What Can We Learn From Evolutionary Optimisation

Xin Yao (x.yao@cs.bham.ac.uk)
The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA)
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham B15 2TT
UK

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

EAs as Stochastic Population-Based Generate-and-Test

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

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

Can you think of an algorithm that is simpler than that?

Evolutionary Algorithms (EAs)

- There are several well-known EAs (e.g., genetic algorithms, evolution strategies, evolutionary programming, genetic programming, etc) with different historical backgrounds, representations, variation operators, and selection schemes.
- 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)**.

Major Areas in Evolutionary Computation

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

Overview of This Talk

1. Global optimisation.
2. Landscape approximation and hybrid algorithms
3. Differential evolution using cooperative coevolution
4. Constraint handling
5. Concluding remarks

Global Optimisation: An Example

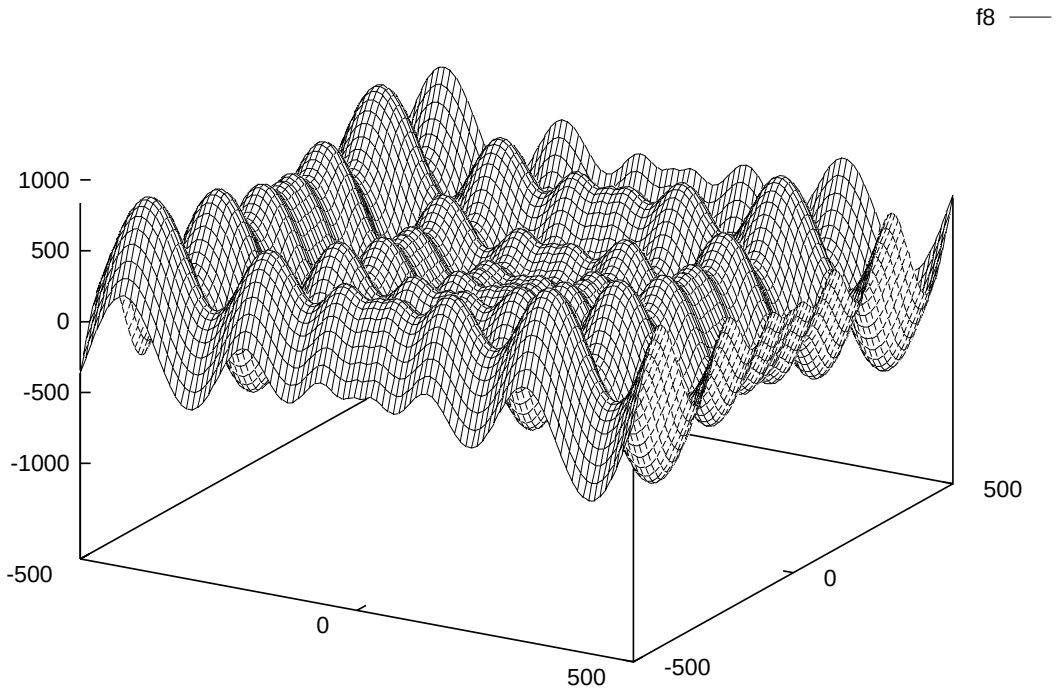


Figure 1: Function f_8.

Simple Mutation-Based EAs

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

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

   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 $N(0,1)$: Mutation Step Size

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.

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 $(x_i{}', \eta_i{}')$ by: for $j = 1, \cdots, n$,

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

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

where 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. For each individual from the union of parents (x_i, η_i) and offspring (x_i', η_i'), $\forall i \in \{1, \cdots, \mu\}$, q opponents are chosen at random. 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', η_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.

How Well Does CEP Work?

- It worked very well for many problems.
- Evolving strategy parameters, η_i, was hailed as a major progress and called **self-adaptation**.
- However, not always good news. Why?
- What Do Mutation and Self-Adaptation Do?

Fast EP

- The idea comes from fast simulated annealing. **A simple idea!**
- 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 variable with the scale parameter $t = 1$.

- Everything else, including Eq.(3), are kept unchanged in order to evaluate the impact of Cauchy mutation on the results.

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.

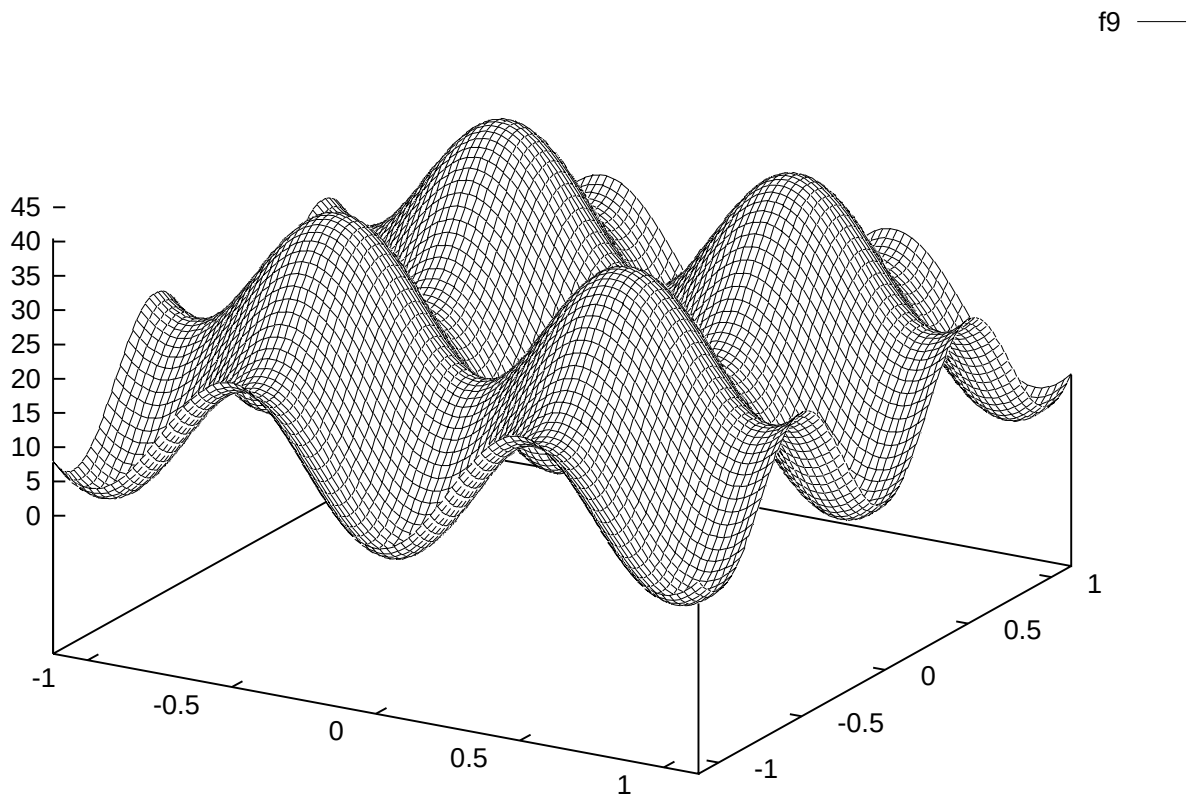


Figure 2: Function f_9 at a closer look.

