Lecture 4 - Case Studies of Search Operators and Representation

Yuhui Shi

CSE, SUSTech

Review of Previous Lectures

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

Outline of This Lecture

- Self-adaptation and Parameter Control in Evolutionary Algorithms
  - ✓Global Optimisation by Mutation-Based EAs
  - ✓What Mutation and Self-Adaptation Do
  - ✓More about Self-adaptation and Parameter Control in EAs
- Representation is Important
- Summary

Outline of This Lecture

- **Self-adaptation and Parameter Control in Evolutionary Algorithms**
  - ✓Global Optimisation by Mutation-Based EAs
  - ✓What Do Mutation and Self-Adaptation Do
  - ✓More about Self-adaptation and Parameter Control in EAs
- Representation is Important
- Summary

Global Optimisation by Mutation-Based EAs I

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

   for $j = 1, \cdots, n$,

$$x'_i(j) = x_i(j) + N_j(0, \sigma^2)$$

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

Global Optimisation by Mutation-Based EAs II

4. Evaluate the fitness of each offspring.
5. Apply round robin tournament selection: For each individual, *q* opponents are chosen randomly from all the parents and offspring with an equal probability. For each comparison, if the individual's fitness is better than the opponent's, it receives a "win".
6. Select the μ best individuals (from 2μ) that have the most wins to be the next generation.
7. Stop if the stopping criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

Example Visualisation of *N(0, σ²)*

With a Normal distribution, the probability for a value within the range of 2σ around the mean is 0.9544。

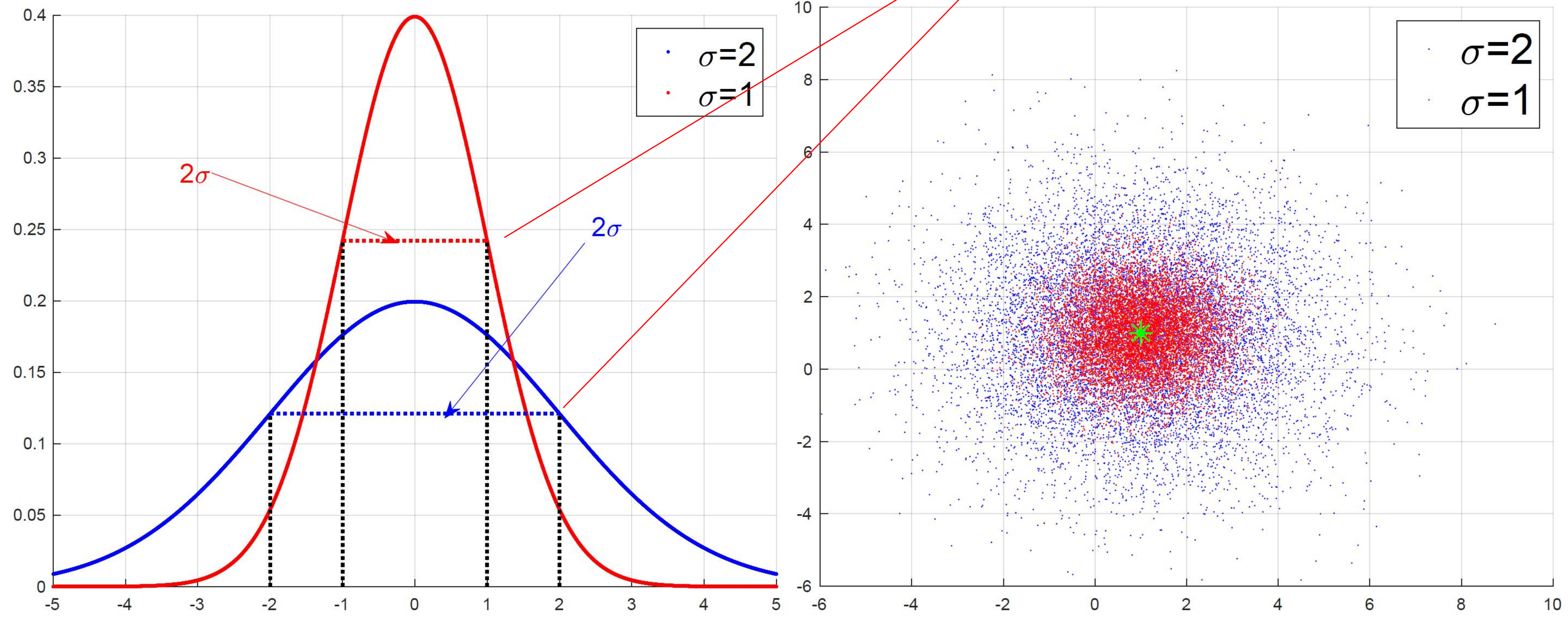


Figure 1: Left: Gaussian distributions $\mathcal{N}(0,1)$ and $\mathcal{N}(0,2)$.
Right: 10000 offspring generated using Eq. (1) with perturbation $\sim \mathcal{N}(0,1)$ and perturbation $\sim \mathcal{N}(0,2)$ around the parent $\mathbf{x}$.

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

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

Function Optimisation by Classical EP (CEP)

Each individual (x_i, η_i), $i = 1, \cdots, \mu$, creates a single offspring (x_i', η_i') by:
for $j = 1, \cdots, n$,

$$\begin{aligned} x_i'(j) &= x_i(j) + \eta_i(j)\mathcal{N}_j(0,1), & (2) \\ \eta_i'(j) &= \eta_i(j)\exp(\tau'\mathcal{N}(0,1) + \tau\mathcal{N}_j(0,1)) & (3) \end{aligned}$$

where

- $x_i(j)$, $x_i'(j)$, $\eta_i(j)$ and $\eta_i'(j)$ denote the j-th components 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 parameters τ and τ' have commonly been set to $\frac{1}{\sqrt{2\sqrt{n}}}$ *and* $\frac{1}{\sqrt{2n}}$.

Fast EP

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

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

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

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

Cauchy Distribution

Its density function is

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

t > 0 is a scale parameter.

