CS361

(Software Engineering Program)

Artificial Intelligence II - Applied Machine Learning

Lecture 3

Evolutionary Computation: Differential Evolution as an Optimization Method

Amr S. Ghoneim

(Assistant Professor, Computer Science Dept.)

Helwan University

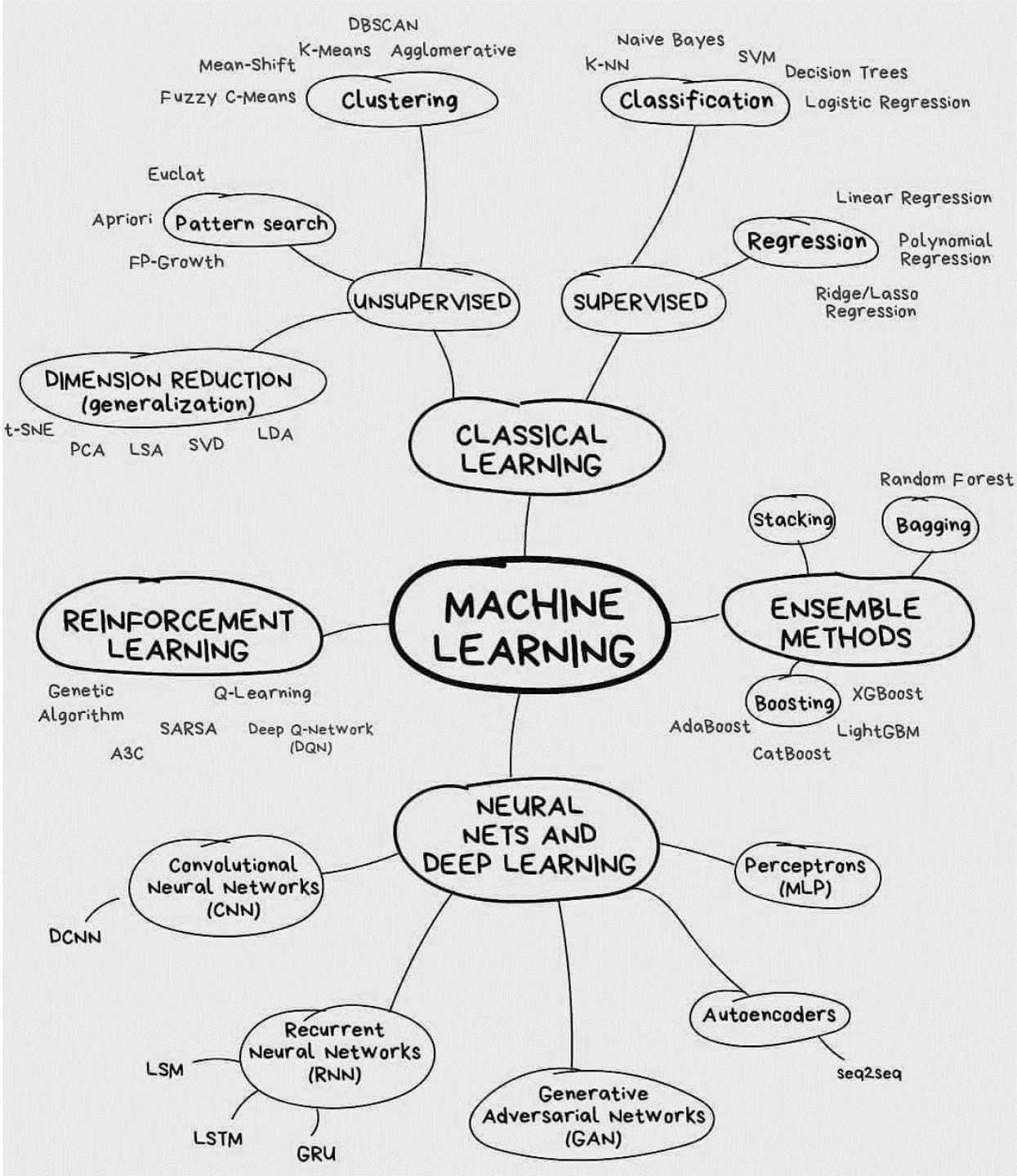
Fall 2019

Lecture is based on its counterparts in the following courses (and the following resources):

- *Evolutionary Computing*, **University of Vaasa (Finland),** Electrical & Energy Eng.
- *Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*, **Journal of Global Optimization** 11: 341–359, 1997.
- *Data Clustering Method based on Mixed Similarity Measures*, **(ICORES 2017),** 192-199
- *Differential Evolution,* **University of Colorado (USA),** Colorado Springs.