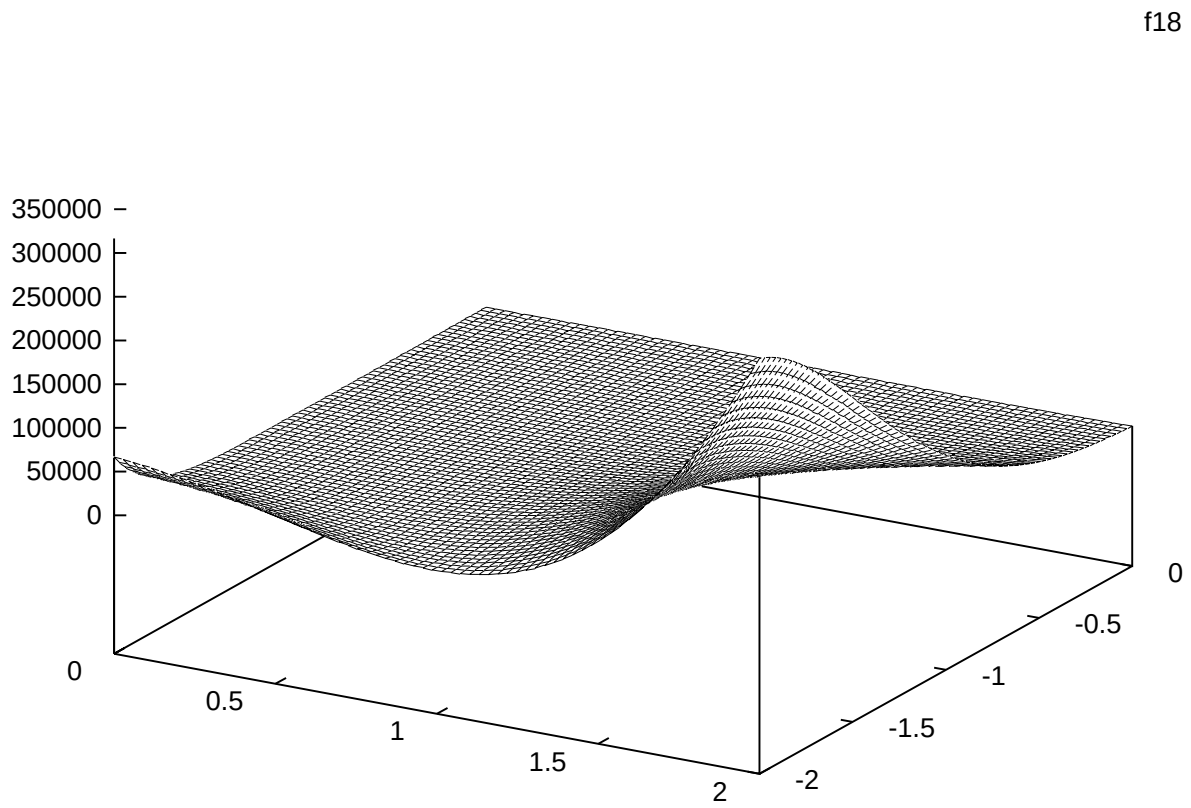


Figure 3: Function f_{18} at a closer look.

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.

Cauchy Distribution

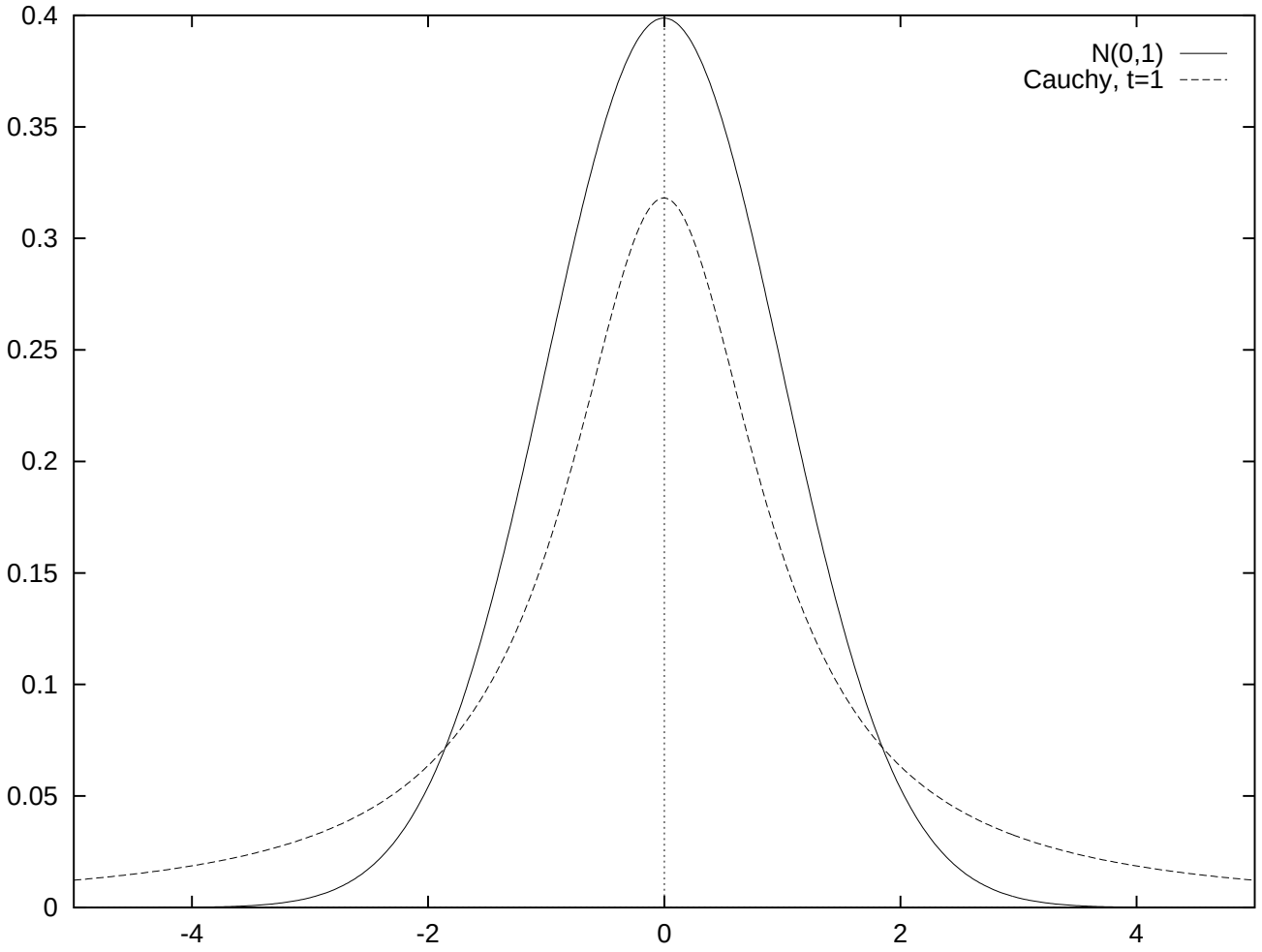
Its density function is

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

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



Why Cauchy Mutation Performed Better

Given $G(0,1)$ and $C(1)$, the expected length of one step 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.

Ah ha! Gaussian mutation jumps while Cauchy mutation flies.

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

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

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.