The corresponding distribution function is

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

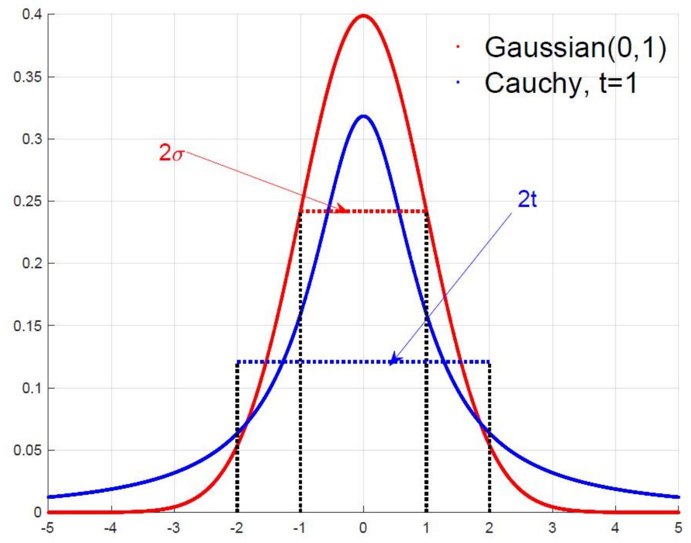


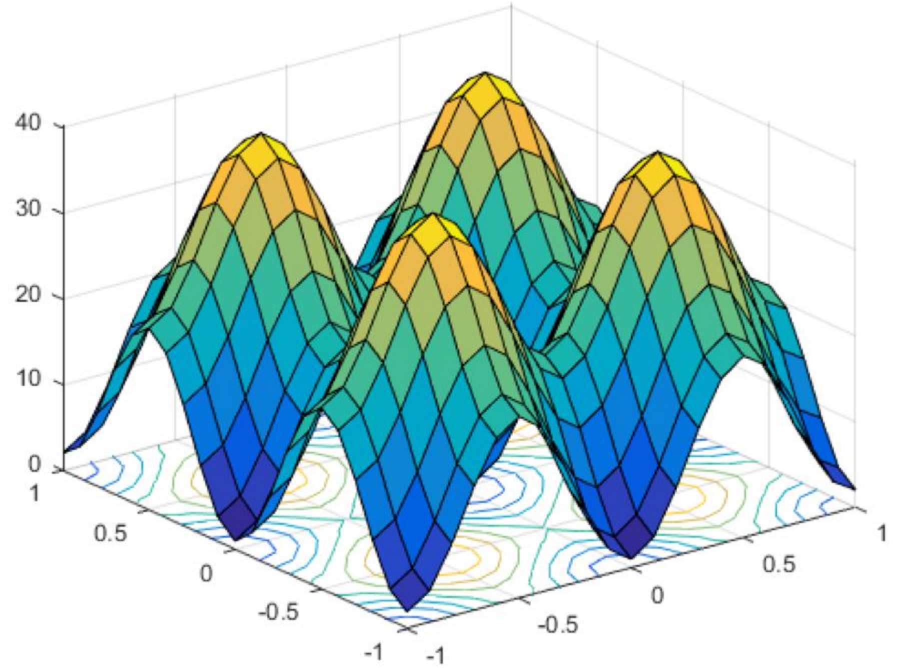
Figure 2: Gaussian and Cauchy density functions.

Benchmark Functions

- Benchmark functions:
  - ✓23 functions were used in the computational studies [1]. They have different characteristics.
  - ✓Some have a relatively high dimension (e.g. 30).
  - ✓Some have many local optima.
- Aim at minimising these functions here.

→ Smaller values are better.

Benchmark Functions at a Closer Look.



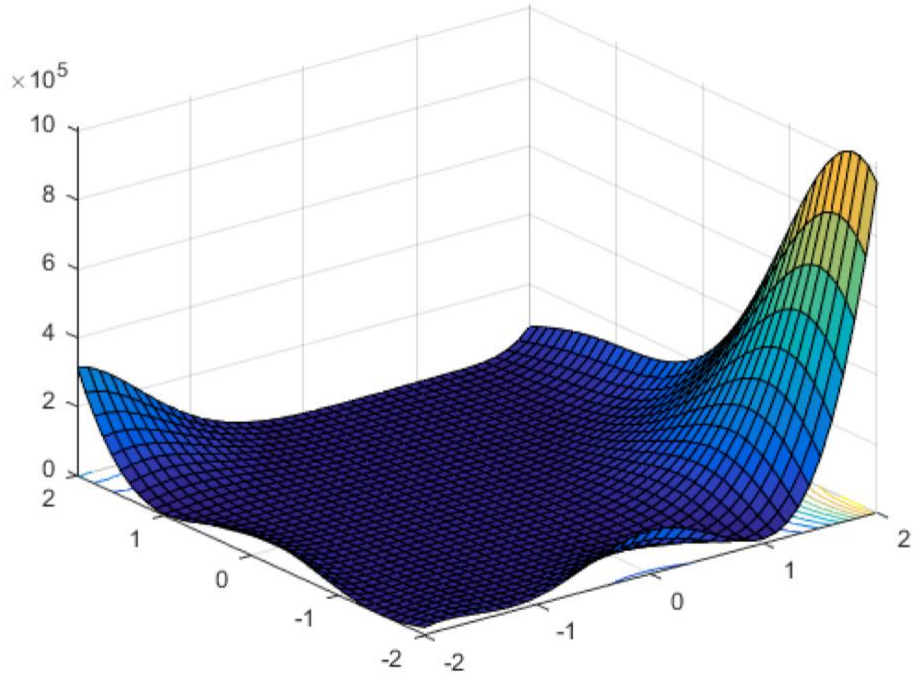


Figure 3: Functions f_9 (left) and f_{18} (right).

