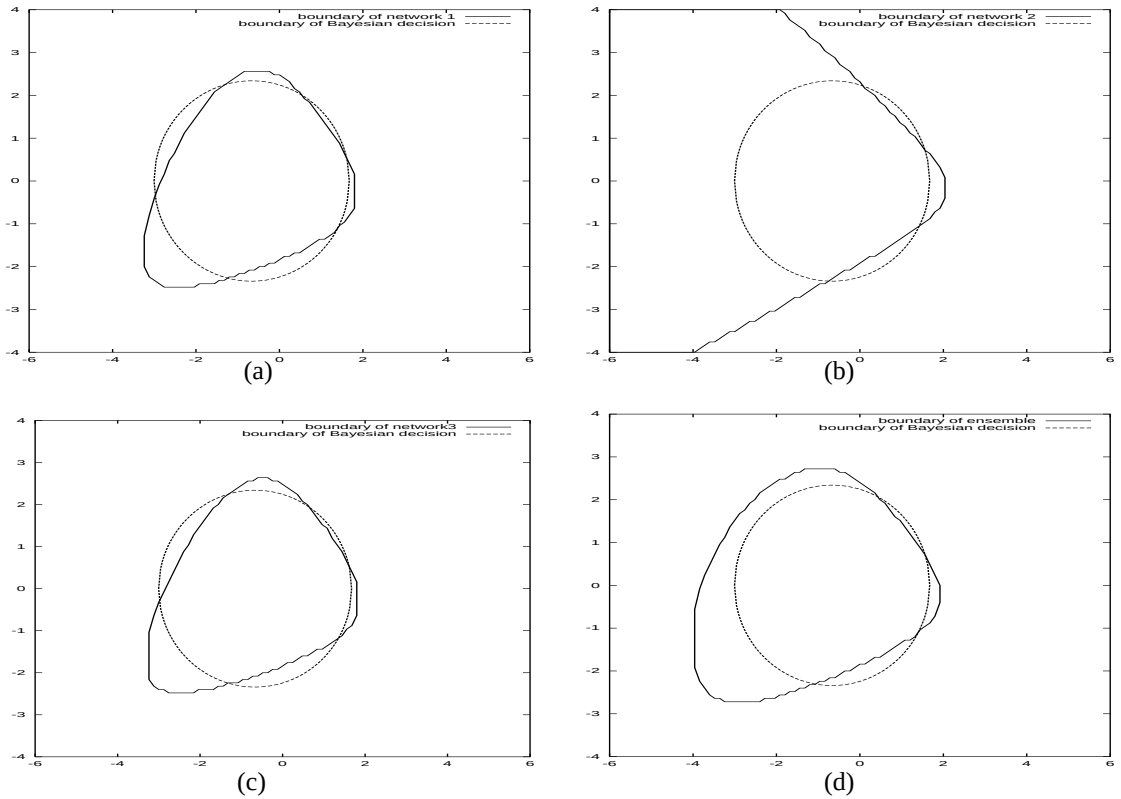


Figure 14: (a) Network 1. (b) Network 2. (c) Network 3. (d) Ensemble. The circle represents the optimum Bayes solution.