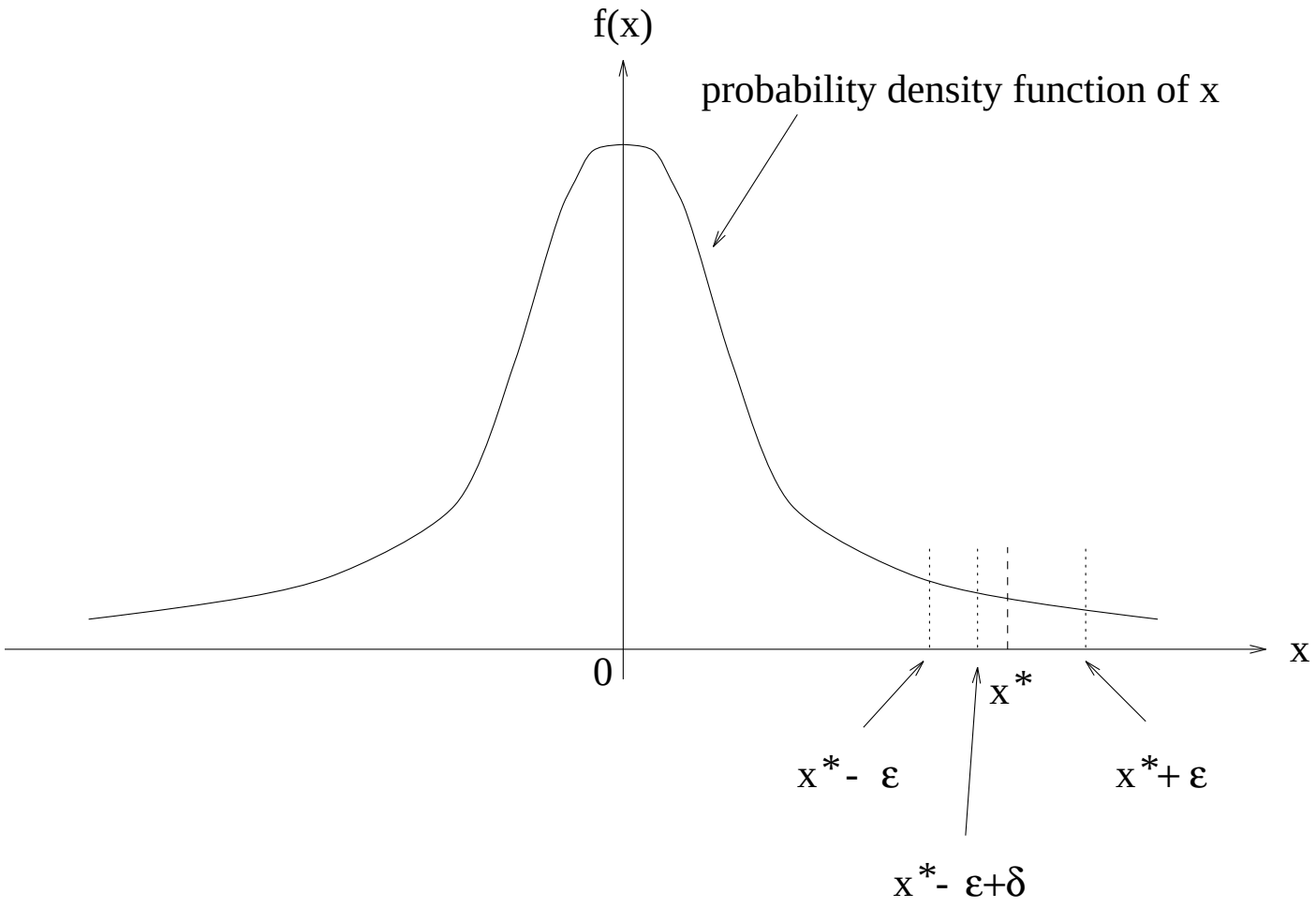


Figure 4: Evolutionary search as neighbourhood search, where x^* is the global optimum and $\epsilon > 0$ is the neighbourhood size.

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 1: 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 2: 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)$_{normal}$.

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.

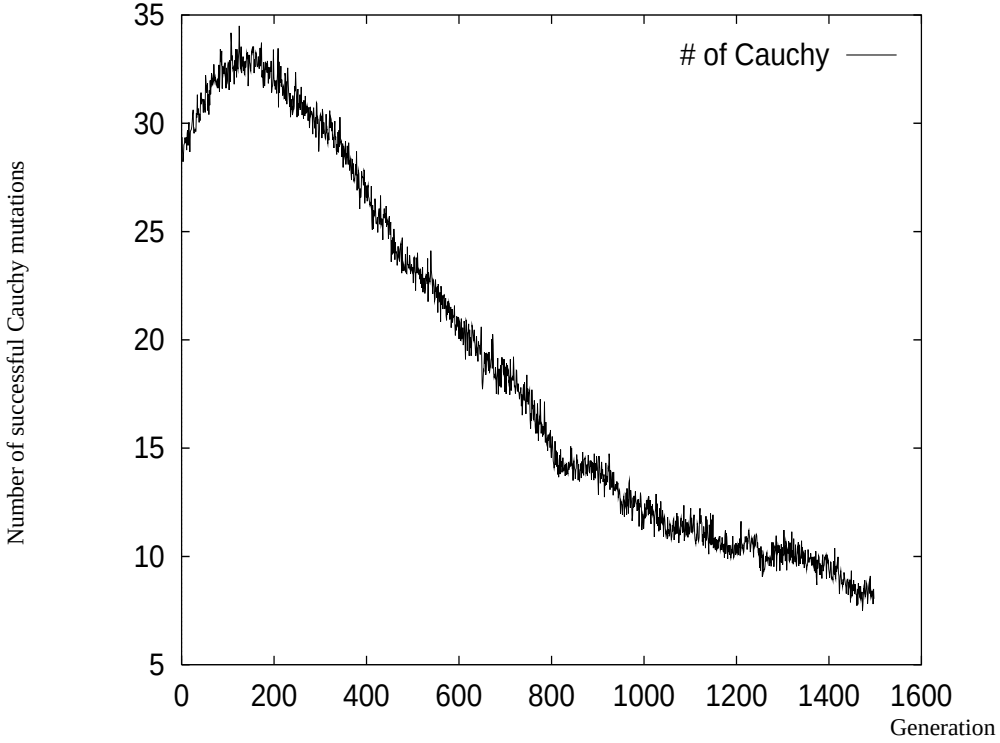


Figure 5: Number of successful Cauchy mutations in a population when IFEP is applied to function f_{10}. The vertical axis indicates the number of successful Cauchy mutations in a population and the horizontal axis indicates the number of generations. The results have been averaged over 50 runs.

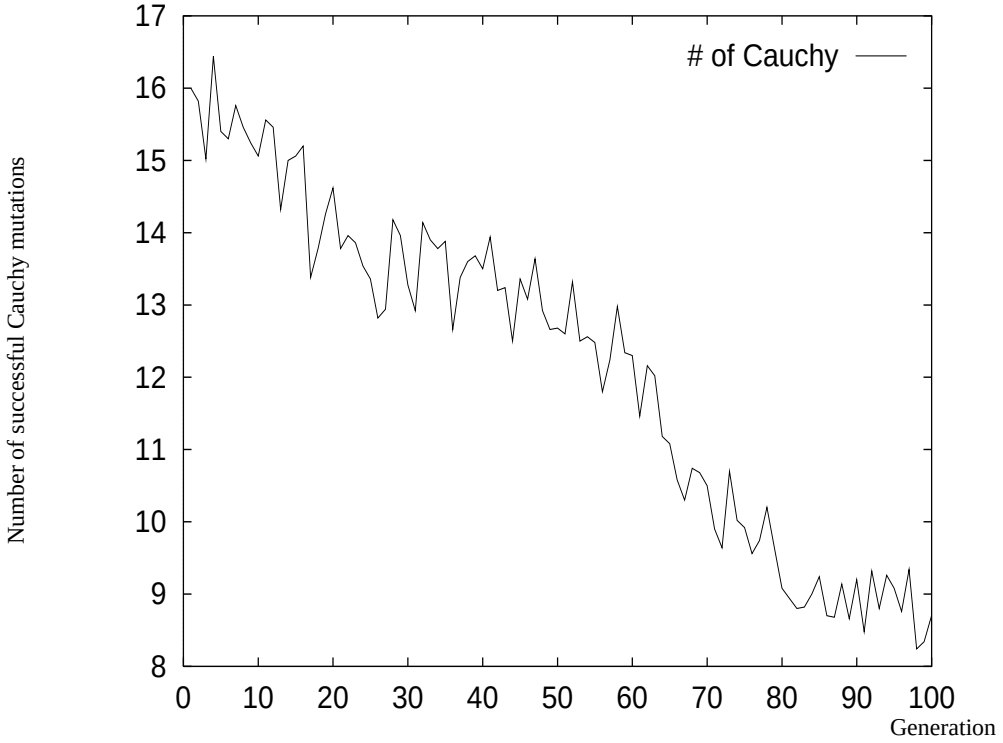


Figure 6: Number of successful Cauchy mutations in a population when IFEP is applied to function f_{21}. The vertical axis indicates the number of successful Cauchy mutations in a population and the horizontal axis indicates the number of generations. The results have been averaged over 50 runs.

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.