Experimental Setup

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

Questions

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

Experiments on Unimodal Functions f_1-f_7

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

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

Experiments on Unimodal Functions f_1-f_7
-*Visualisation of Some Functions*

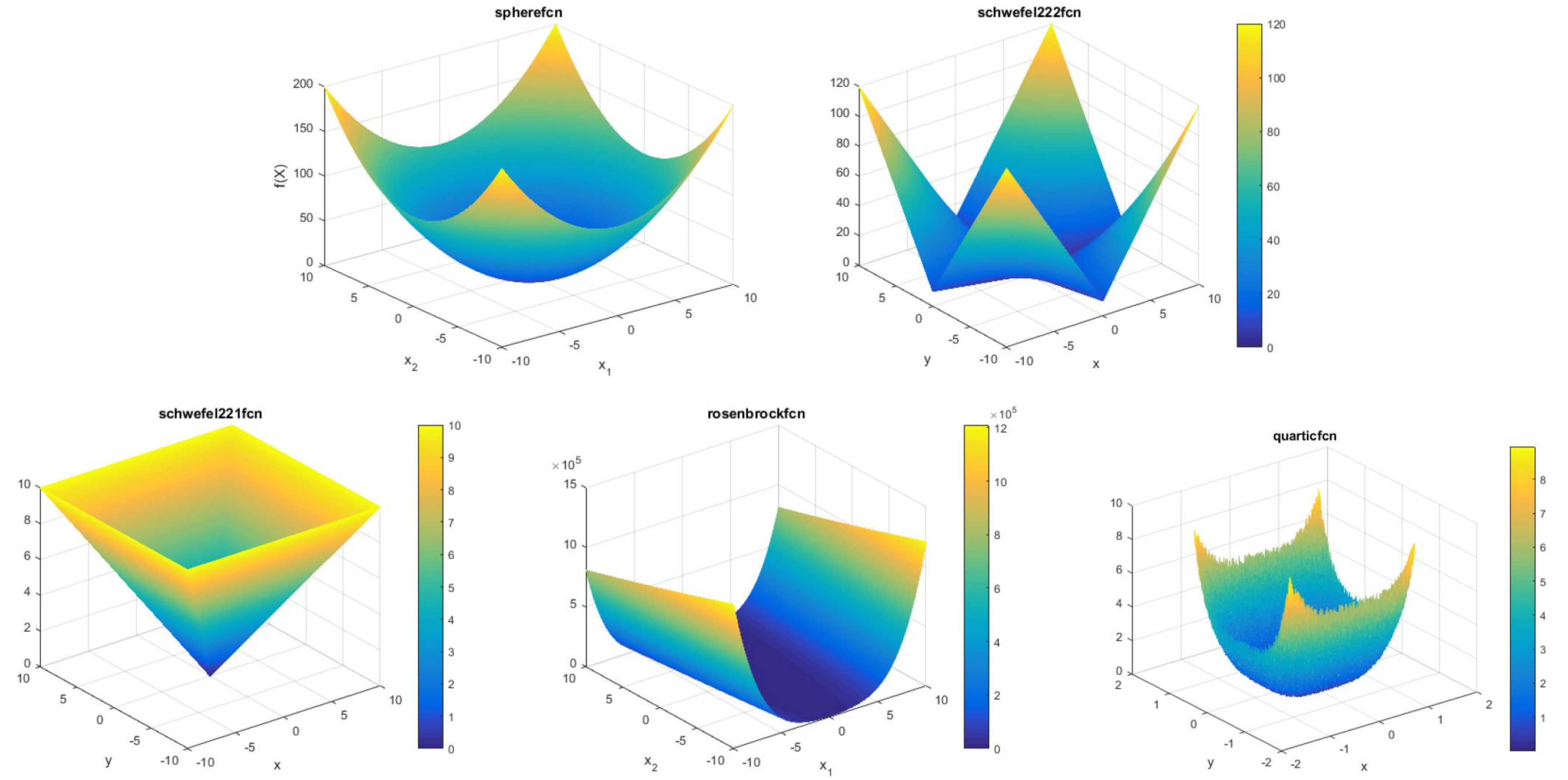


Figure 4: Functions f_1, f_2, f_4, f_5 and f_7. Sources: `http://H.Bbenchmarkfcns.H.Bxyz/H.BfcnsH.B`

Experiments on Unimodal Functions f_1-f_7 -*Numerical Results*

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

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

Observations:

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

Experiments on Unimodal Functions f_1-f_7 -*Evolutionary Curves for f_1 and f_2 and f_7*

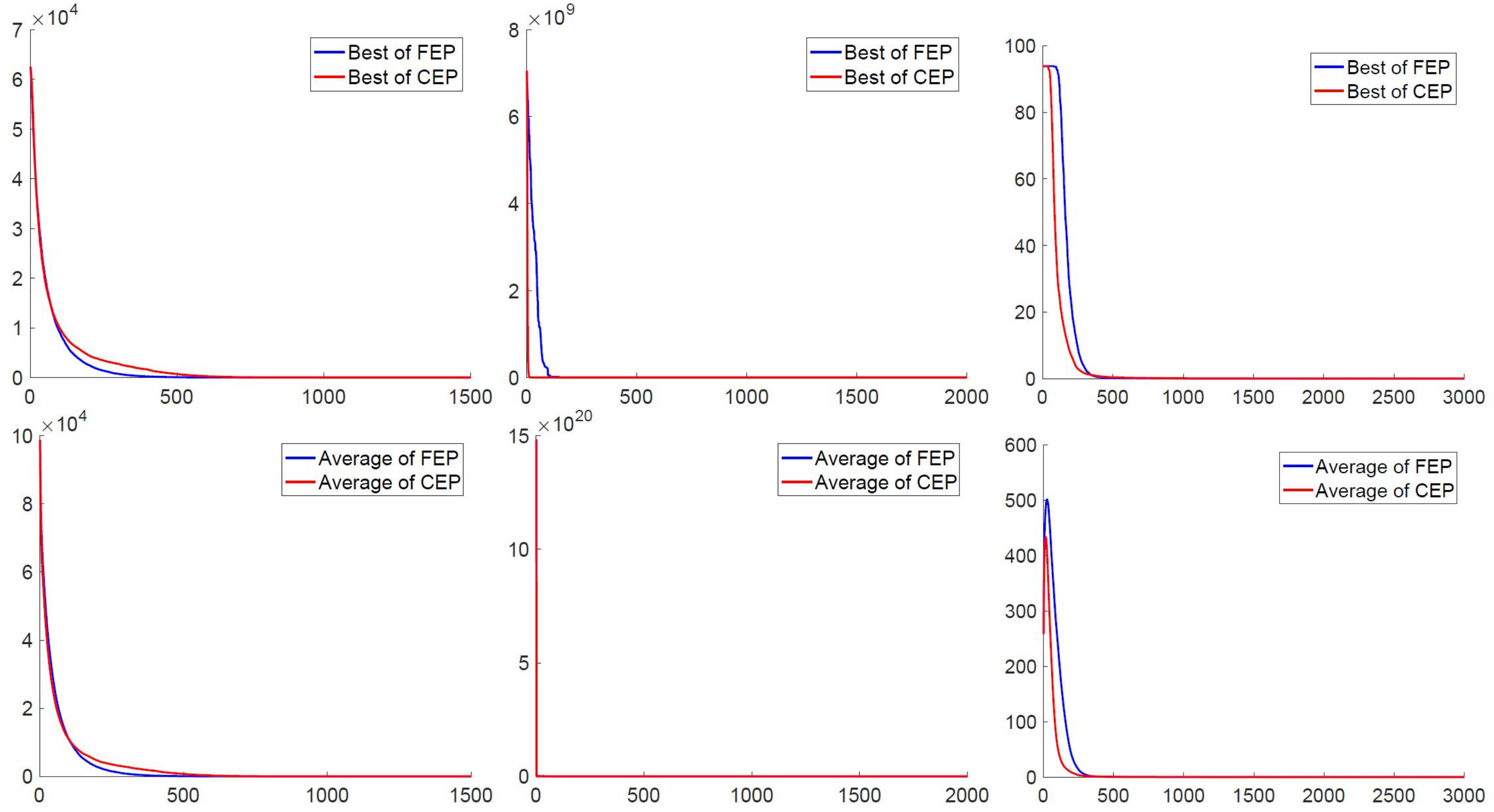


Figure 5: Evolutionary curves of CEP and FEP for f_1 (left), f_2 (middle) and f_7 (right). Plot from 1^{st} generation.