Decision Boundaries by the Boosting Algorithm

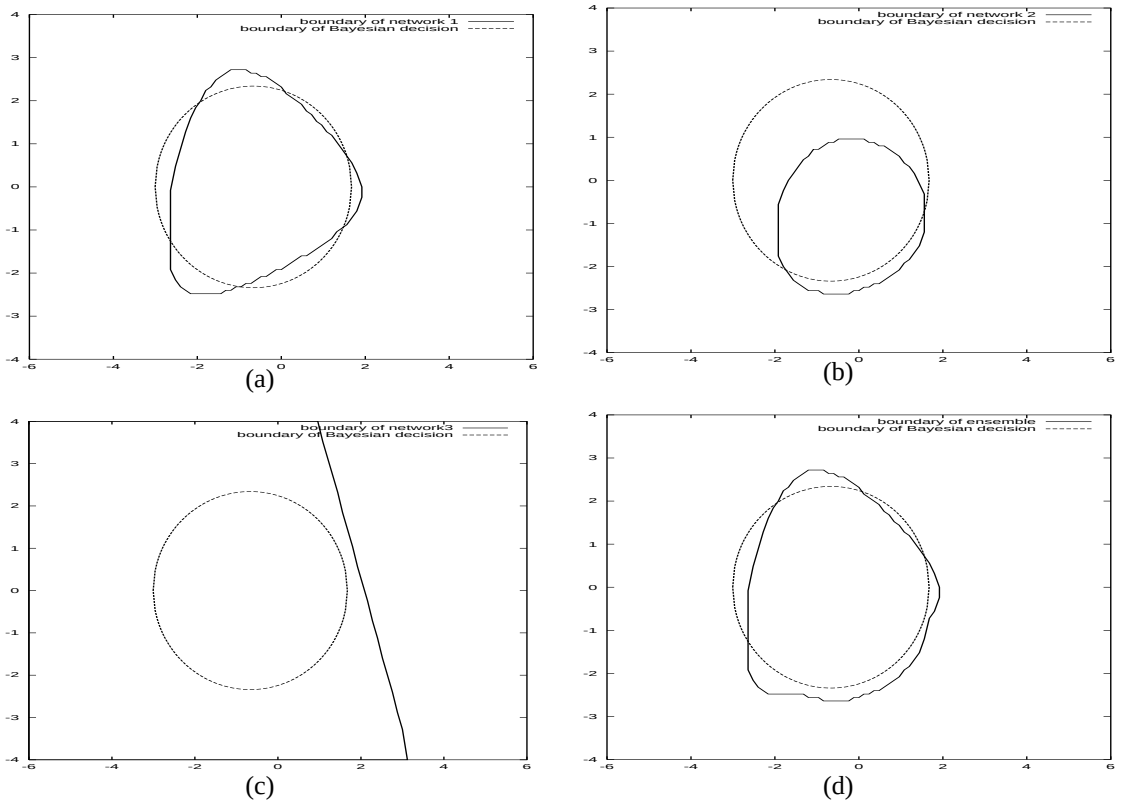


Figure 15: (a) Network 1. (b) Network 2. (c) Network 3. (d) Ensemble. The circle represents the optimum Bayes solution.