Lesson I

1. Simplicity does not mean ineffectiveness.
2. Insight is very useful in guiding the development of new algorithms.

Reference

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

An Anomaly of Self-adaptation in EP

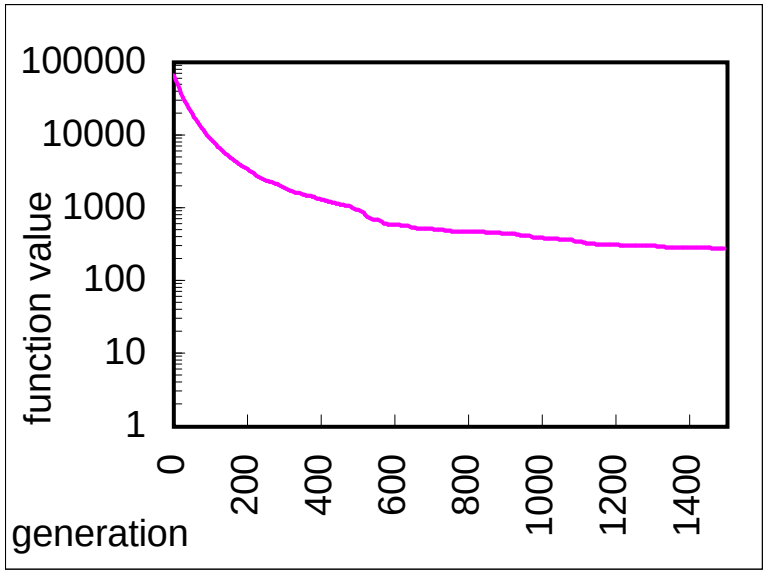


Figure 7: The 30-d sphere model stagnates early from mean of 50 runs.

Why EP Stagnates Early

Table 3: 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!

- A fixed lower bound, e.g., 10^{-3}.
- A dynamic lower bound, e.g., using the median of the mutation step sizes from all accepted (successful) offspring as the new lower bound for the next generation.

Reference:

1. 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.

Representation Is Important

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

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.

Reference:

1. 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.

Lesson II

1. Always read papers with questions in mind.
2. Don't always follow the trend. Stepping back is often very useful.

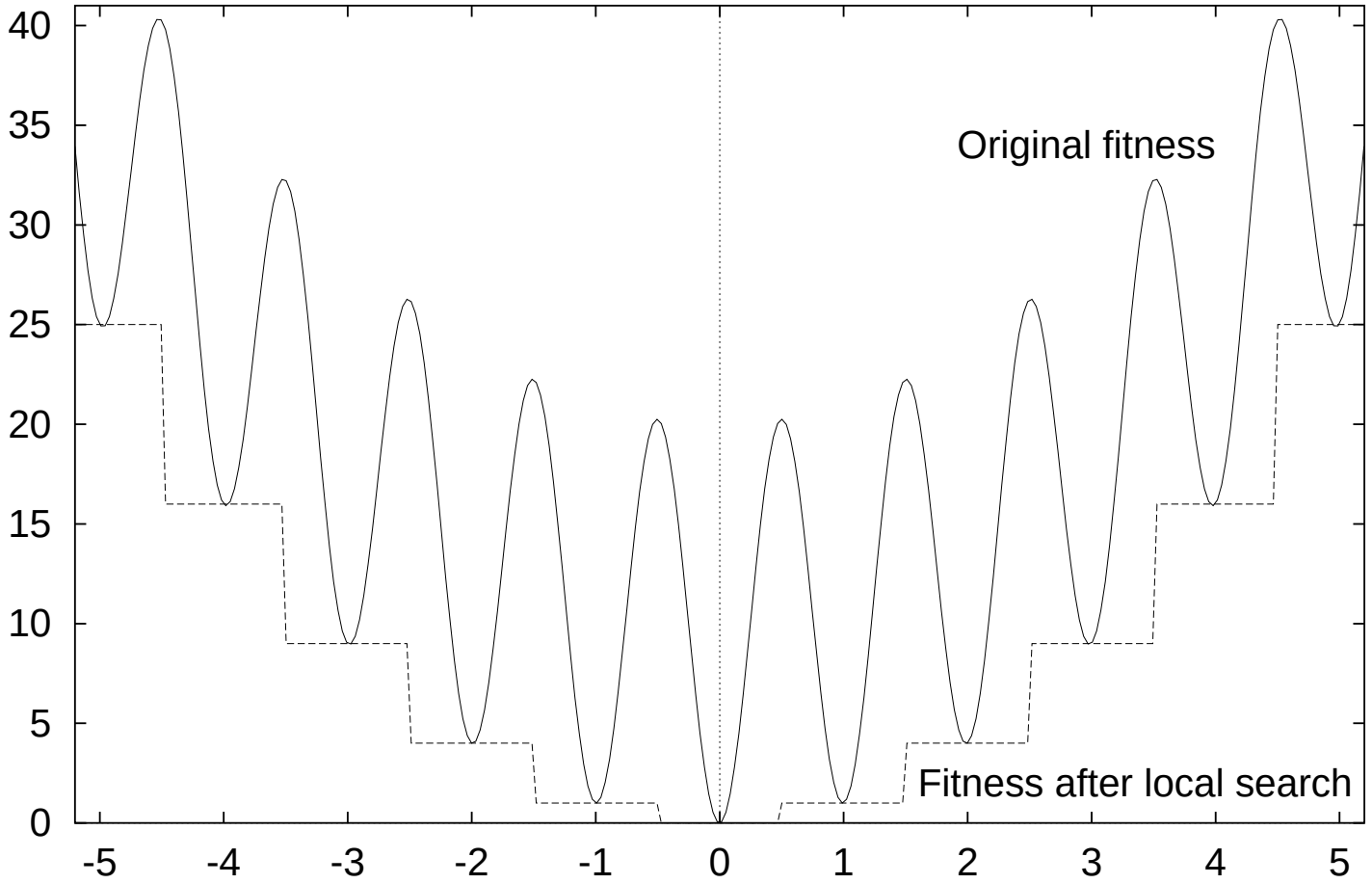
Are You Tired?

I am. I wish the problems can be simpler so I don't have to work so hard!

Stepping Back Again: Why Always Operators

1. A lot of efforts have been spent on improving evolutionary algorithms, e.g., fancy search operators. However, the "return on investment" is not always great.
2. Is there an easier way of solving a complex problem? (If only all problems are smooth and unimodal!)
3. Well, how about transforming a complex problem into a simpler one?