Experiments on Unimodal Functions f_1-f_7
-*Evolutionary Curves for f_1 and f_2 and f_7*

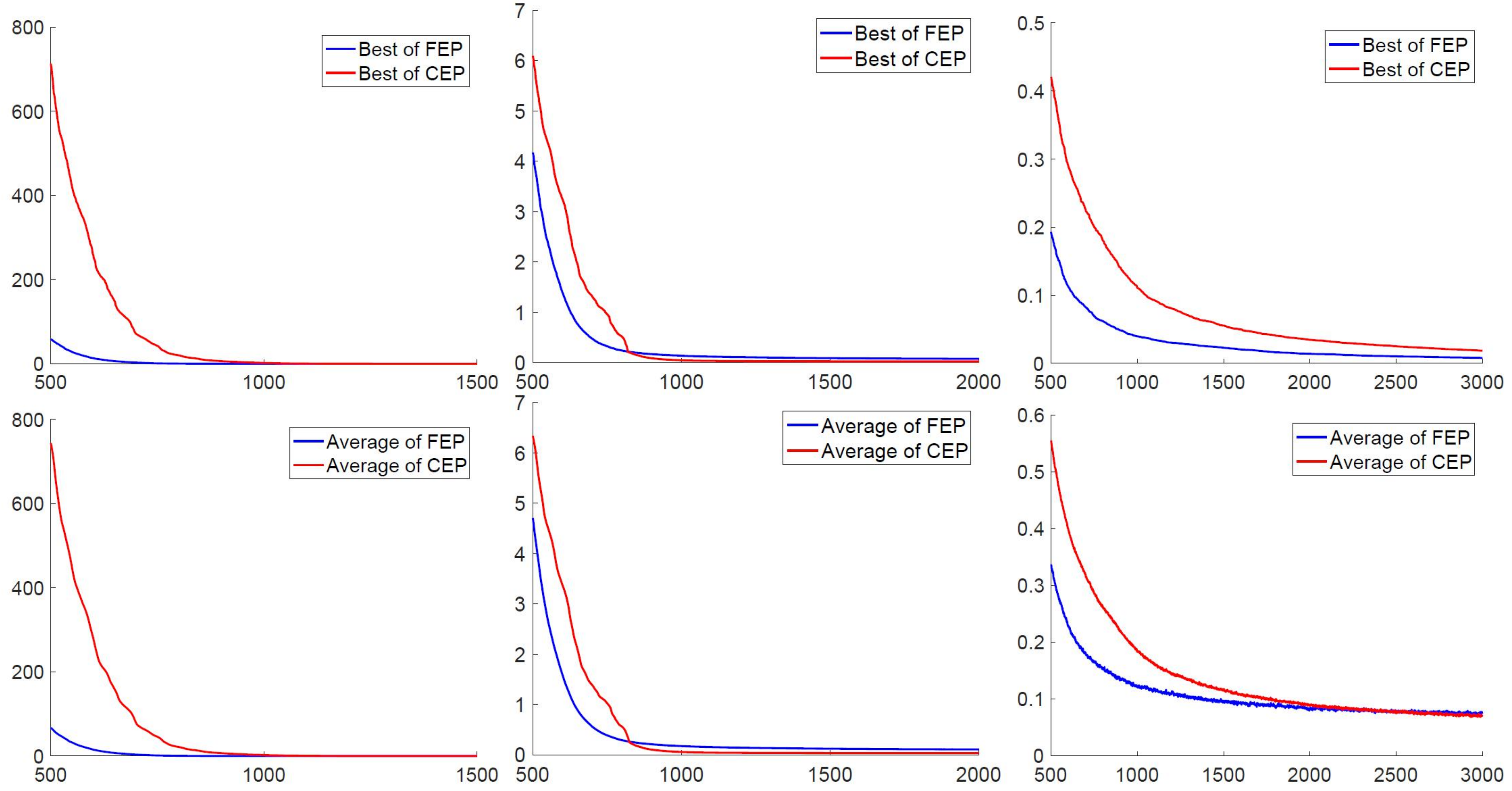


Figure 6: Evolutionary curves of CEP and FEP for f_1 (left), f_2 (middle) and f_7 (right). Plot from 500^{th} generation.

Statistical Tests and Comparisons

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

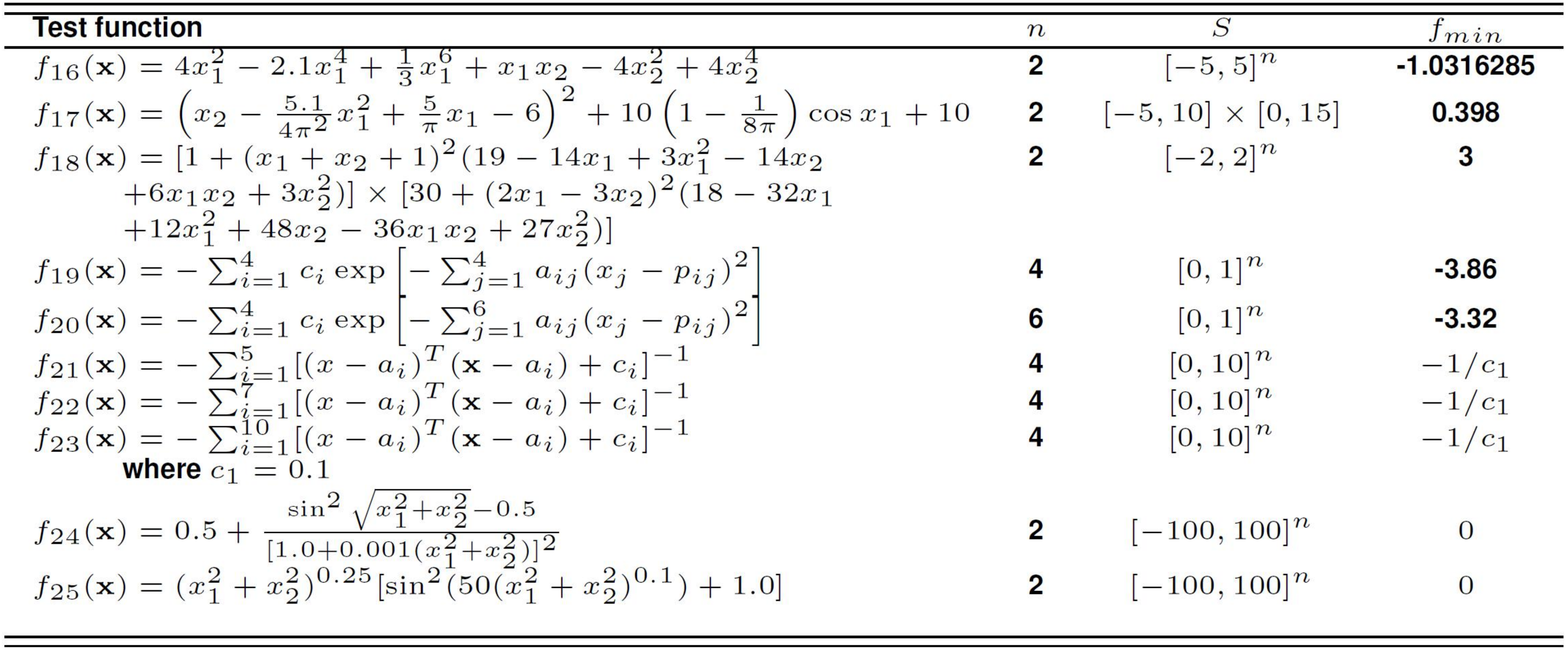
[1] Why it is called "Student's T-test"? An interesting history to know:)