Decision Boundaries by NCL

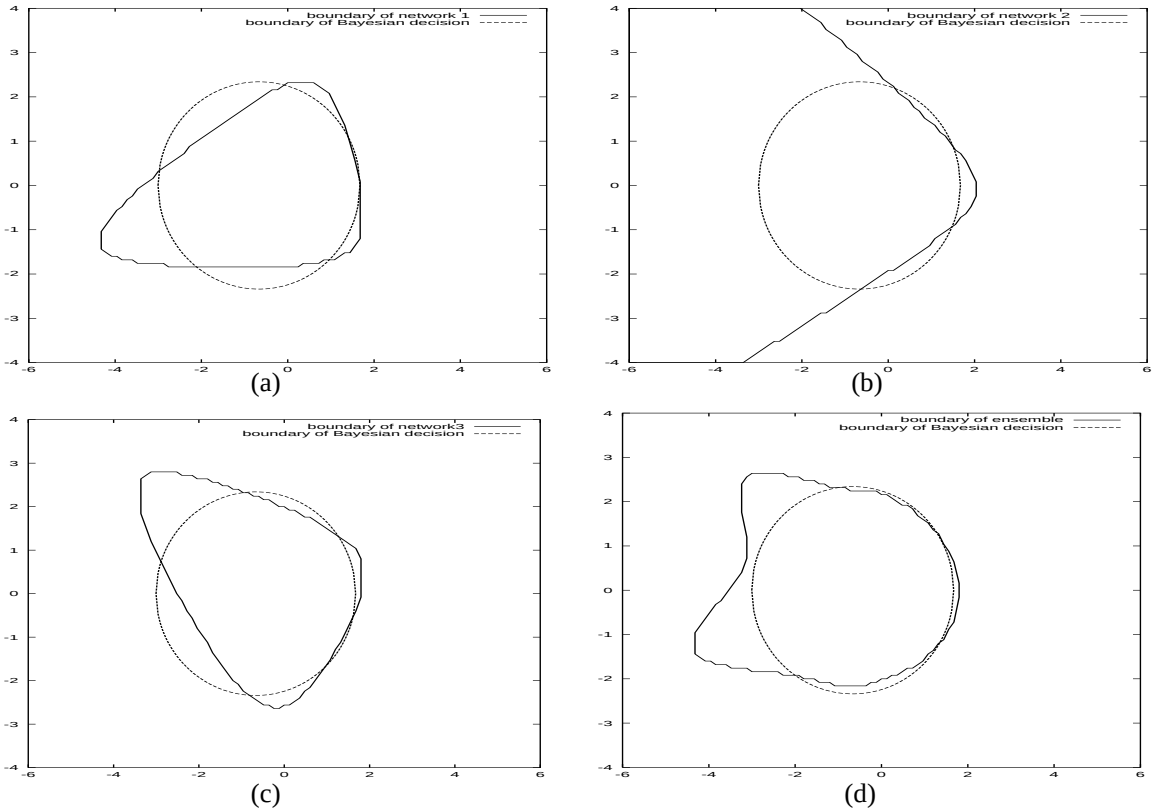


Figure 16: (a) Network 1. (b) Network 2. (c) Network 3. (d) Ensemble. The circle represents the optimum Bayes solution.

Comparisons

Boosting Training			
Network 1	Network 2	Network 3	Ensemble
81.11	75.26	73.09	81.03
Negative Correlation Learning (NCL)			
Network 1	Network 2	Network 3	Ensemble
80.71	80.55	80.97	81.41
Independent Training ($\lambda = 0$ in NCL)			
Network 1	Network 2	Network 3	Ensemble
81.13	80.48	81.13	80.99

Discussions

- The independently trained neural networks tended to generate similar decision boundaries because of lack of interactions among the individual networks during learning.
- The boosting algorithm performed well, but was hindered by its data filtering process which generated highly unbalance training data points. For example, the ensemble performance actually got worse than network 1.
- No process of filtering data is needed in NCL. The performance of NCL (81.41) is very close to the theoretical optimum (81.51).

Evolving ANN Ensembles

No need to predefine the number of ANNs in an ensemble.

1. Generate an initial population of M NNs, and set $k = 1$.
2. Train each NN in the population on the training set for a certain number of epochs using the negative correlation learning.
3. Randomly choose n_b NNs as parents to create n_b offspring NNs by Gaussian mutation.
4. Add the n_b offspring NNs to the population and train the offspring NNs using the negative correlation learning while the rest NNs' weights are frozen.
5. Calculate the fitness of each NN in the population and prune the population to the M fittest NNs.
6. Stop if certain criteria are satisfied and go to Step 7. Otherwise, $k = k + 1$ and go to Step 3.
7. Cluster the NNs in the population. These clusters are then used to construct NN ensembles.

Fitness Sharing and Fitness Evaluation

- An implicit fitness sharing is used based on the idea of "covering" the same training case by shared individuals. The procedure of calculating shared fitness is carried out case-by-case over the training set.
- For each training case, if there are $p > 0$ individuals that correctly classify it, then each of these p individuals receives $1/p$ fitness reward, and the rest individuals in the population receive zero fitness reward. The fitness reward is summed over all training cases. This method is expected to generate a smoother shared fitness landscape.