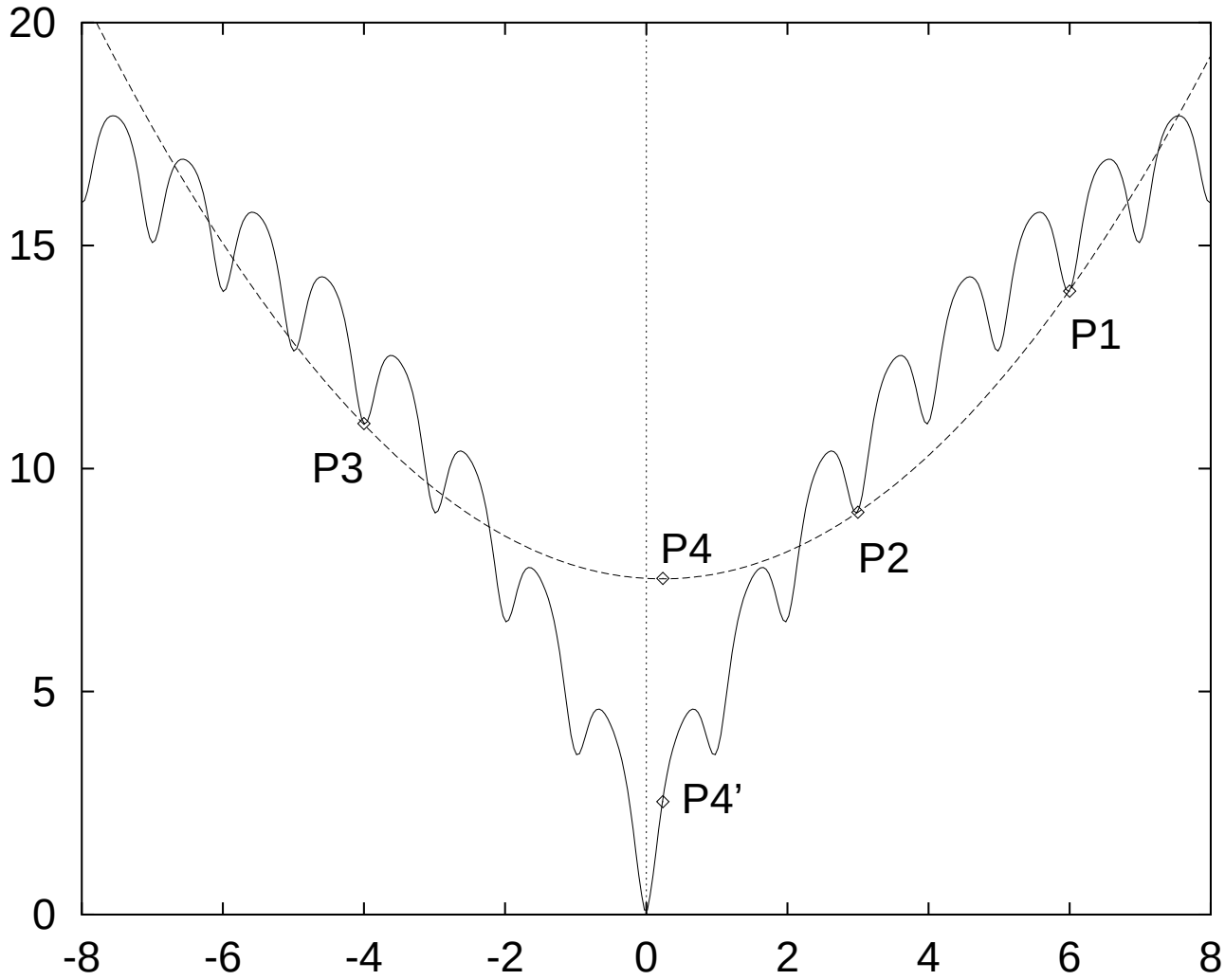
An Observation



Idea: Problem Approximation + Evolutionary Search

1. Initialize μ individuals at random.
2. Perform local search from each individual.
3. REPEAT
   (a) Generate 3 points P_1, P_2, P_3 by global discrete recombination.
   (b) Perform a quadratic approximation using P_1, P_2, P_3 to produce a point P_4.
   (c) Perform a local search from P_4 and update P_4 with the search result.
   (d) Place P_1, P_2, P_3, P_4 into the population and do a $(\mu + 4)$ truncation selection.
4. UNTIL termination criteria are met.

An Ideal Example



Global Discrete Recombination

- Each component of the offspring is copied from the corresponding component of a random individual in the population.
- The number of parents is the same as the population size.
- Many offspring may be generated. We only need three in our algorithm.

Quadratic Approximation

Let $x_{i,j}$ be the j-th component of the vectors x_i, $\forall i \in \{1, 2, 3\}$, $j \in \{1, \cdots, n\}$, where n is the dimensionality. We approximate the position of P_4 using the quadratic interpolation method as follow.

$$x_{4,j} = \frac{1}{2} \cdot \frac{(x_{2,j}^2 - x_{3,j}^2) f(x_1) + (x_{3,j}^2 - x_{1,j}^2) f(x_2) + (x_{1,j}^2 - x_{2,j}^2) f(x_3)}{(x_{2,j} - x_{3,j}) f(x_1) + (x_{3,j} - x_{1,j}) f(x_2) + (x_{1,j} - x_{2,j}) f(x_3)}$$

Local Search With Random Memorising

- Store best solutions in a memory.
- Retrieve a random one (old best) when a new best solution is found.
- Search along the direction of old$\longrightarrow$new, i.e., the direction of

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

Experimental Studies

- 18 multimodal benchmark functions.
- Population size 30
- Maximum function evaluation $500,000$.
- 50 independent runs for each function.

18 Benchmark Functions ($f_8 - f_{15}$)

Test function	n	S	f_{min}
$f_8(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	30	$[-500, 500]^n$	-12569.
$f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10)]$	30	$[-5.12, 5.12]^n$	0
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right)$ $+20 + e$	30	$[-32, 32]^n$	0
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$	0
$f_{12}(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})]\right.$ $\left.+(y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$	30	$[-50, 50]^n$	0
$f_{13}(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})]\right.$ $\left.+(x_n - 1)^2[1 + \sin^2(2\pi x_n)]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$	0
$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	2	$[-65.536, 65.536]^n$	1
$f_{15}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	$[-5, 5]^n$	0.00030

18 Benchmark Functions ($f_{16} - f_{25}$)

Test function	n	S	f_{min}
$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5,5]^n$	-1.0316285
$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2	$[-5,10] \times [0,15]$	0.398
$f_{18}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2,2]^n$	3
$f_{19}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{4} a_{ij}(x_j - p_{ij})^2\right]$	4	$[0,1]^n$	-3.86
$f_{20}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right]$	6	$[0,1]^n$	-3.32
$f_{21}(x) = -\sum_{i=1}^{5} [(x - a_i)^T(x - a_i) + c_i]^{-1}$	4	$[0,10]^n$	$-1/c_1$
$f_{22}(x) = -\sum_{i=1}^{7} [(x - a_i)^T(x - a_i) + c_i]^{-1}$	4	$[0,10]^n$	$-1/c_1$
$f_{23}(x) = -\sum_{i=1}^{10} [(x - a_i)^T(x - a_i) + c_i]^{-1}$ where $c_1 = 0.1$	4	$[0,10]^n$	$-1/c_1$
$f_{24}(x) = 0.5 + \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{[1.0 + 0.001(x_1^2 + x_2^2)]^2}$	2	$[-100,100]^n$	0
$f_{25}(x) = (x_1^2 + x_2^2)^{0.25}[\sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1.0]$	2	$[-100,100]^n$	0

Results on f_8–f_{13}

Prob	IFEP					LALS			
	Eval.	Mean	StdDev	Found	G. hit	Eval.	Mean	StdDev	G. h
8	900000	-10640.18	431.21	7575	0/50	199244	-11834.65	298.78	1/5
9	500000	2.89	1.98	5000	1/50	306280	2.67	1.58	5/5
10	150000	6.33e-4	3.86e-4	1500	50/50	370686	2.08e-12	2.11e-12	50/5
11	200000	1.27e-1	1.90e-1	1765	3/50	173592	1.87e-10	1.31e-9	50/5
12	150000	2.49e-2	5.70e-2	1500	40/50	476245	5.84e-9	2.16e-8	50/5
13	300000	4.42e-8	2.65e-8	1500	50/50	504212	1.19e-2	3.01e-2	33/5

Results on f_8–f_{13} With Population Size 50 and 60

Prob.	Pop. size	Eval.	Mean	Std Dev	Global hit
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

Results on f_{14}–f_{25}

Prob	IFEP					LALS			
	Eval.	Mean	StdDev	Found	G. hit	Eval.	Mean	StdDev	G. h
14	10000	1.56	0.96	10000	33/50	3052	1.04	0.20	48/
15	400000	4.17e-4	2.98e-4	235500	44/50	111748	3.0749e-4	2.45e-10	50/
16	10000	-1.031628	2.70e-8	3500	50/50	2817	-1.031628	2.70e-8	50/
17	10000	0.3979	8.26e-9	4500	50/50	5496	0.3979	8.26e-9	50/
18	10000	3	0	4500	50/50	4676	3	0	50/
19	10000	-3.86	0	6000	50/50	6852	-3.86	0	50/
20	20000	-3.26	5.90e-2	17500	24/50	17504	-3.32	2.35e-7	50/
21	10000	-7.02	2.74	10000	21/50	13790	-10.1532	0	50/
22	10000	-8.42	2.96	9500	34/50	13354	-10.4029	2.16e-7	50/
23	10000	-9.09	2.87	7500	39/50	14312	-10.5364	4.58e-7	50/
24	10000	8.86e-3	2.67e-3	8500	4/50	10754	3.89e-4	1.90e-3	48/
25	10000	2.47e-3	1.27e-3	10000	50/50	15614	1.07e-4	7.51e-4	50/

Results on f_{16}–f_{23} With Population Size 10

Prob.	Pop. size	Eval.	Mean	Std Dev	Global hit
16	10	1063	-1.031628	2.70e-8	50/50
17	10	2278	0.3979	8.26e-9	50/50
18	10	1665	3	0	50/50
19	10	2458	-3.86	0	50/50
20	10	6576	-3.32	5.67e-7	50/50
21	10	4990	-10.1532	1.29e-6	50/50
22	10	5455	-10.4029	2.16e-7	50/50
23	10	5554	-10.5364	4.58e-7	50/50

A General Framework

Step 1: Generate μ individuals and their local search information.