Multimodal Functions f_8-f_{15}

- #local minima increases exponentially with n

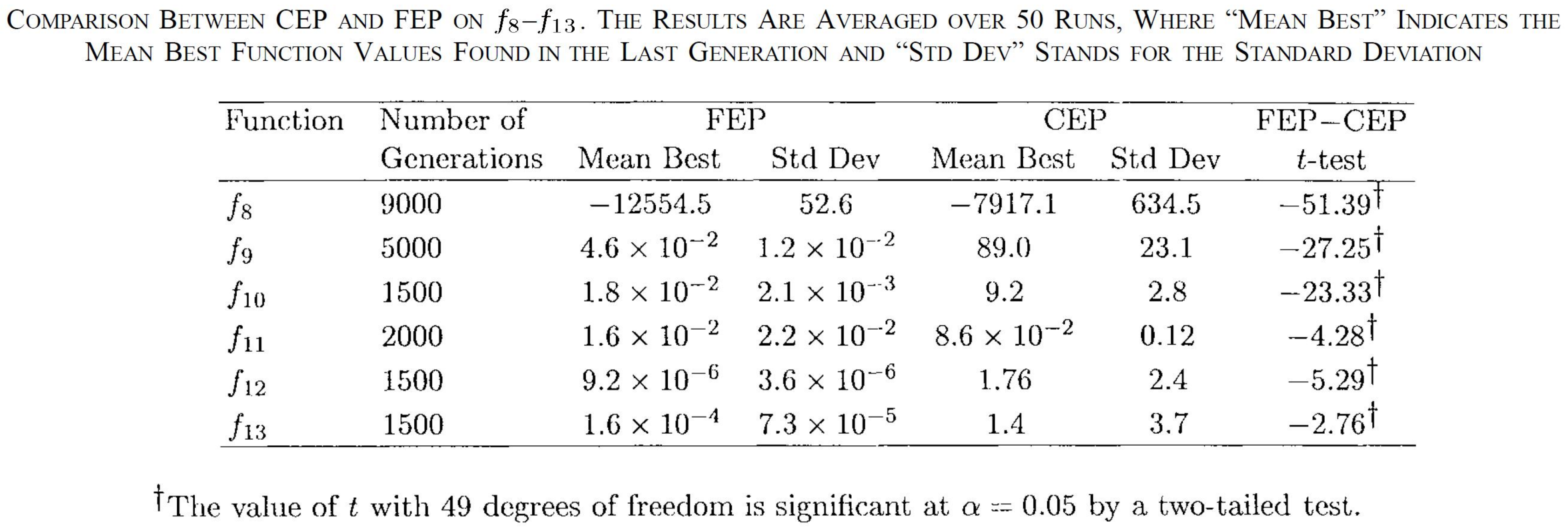
Test function	n	S	f_{min}
$f_8(\mathbf{x}) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	**30**	$[-500, 500]^n$	**-12569.5**
$f_9(\mathbf{x}) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10)]$	**30**	$[-5.12, 5.12]^n$	**0**
$f_{10}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) +20 + e$	**30**	$[-32, 32]^n$	**0**
$f_{11}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	**30**	$[-600, 600]^n$	**0**
$f_{12}(\mathbf{x}) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] +(y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \leq x_i \leq a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$	**30**	$[-50, 50]^n$	**0**
$f_{13}(\mathbf{x}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] +(x_n - 1)^2[1 + \sin^2(2\pi x_n)]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	**30**	$[-50, 50]^n$	**0**
$f_{14}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	**2**	$[-65.536, 65.536]^n$	**1**
$f_{15}(\mathbf{x}) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	**4**	$[-5, 5]^n$	**0.0003075**