Accuracy on the Australian Credit Card Problem

Accuracy	Simple Averaging		Majority Voting		Winner-Takes-All	
Rate	Training	Testing	Training	Testing	Training	Testing
Mean	0.910	0.855	0.917	0.857	0.887	0.865
SD	0.010	0.039	0.010	0.039	0.007	0.028
Min	0.897	0.797	0.900	0.812	0.874	0.812
Max	0.924	0.913	0.928	0.913	0.895	0.913

Table 12: The results are averaged on 10-fold cross-validation.

Accuracy on the Diabetes Data Set

Accuracy	Simple Averaging		Majority Voting		Winner-Takes-All	
Rate	Training	Testing	Training	Testing	Training	Testing
Mean	0.795	0.766	0.802	0.764	0.783	0.779
SD	0.007	0.039	0.007	0.042	0.007	0.045
Min	0.783	0.703	0.786	0.688	0.774	0.703
Max	0.805	0.828	0.810	0.828	0.794	0.844

Table 13: The results are averaged on 12-fold cross-validation.

Comparisons on the Australian Data Set (Error Rate)

Algorithm	Error Rate	Algorithm	Error Rate	Algorithm	Error Rate
EENCL	0.135, 0.132	CART	0.145	ITrule	0.137
Discrim	0.141	IndCART	0.152	Cal5	0.131
Quadisc	0.207	NewID	0.181	Kohonen	FD
Logdisc	0.141	AC^2	0.181	DIPOL92	0.141
SMART	0.158	Baytree	0.171	Backprop	0.154
ALLOC80	0.201	NaiveBay	0.151	RBF	0.145
k-NN	0.181	CN2	0.204	LVQ	0.197
CASTLE	0.148	C4.5	0.155	Default	0.440

Table 14: The results are averaged on 10-fold cross-validation. Two error rates are listed for EENCL, which are the results for the ensembles using the whole population and the ensembles using the representatives from species.

Comparisons on the Diabetes Data Set (Error Rate)

Algorithm	Error Rate	Algorithm	Error Rate	Algorithm	Error Rate
EENCL	0.221, 0.223	CART	0.255	ITrule	0.245
Discrim	0.225	IndCART	0.271	Cal5	0.250
Quadisc	0.262	NewID	0.289	Kohonen	0.273
Logdisc	0.223	AC^2	0.276	DIPOL92	0.224
SMART	0.232	Baytree	0.271	Backprop	0.248
ALLOC80	0.301	NaiveBay	0.262	RBF	0.243
k-NN	0.324	CN2	0.289	LVQ	0.272
CASTLE	0.258	C4.5	0.270		

Table 15: The results are averaged on 12-fold cross-validation.

Constructive Neural Network Ensemble (CNNE)

- CNNE is non-evolutionary.
- CNNE uses incremental training, in association with negative correlation learning, in determining ensemble architectures.
- Individual NNs and hidden nodes are added to the ensemble architecture one by one in a constructive fashion during training.

Md. Monirul Islam, X. Yao and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," Submitted to *IEEE Transactions on Neural Networks*, 2002.

Conclusion

1. Learning is different from optimisation, especially in evolutionary computation.
2. Population contains more information than any single individual. Exploiting useful population information can be achieved for different kinds of population-based learning, either evolutionary or non-evolutionary.
3. Progresses have been made in the last few years towards automatic divide-and-conquer using both evolutionary and non-evolutionary approaches, although more needs to be done.

References (Downloadable)

1. Y. Liu, X. Yao and T. Higuchi, "Evolutionary Ensembles with Negative Correlation Learning," *IEEE Transactions on Evolutionary Computation*, **4**(4):380-387, November 2000.

2. X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, **87**(9):1423-1447, September 1999.

3. Y. Liu and X. Yao, "Simultaneous training of negatively correlated neural networks in an ensemble," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, **29**(6):716-725, December 1999.

4. X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, **28**(3):417-425, June 1998.

5. X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. on Neural Networks*, **8**(3):694-713, May 1997.

6. P. Darwen and X. Yao, "Speciation as automatic categorical modularization," *IEEE Trans. on Evolutionary Computation*, **1**(2):101-108, 1997.