Step 2: Obtain λ individuals by global discrete recombination, and generate ρ predicted minimum points using approximation techniques.

Step 3: Obtain local search information for ρ individuals.

Step 4: Select next generation from μ, λ, and ρ individuals.

Step 5: Go to Step 2 if termination criteria are not met.

Another Algorithm on Approximation + Search

1. A simplified quadratic polynomial approximation.
2. Simplified local search for the initial population.

Results on f_8–f_{13},f_{26},f_{27}

P	ES			IFEP			EANA		
	Eval.	Mean	G. hit	Eval.	Mean	G. hit	Eval.	Mean	G. hi
8	241500	-10091.26	0/50	757500	-10640.18	0/50	**102470**	**-1219 1.21**	**0/5**
9	500000	344.44	0/50	500000	2.89	1/50	**83483**	**1.99 e-2**	**49/5**
10	**100000**	**2.66e-10**	**50/50**	150000	6.33e-4	50/50	18 1578	3.30e-5	50/5
11	100000	0	50/50	176500	1.27e-1	3/50	**9372**	**2.05 e-10**	**50/5**
12	**50000**	**1.06e-9**	**50/50**	476245	2.49e-2	40/50	18 5318	8.14e-11	50/5
13	50000	4.40e-4	48/50	**150000**	**4.42e-8**	**50/50**	34 9059	1.88e-9	50/5
26	200000	5223.03	0/50	200000	23438.64	0/50	1139316	222.81	0/5
27	200000	-9.20e-3	N/A	200000	-8.55e-2	N/A	**49839**	**-0.24 325**	**N/A**

Results on f_{14}–f_{25}

Prob	ES			IFEP			EANA		
	Eval.	Mean	G. hit	Eval.	Mean	G. hit	Eval.	Mean	G. hi
14	**1000**	**4.18**	**33/50**	10000	1.56	33/50	1685	7.36	4/50
15	400000	3.92e-3	44/50	235500	4.17e-4	44/50	**31645**	**3.07e- 4**	**50/50**
16	2500	-1.031628	50/50	3500	-1.031628	50/50	**863**	**-1.0316 28**	**50/50**
17	3000	0.3979	50/50	4500	0.3979	50/50	**945**	**0.3979**	**50/50**
18	3500	3	50/50	4500	3	50/50	**822**	**3**	**50/50**
19	4000	-3.86	50/50	6000	-3.86	50/50	**1297**	**-3.86**	**50/50**
20	10500	-3.31	43/50	17500	-3.26	24/50	**8357**	**-3.3128**	**46/50**
21	5000	-6.71	24/50	10000	-7.02	21/50	**2958**	**-9.6512**	**46/50**
22	5000	-9.14	41/50	9500	-8.42	34/50	**2880**	**-9.1430**	**41/50**
23	5000	-9.32	42/50	7500	-9.09	39/50	**3118**	**-9.5767**	**43/50**
24	4000	1.81e-2	1/50	8500	8.86e-3	4/50	**1181**	**2.97e- 2**	**30/50**
25	10000	1.42e-1	48/50	**10000**	**2.47e-3**	**50/50**	1687	3.19e-1	44/50

Lesson III

1. If the Mountain doesn't come to Mohammed, Mohammed must go to the Mountain.
2. There is no restriction on the approximation or local search algorithm used.
3. Other ways of improving "return on investment"?

Reference:

1. K.-H. Liang, X. Yao and C. Newton, "Evolutionary search of approximated N-dimensional landscapes," *International Journal of Knowledge-Based Intelligent Engineering Systems*, **4**(3):172-183, July 2000.

Scaling Up Evolutionary Algorithms

- Most reported studies on EAs are obtained using low-dimensional problems, e.g., smaller than 100 dimensions.
- Most EAs suffer from

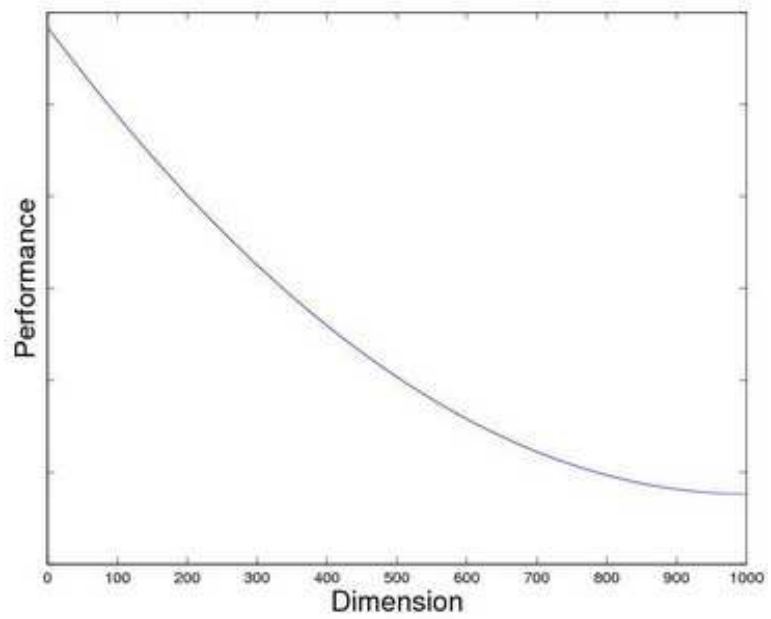


Figure 8: EAs' performance deteriorates with the growth of dimension.

New Old Idea: Divide-and-Conquer

- Problem decomposition
- Subcomponent optimization
- Cooperative combination

Crucial issues:

- How to execute problem decomposition?
- How to execute subcomponents combination?

Simplest Method — One Dimension Per Population

The following one-dimensional base methods have been applied in GA and FEP to produce corresponding CCGA and FEPCC [a]:

- Step 0: Assign a population to each variable.
- Step 1: $j = 1$. Apply optimizer to the population of variable 1.
- Step 2: $j = j + 1$. Apply optimizer to the population of variable j.
- Step 3: if $j < n$. go to Step 2. Otherwise, go to Step 1 for the next cycle.

---

[a] Y. Liu, X. Yao, Q. Zhao and T. Higuchi, "Scaling Up Fast Evolutionary Porgramming with Cooperative Coevolution," *Proc. of the Congress on Evolutionary Computation*, pp. 1101–1108, 2001.

Discussions

- The one-dimensional based methods are very efficient and effective on separable problems.
- However, they are not applicable on nonseparable problems, in which interactions existed between variables.
- Interactions usually exist in some subsets of all the variables.
- So, is it possible to split the variables into several independent **groups**, and interactions only exist between variables within the same group?

Random-grouping Based Methods I: DECC-I

Split the objective vector into several groups randomly, and evolve a group of variables separately[a]:

1. Decomposition: Group the n-dimensional objective vector into m s-dimensional subcomponents randomly, where we assume $n = m * s$. Here "randomly" means each variable has the same chance to be assigned into any of the groups.
2. Cooperation: Apply a weight to each of the groups after each complete circle; Evolve the weight vectors for the best, the worst and a random member in the current population.

Implemented with a DE (Differential Evolution) variant: **DECC-I**.

[a] Zhenyu Yang, Ke Tang and Xin Yao, "Differential Evolution for High-Dimensional Function Optimization," *Proc. of the Congress on Evolutionary Computation*, Singapore, 2007, pp.3523-3530.