Multimodal Functions f_{16}-f_{25}



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

Experiments on Multimodal Functions f_8-f_{13}

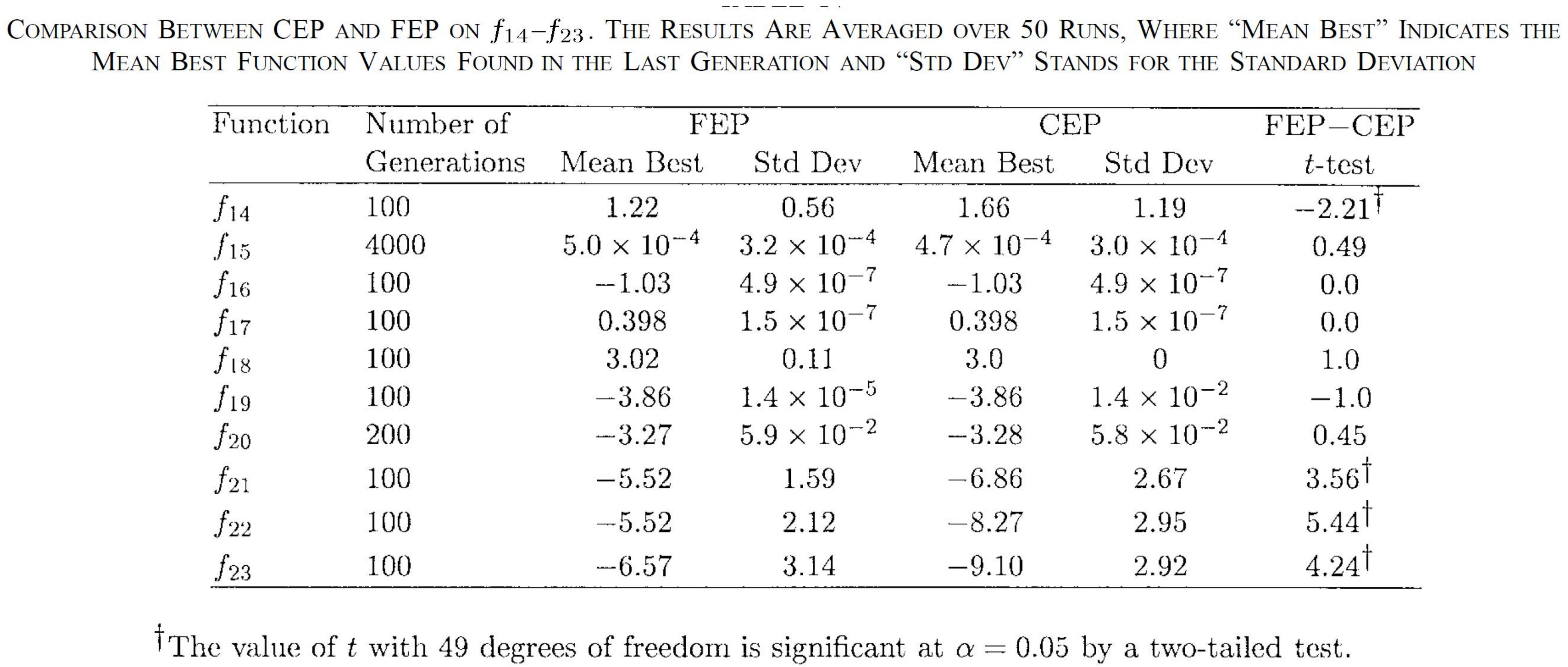


COMPARISON BETWEEN CEP AND FEP ON f_8–f_{13}. THE RESULTS ARE AVERAGED OVER 50 RUNS, WHERE "MEAN BEST" INDICATES THE MEAN BEST FUNCTION VALUES FOUND IN THE LAST GENERATION AND "STD DEV" STANDS FOR THE STANDARD DEVIATION

Function	Number of	FEP		CEP		FEP−CEP
	Generations	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.

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



COMPARISON BETWEEN CEP AND FEP ON f_{14}–f_{23}. THE RESULTS ARE AVERAGED OVER 50 RUNS, WHERE "MEAN BEST" INDICATES THE MEAN BEST FUNCTION VALUES FOUND IN THE LAST GENERATION AND "STD DEV" STANDS FOR THE STANDARD DEVIATION

Function	Number of Generations	FEP Mean Best	FEP Std Dev	CEP Mean Best	CEP Std Dev	FEP−CEP 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.

Outline of This Lecture

- Self-adaptation and Parameter Control in Evolutionary Algorithms
    - ✓Global Optimisation by Mutation-Based EAs
    - ✓What Do Mutation and Self-Adaptation Do
    - ✓More about Self-adaptation and Parameter Control in EAs
- **Representation is Important**
- Summary

Representation is Important [2]

- What: What is an individual's representation?
- Why: Why is it the most critical design choice?
- How: Common representation schemes
  - ✓ Binary Strings
  - ✓ Real-Valued Vectors
  - ✓ Permutations
  - ✓ Trees & Graphs
- So What: The mapping from genotype to phenotype and its consequences.
- Conclusion & Key Takeaways

The Core Idea: What is Representation?

- The Individual's "DNA"
- Definition: The representation is a data structure that encodes a potential solution to the problem at hand. It's the genotype.
- It defines the search space the EA will explore.
- The actual solution, as evaluated by the fitness function, is the phenotype.
- Analogy: Your genome (genotype) influences, but does not solely determine, your physical traits and abilities (phenotype).

Why Does Representation Matter? A Thought Experiment

- The Right Representstion for the Right Problem
- Example: Finding the Highest Point in a Landscape
- Representation A (Real-Valued): (x, y)
  - ✓ (2.51, -1.87)
  - ✓ Crossover: Blend x and y values of parents.
  - ✓ Mutation: Add a small random number to x or y.
  - ✓ Intuitive? Yes! Exploits problem structure.
- Representation B (Binary String): Encode x and y as separate 10-bit strings: 0101010101 1010101010
  - ✓ Crossover: Cut and splice bit strings.
  - ✓ Mutation: Flip a random bit.
  - ✓ Problem: A single bit flip can drastically change the value (e.g., 0111111111 to 1111111111). Less efficient for this smooth problem.
- Key Insight: Representation B creates a more rugged, deceptive landscape for the same problem. The algorithm didn't change; the representation did.

The Link to Search Operators

- Representation Dictates the "Rules of Evolution"
- You cannot design crossover and mutation in a vacuum. They are defined on the representation.
  - ✓ Bad Fit: A representation that doesn't match the problem leads to operators that create nonsensical or poor-quality offspring.
  - ✓ The Principle of Meaningful Building Blocks (Schema Theorem): Effective EAs combine good "genes" (short, low-order schemas). The representation must allow for these building blocks to exist and be combined meaningfully.
- Example: You can't use simple one-point crossover on a simple list to solve the Traveling Salesperson Problem (TSP). It would create duplicate and missing cities.

Common Representation 1: Binary Strings

- The Classic Representation
- A string of 0s and 1s (Chromosome): 1 0 1 1 0 0 1 0 1 1
- Best For:
  - ✓ Theoretical analysis (Schema Theorem).
  - ✓ Problems with discrete, on/off decisions (e.g., feature selection, knapsack problem).
- Pros:
  - ✓ Simple, universal.
  - ✓ Large body of theoretical work.
- Cons:
  - ✓ Hamming Cliff: A small change in genotype (e.g., 1111 (15) to 0111 (7)) can be a large change in phenotype (15->7). Can hinder local search.
  - ✓ Often not a natural fit for real-world problems.

Common Representation 2: Real-Valued Vectors

- For Continuous Optimization
- A vector of floating-point number (Chromosome): [45.2, -12.8, 0.05, 9.99]
- Best For: Continuous parameter optimization (e.g., tuning controller parameters, function minimization).
- Pros:
  - ✓Natural representation for many problems.
  - ✓Enables powerful operators (Blend Crossover alpha (BLX-α crossover), Gaussian mutation).
  - ✓Efficient local search.
- Cons:
  - ✓Less studied theoretically than binary representations.

Common Representation 3: Permutations

- For Ordering Problems
- A list of numbers representing a path (Chromosome): [A, D, B, F, C, E] (A tour for TSP)
- Best For: Ordering/sequencing problems (TSP, scheduling, task ordering).
- Challenge: Standard crossover destroys the permutation (creates duplicates/missing elements).
- Solution: Specialized Operators!
    - ✓ Order Crossover (OX): Preserves relative order from one parent, absolute position from the other.
    - ✓ Partially Mapped Crossover (PMX): Preserves absolute positions and mappings.
    - ✓ Swap Mutation: Swaps two random elements.

Common Representation 4: Trees & Graphs

- For Structure and Program Evolution
- A tree diagram representing a mathematical expression (Chromosome): (+ (* 2 x) 5) which is y = 2*x + 5
- Best For: Genetic Programming (evolving programs, formulas, decision trees), evolving neural network structures.
- Pros: Immense flexibility and expressiveness.
- Cons:
  - ✓High computational cost.
  - ✓Bloat: Trees can grow uncontrollably without compromising fitness.
  - ✓Requires complex crossover (subtree swap) and mutation operators.

The Mapping: Genotype -> Phenotype

- The Translation Layer
- The fitness function evaluates the phenotype. The representation is the genotype. The mapping between them is crucial.
- Direct Mapping: Genotype is the phenotype (e.g., (x,y) for a point).
- Indirect Mapping (Grammatical Evolution): Genotype is a string of bits/integers that are used as instructions to build a phenotype (e.g., a program) using a grammar.
- Benefit: Separates the search space from the solution space. A small change in genotype can lead to a small or large change in phenotype, decoupling the landscape.

Practical Guidelines & Pitfalls

- How to Choose? Ask These Questions
  1. What is a natural way to represent a solution? (Often the best starting point).
  2. Do my chosen genetic operators produce valid offspring?
  3. Does the representation avoid creating inherent bias or deception?
  4. Does the representation support the creation of meaningful building blocks?
  5. Is the representation efficient to evaluate?
- Pitfall: Choosing a representation because it's "standard" rather than because it's "right for the problem."

Representation is Foundational

- The choice of representation is the first and most important design decision in an EA.
- It defines the search space and dictates the design of genetic operators.
  - ✓A poor representation can make a problem unsolvable;
  - ✓a good one can make it trivial.
- There is no single best representation for all problems. The "No Free Lunch" theorem applies here.
- Always let the problem domain guide your choice.

Michigan vs. Pittsburgh Representation

individual 1
individual 2
...
individual n

each individual reprents a potential solution

$\downarrow$

Pittsburgh approach

individual 1
individual 2
...
individual n

all or a subset of individuals reprent a potential solution

$\downarrow$

Michigan approach

The Michigan Approach

- Core Idea: Each individual in the population represents a single rule or solution component. The entire population collectively represents the complete solution.
  - ✓ Individual: IF condition THEN action (e.g., a classifier rule, a neural network node)
  - ✓ Population: The entire rule set / solution.
  - ✓ Fitness Assignment:
    - ➢ Credit Assignment Problem: How to evaluate a single rule's contribution to the overall solution's performance?
    - ➢ Often uses techniques like the Bucket Brigade Algorithm or Profit Sharing.
- Analogy: A Swarm of Experts
  - ✓ Each individual is a specialist.
  - ✓ The group's collective knowledge solves the problem.
- Common Application: Learning Classifier Systems (LCS)

Population = Complete Solution
[Individual 1: Rule A]
[Individual 2: Rule B]
[Individual 3: Rule C]
[...]
[Individual N: Rule Z]
```

*The entire population is the solution.*
```

The Pittsburgh Approach

- Core Idea: Each individual in the population encodes a complete set of rules or a full solution. The population is a set of competing full solutions.
  - ✓ Individual: [Rule A, Rule B, Rule C, ..., Rule Z] (e.g., a full classifier system, a complete neural network)
  - ✓ Population: A set of competing candidate solutions.
  - ✓ Fitness Assignment:
    - ➢ Straightforward: The fitness of an individual is the performance of the entire rule set/solution it encodes.
    - ➢ No credit assignment issue.
- Analogy: A Team of Competing Companies
  - ✓ Each individual is a complete, self-sufficient company.
  - ✓ The best company (individual) wins.
- Common Application: Genetic Programming, Evolved Neural Networks

Population = Competing Solutions
[Individual 1: [Rule A, Rule B, ...]] → Fitness 85%
[Individual 2: [Rule C, Rule D, ...]] → Fitness 92% ←
[Individual 3: [Rule E, Rule F, ...]] → Fitness 78%
*```*

Each individual is a complete solution. The best one is selected.

Head-to-Head Comparison

Feature	Michigan Style	Pittsburgh Style
Individual	A single rule / partial solution	A complete solution (set of rules)
Solution	The entire population	The best individual in the population
Fitness Eval.	Complex (Credit Assignment)	Simple & Direct
Chromosome Length	Short, Fixed	Long, Variable (can use GAs/GP)
Search Space	Smaller per individual	Larger per individual
Computational Cost	Lower per evaluation	Higher per evaluation
Strengths	Co-operation, adaptivity, scalability for large rule sets	Simplicity, powerful GA operators, clear fitness
Weaknesses	Credit assignment problem, competition vs. cooperation	Can be computationally expensive, slower adaptation

Summary

- Fundamental Difference: It's a question of granularity.
  - ✓ Michigan: Fine-grained. Evolves the pieces of a solution.
  - ✓ Pittsburgh: Coarse-grained. Evolves entire solutions.
- When to Use Which?
  - ✓ Use Pittsburgh when your problem has a natural, variable-length structure (e.g., GP) and fitness is easy to define for a full solution.
  - ✓ Use Michigan for rule discovery and problems where solutions benefit from co-adaptive, cooperative parts (e.g., adaptive agents, data mining).
- Hybrid Approaches Exist: Modern systems often blend concepts from both paradigms to leverage their respective strengths.

Example: Robot Pursuit - one prey

population

Robot1^1, Robot2^1, Robot3^1, ..., RobotN_s^1

Robot1^2, Robot2^2, Robot3^2,..., RobotN_s^2

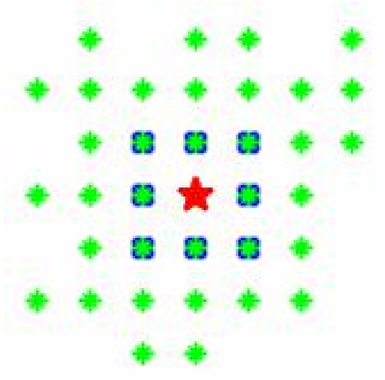
……

Robot1^i, Robot2^i, Robot3^i, ..., RobotN_s^i

……

Robot1^{50}, Robot2^{50}, Robot3^{50},..., RobotN_s^{50}

individual



N_s : number of predators

How to simulate and evaluate each individual?