Random-grouping Based Methods II: DECC-II

Select a subset of the objective variables randomly, and evolve the subset with a basic optimizer:

1. Select a subset with s variables randomly from the object vector to form an s-dimensional subcomponent.
2. Optimize the subset with the optimizer for one cycle, while keeping the other variables constant.
3. Stop if the halting criterion is satisfied; otherwise go to step 1 for the next cycle.

Implemented with a DE variant: **DECC-II**.

Random-grouping with Dynamic Structure: DECC-G

Advanced version of DECC-I by changing the decomposition structure dynamically at the start of each cycle[a]:

1. Set $i = 1$ to start a new *cycle*.
2. Split the n-dimensional object vector into m subcomponents (s-dimensional) randomly, assuming $n = m * s$.
3. Optimize the i-th subcomponent with a certain EA.
4. If $i < m$ then $i++$, and go to step 3.
5. Apply a weight to each of the subcomponents; Evolve the weight vectors for the best, the worst and a random members of current population.
6. Stop if halting criterion is satisfied; otherwise go to step 1.

Implemented with a DE variant: **DECC-G**.

[a] Zhenyu Yang, Ke Tang and Xin Yao, "Cooperative Coevolution for High-dimensional Optimization Problems," *Accepted by Information Sciences*, 2008.

Opportunities for Future Work

- We have chosen the most recent DE variant **SaNSDE** to implement our decomposition methods within the cooperative coevolution framework. And thus we have proposed algorithms DECC-I, DECC-II and DECC-G.
- But, any other DE variant and even other EAs can be introduced easily to produce corresponding **EACC-?**.
- Moreover, even more than one kind of EAs can be employed as subcomponent optimizer to blend different search biases.

Why and How Well EACC-G Works

The probability of EACC-G to assign two interacting variables x_i and x_j into a single subcomponent for at least k cycles is:

$$P_k = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1 - \frac{1}{m}\right)^{N-r} \tag{6}$$

where N is the number of cycles, and m is the number of subcomponents.

Given $n = 1000$, $s = 100$, $m = n/s = 10$, $N = 50$, we can achieve:

$$P_1 = 1 - p_0 = 0.9948, \quad P_2 = 1 - p_0 - p_1 = 0.9662$$

The probability is very high.

Experimental Studies: Setup

We have chosen function optimization as the test bed. The proposed algorithms were evaluated on a suite of benchmark functions provided by *CEC2005 special session*.

- Select functions: f_1, f_3, f_5, f_6, f_8, f_9, f_{10} and f_{13}.
- f_1 and f_9 are separable functions, on which one-dimensional based decomposition methods may work very well.
- All other functions are nonseparable, which can be used to evaluate the effectiveness of grouping-based methods.
- Details of these functions can be found from the web.

More Setup

- The algorithms to be compared are:
  - **DECC-I**
  - **DECC-II**
  - **DECC-G**
  - **DE:** The original DE given by Storn and Price.
  - **SaNSDE:** A most recent DE variant.
  - **DECC-O:** The algorithm which adopts the one-dimensional based decomposition methods.
- The same numbers of fitness evaluations (FEs) were used for all algorithms, and they were set to grow in the order of $O(n)$.
- For evolutionary process curves, the horizontal axis is the number of generations, and the vertical axis is the function value(the lower, the better).

Experimental Results I

DECC-I/DECC-II Performance on CEC'2005 Benchmark Functions:

:	# of Dim's	DE Mean	DECC-I Mean	DECC-II Mean
	500	1.56e+06	**2.18e-13**	4.07e-13
	1000	3.36e+06	**4.91e-13**	6.64e-13
	500	9.25e+10	**1.75e+08**	2.29e+08
	1000	2.20e+11	**3.85e+08**	4.10e+08
	500	2.39e+05	**9.65e+04**	1.75e+05
	1000	3.94e+05	**1.98e+05**	2.40e+05

Experimental Results II

DECC-I/DECC-II Performance on CEC'2005 Benchmark Functions:

	# of Dim's	DE Mean	DECC-I Mean	DECC-II Mean
	500	1.68e+12	**4.91e+02**	8.82e+02
	1000	1.68e+12	**9.86e+02**	2.25e+03
	500	2.16e+01	2.16e+01	2.16e+01
	1000	2.16e+01	2.16e+01	2.16e+01
	500	8.85e+03	**2.22e+03**	2.91e+03
	1000	1.81e+04	**4.59e+03**	5.22e+03
	500	1.38e+04	**2.58e+03**	5.88e+03
	1000	2.93e+04	**6.85e+03**	9.40e+03
	500	7.90e+02	**2.14e+02**	2.94e+02
	1000	2.08e+03	**4.14e+02**	5.32e+02

Comparisons

ween DECC-G and DE, SaNSDE, DECC-O on CEC2005 functions, with
500 and 1000. All results have been averaged over 10 independent runs.

of im's	DE Mean	SaNSDE Mean	DECC-O Mean	DECC-G Mean	DECC-G/O t-test
500	1.56e+06	**5.68e-14**	1.76e-12	2.18e-13	-7.76e+01†
000	3.36e+06	4.84e-05	3.52e-12	**4.91e-13**	-8.10e+01†
500	9.25e+10	**1.58e+08**	2.21e+08	1.75e+08	-3.13e+00†
000	2.20e+11	4.75e+08	4.76e+08	**3.85e+08**	-7.54e+00†
500	2.39e+05	1.21e+05	2.33e+05	**9.65e+04**	-4.49e+01†
000	3.94e+05	2.51e+05	3.85e+05	**1.98e+05**	-3.64e+01†
500	1.68e+12	1.29e+03	9.81e+02	**4.91e+02**	-3.99e+00†
000	1.68e+12	2.75e+03	1.73e+03	**9.86e+02**	-8.61e+00†
500	2.16e+01	2.15e+01	**2.12e+01**	2.16e+01	8.28e+01†
000	2.16e+01	2.16e+01	**2.12e+01**	2.16e+01	8.84e+02†
500	8.85e+03	3.93e+02	**2.91e+00**	2.13e+02	3.98e+01†
000	1.81e+04	1.89e+03	**6.17e+00**	4.66e+02	1.13e+02†
500	1.38e+04	5.95e+03	1.60e+04	**2.58e+03**	-4.51e+01†
000	2.93e+04	1.50e+04	3.13e+04	**6.85e+03**	-4.45e+01†
500	7.90e+02	3.03e+02	**2.53e+01**	2.14e+02	7.35e+01†
000	2.08e+03	7.93e+02	**5.33e+01**	4.14e+02	9.69e+01†

Lesson IV

There are many interesting ideas in evolutionary computation, which can be and should be studied more. They can sometime bring good results.

Constraint Handling

The general nonlinear programming problem (A) can be formulated as solving the *objective function*

$$\text{minimize} \quad f(\vec{x}), \quad \vec{x} = (x_1, \ldots, x_n) \in \mathcal{R}^n \tag{7}$$

where $\vec{x} \in \mathcal{S} \cap \mathcal{F}$, $\mathcal{S} \subseteq \mathcal{R}^n$ defines the *search space* which is a n-dimensional space bounded by the *parametric constraints*

$$\underline{x}_i \leq x_i \leq \overline{x}_i, \quad 1 \leq i \leq n, \tag{8}$$

and the *feasible region* $\mathcal{F}$ is defined by

$$\mathcal{F} = \{\vec{x} \in \mathcal{R}^n \mid \mathrm{g_j}(\vec{\mathrm{x}}) \leq 0 \ \forall \ \mathrm{j} \in \{1, \ldots, \mathrm{m}\}\}, \tag{9}$$

where $\mathrm{g_j}(\vec{\mathrm{x}}), \mathrm{j} \in \{1, \ldots, \mathrm{m}\}$, are the constraints.