population

Robot1^1

Robot1^2

……

Robot1^i

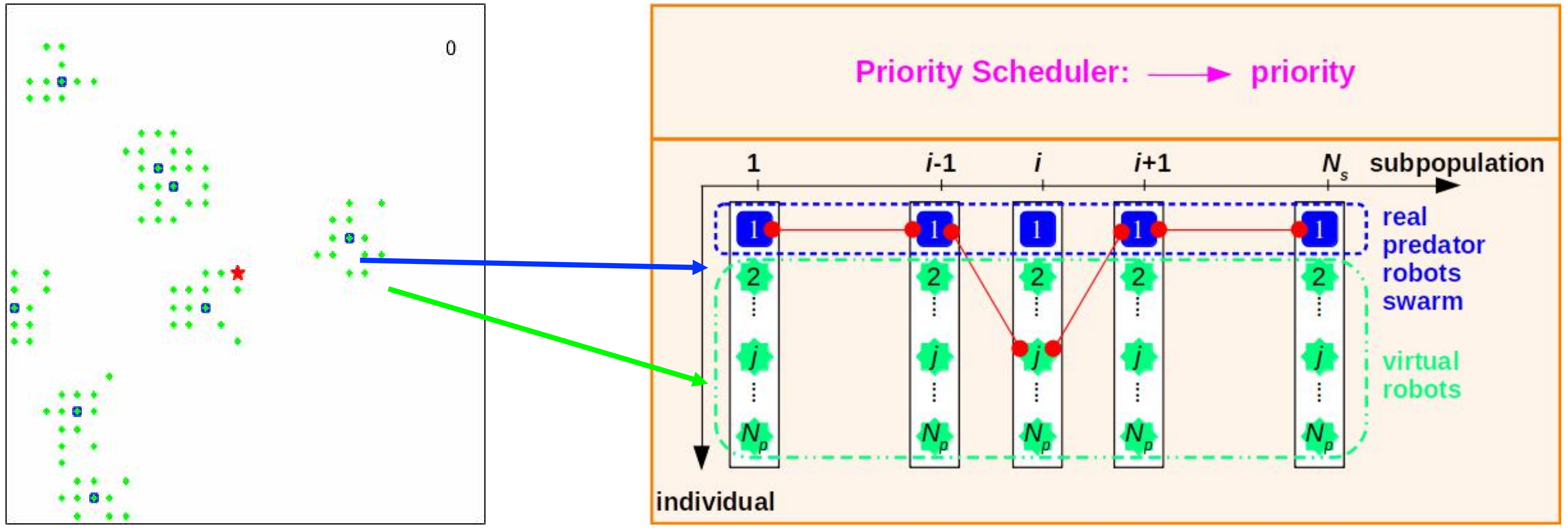
……

Robot1^{50}

individual

[3] Sun, Lijun, Chao Lyu, and Yuhui Shi. "Cooperative coevolution of real predator robots and virtual robots in the pursuit domain." Applied Soft Computing 89 (2020): 106098.

Robot Pursuit - one prey [3]



CCPSO-R

N_s : the number of subpopulation;

N_p : the number of individuals in each subpopulation

Each subpopulation:

contains one real predator and N_p- 1 virtual predator;

evolves based on PSO;

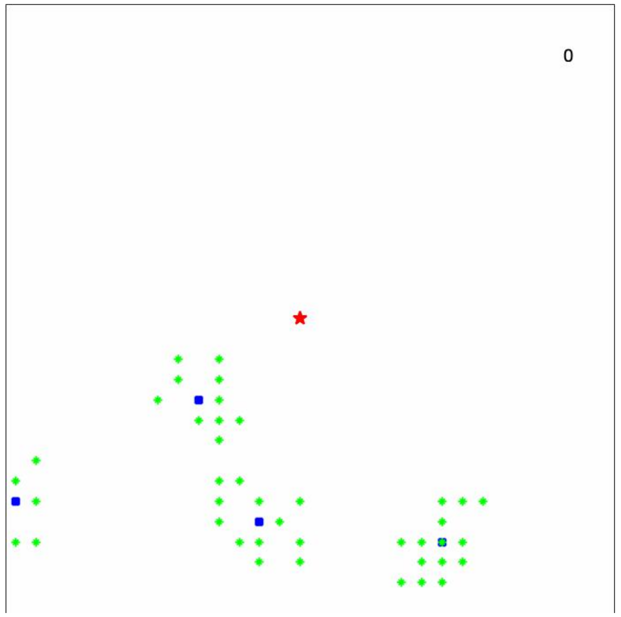
evaluation:

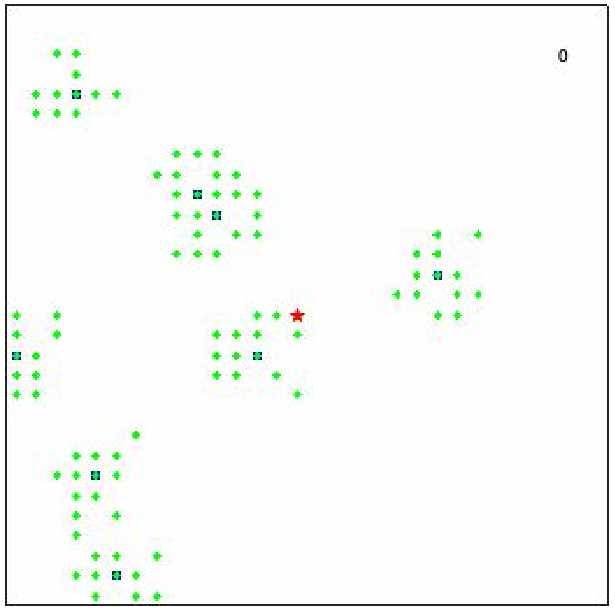
real predator: *all N_s real predators from N_s subpopulations*

each virtual predator: *this virtual predator + N_s -1 real predators from other N_s subpopulations*

All subpopulations cooperate through evaluation

Robot Pursuit - one prey: experiments





Outline of This Lecture

- Self-adaptation and Parameter Control in Evolutionary Algorithms
  - ✓Global Optimisation by Mutation-Based EAs
  - ✓What Do Mutation and Self-Adaptation Do
  - ✓More about Self-adaptation and Parameter Control in EAs
- Representation is Important
- **Summary**

Summary

- Self-adaptation and Parameter Control in Evolutionary Algorithms
- Representation is Important
- Michigan and Pittsburgh Representations
    - ✓Robot Pursuit - one prey

Essential Reading for This Lecture

1. X. Yao, Y. Liu and G. Lin, “Evolutionary programming made faster,” IEEE Transactions on Evolutionary Computation, 3(2):82-102, July 1999.
2. DeepSeek
3. Sun, Lijun, Chao Lyu, and Yuhui Shi. "Cooperative coevolution of real predator robots and virtual robots in the pursuit domain." Applied Soft Computing 89 (2020): 106098.