Penalty Function Approach

$$\psi(\vec{x}) = f(\vec{x}) + r_g \ \phi(\mathrm{g_j}(\vec{\mathrm{x}}); \mathrm{j} = 1, \ldots, \mathrm{m}) \tag{10}$$

where $\phi \geq 0$ is a real valued function which imposes a 'penalty' controlled by a sequence of *penalty coefficients* $\{r_g\}_0^G$, where g is the generation counter. The following quadratic function has often been used as the *penalty function*:

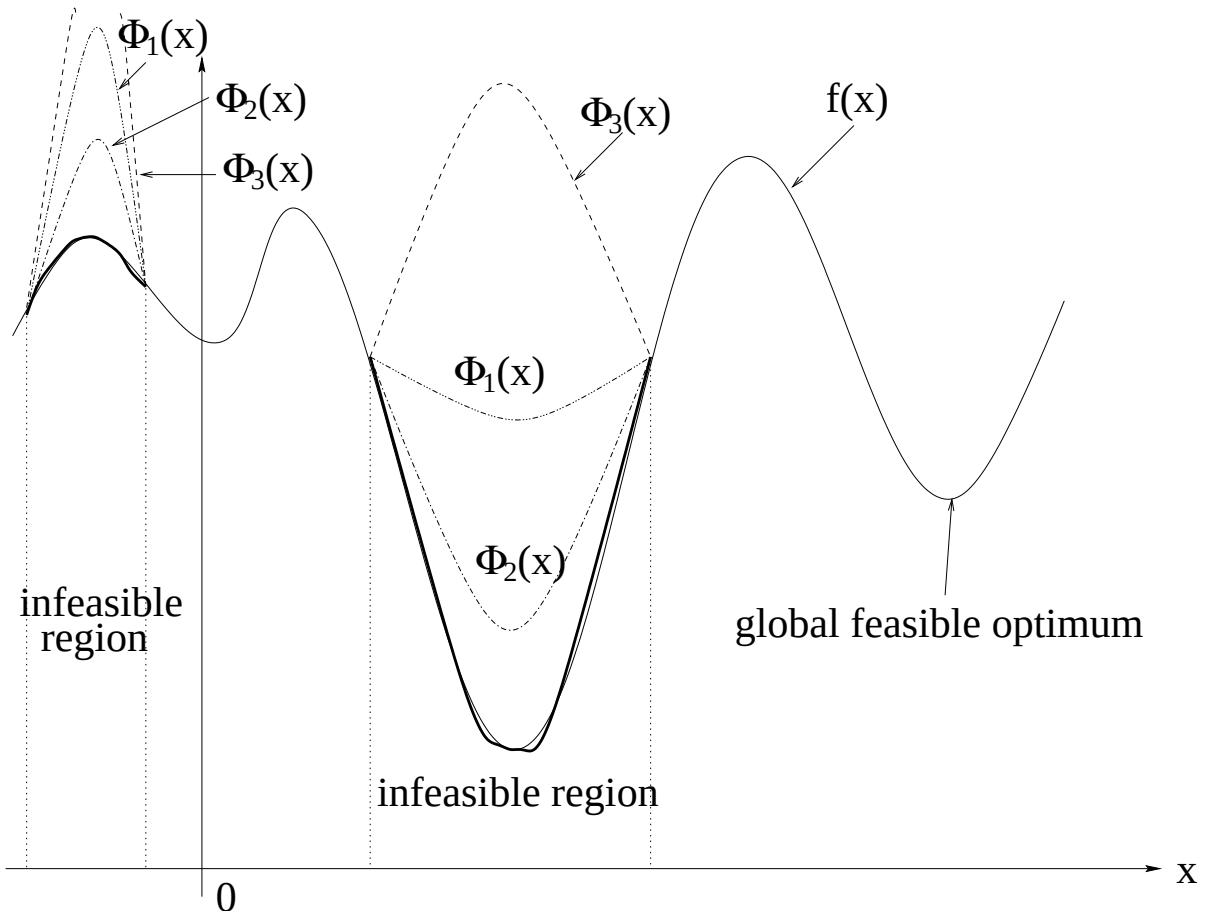
$$\phi(g_j(\vec{x}); j = 1, \ldots, m) = \sum_{j=1}^{m} \max\{0, \mathrm{g_j}(\vec{\mathrm{x}})\}^2. \tag{11}$$

although our proposed constraint handling technique is equally applicable to any other forms of penalty functions.

What Does a Penalty Method Really Do

1. All that a penalty method tries to do is to obtain the right balance between objective and penalty functions so that the search is moving towards the optimum in the feasible space, **not** just towards the optimum in the combined feasible and infeasible space.
2. A simple way to measure the balance of dominance of objective and penalty functions is to count how many comparisons of adjacent pairs are dominated by the objective and penalty function respectively.
3. One way to achieve such balancing effectively and efficiently is to adjust such balance directly and explicitly. This is what stochastic ranking does.

What Would be an Appropriate Penalty



The balance between objective function and penalties is very important.

Stochastic Ranking

1. Stochastic Ranking (SR) is very simple. It only changes the selection scheme into a ranking scheme in order to perform constrained optimisation. No special operators or representations are necessary.
2. We introduce a probability P_f of using only the objective function for comparisons in ranking.
3. SR is implemented through a random version of bubble-sort.

Implementation of Stochastic Ranking

```
1   I_j = j ∀ j ∈ {1, ..., λ}
2   for i = 1 to N do
3       for j = 1 to λ − 1 do
4           sample u ∈ U(0, 1)
5           if (φ(I_j) = φ(I_{j+1}) = 0 ) or (u < P_f) then
6              if (f(I_j) > f(I_{j+1})) then
7                 swap(I_j, I_{j+1})
8              fi
9           else
10             if (φ(I_j) > φ(I_{j+1})) then
11                swap(I_j, I_{j+1} )
12             fi
13          fi
14      od
15      if no swap done break fi
16  od
```

Experimental Studies

1. (30,200)-ES was used.
2. 13 benchmark functions were used.
3. Different P_f values were studied.

Experimental Results

Table 4: Comparison between our (indicated by **RY**) and Koziel and Michalewicz's (indicated by **KM** algorithms. The two values in the 'Mean' column for problem g13 represent medians.

		Best Result		Mean Result		Worst Result	
fcn	optimal	RY	KM	RY	KM	RY	KM
g01	−15.000	−15.000	−14.7864	−15.000	−14.7082	−15.000	−14.6154
g02	−0.803619	−0.803515	−0.79953	−0.781975	−0.79671	−0.726288	−0.79119
g03	−1.000	−1.000	−0.9997	−1.000	−0.9989	−1.000	−0.9978
g04	−30665.539	−30665.539	−30664.5	−30665.539	−30655.3	−30665.539	−30645.9
g05	5126.498	5126.497	—	5128.881	—	5142.472	—
g06	−6961.814	−6961.814	−6952.1	−6875.940	−6342.6	−6350.262	−5473.9
g07	24.306	24.307	24.620	24.374	24.826	24.642	25.069
g08	−0.095825	−0.095825	−0.0958250	−0.095825	−0.0891568	−0.095825	−0.0291438
g09	680.630	680.630	680.91	680.656	681.16	680.763	683.18
g10	7049.331	7054.316	7147.9	7559.192	8163.6	8835.655	9659.3
g11	0.750	0.750	0.75	0.750	0.75	0.750	0.75
g12	−1.000000	−1.000000	−0.999999857	−1.000000	−0.999134613	−1.000000	−0.991950498
g13	0.053950	0.053957	0.054	0.057006	0.064	0.216915	0.557

References (Downloadable)

http://www.cs.bham.ac.uk/~xin/

1. T. P. Runarsson and X. Yao, "Stochastic Ranking for Constrained Evolutionary Optimization," *IEEE Transactions on Evolutionary Computation*, **4**(3):284-294, September 2000.

2. T. Runarsson and X. Yao, "Search Bias in Constrained Evolutionary Optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, **35**(2):233-243, May 2005.

Concluding Remarks

1. Evolutionary computation is fun and rich for interesting ideas.
2. Simple is often beautiful.
3. Taking one step back can often lead to two (or more) steps forward.
4. You do not need to be a genius to do research in evolutionary computation.