Machine Learning?

{*Artificial Intelligence*}
Machine Learning Map

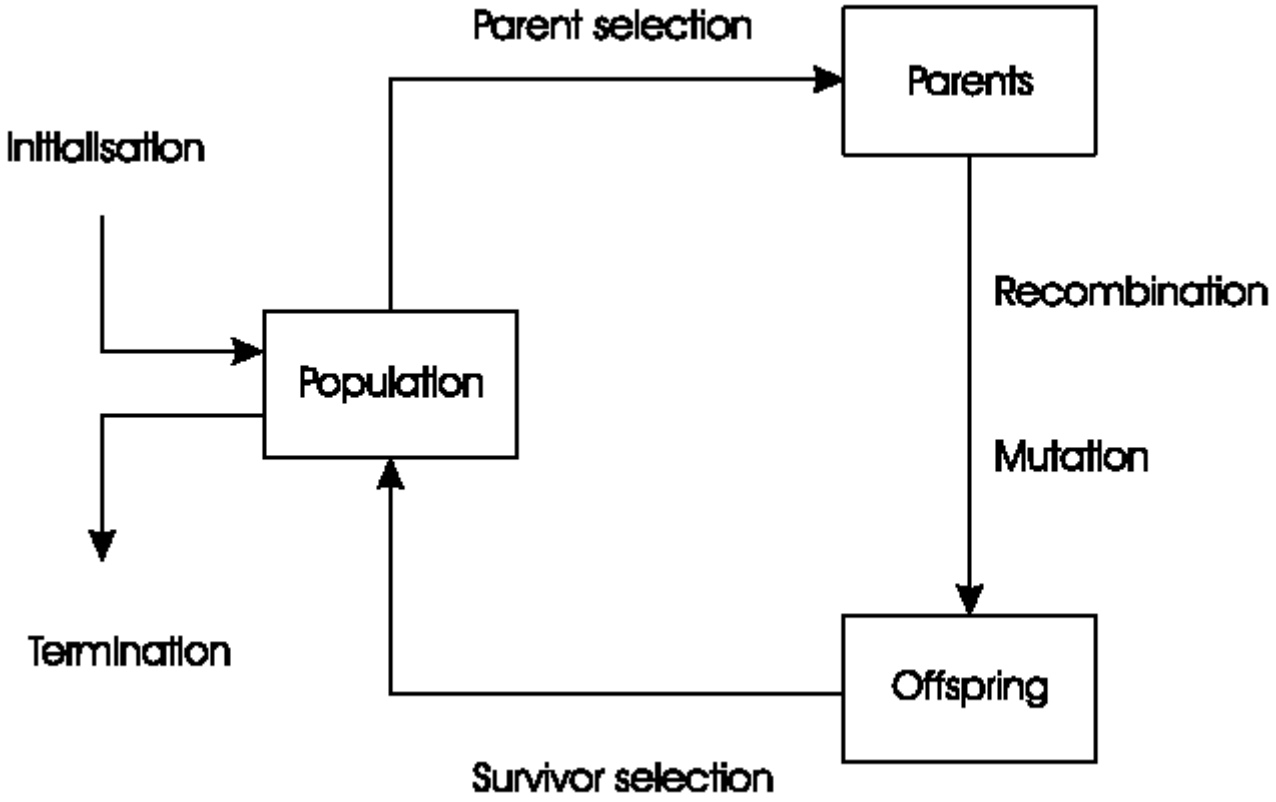


Recap:
Evolutionary Computation
Genetic Algorithms

Evolutionary Computation (EC)

- Evolutionary Computing *(or Evolutionary Algorithms "EA")* is a field of science and engineering that tries to apply some phenomena that appears in the nature to the optimization.
- Most notable is the adaptation of Charles Darwin's evolution theory in order to solve difficult search and optimization tasks.
- The method is universally applicable, since it has been successfully applied to almost all thinkable search & optimization problems in engineering, science & people's everyday life.

General Scheme of Genetic Algorithms

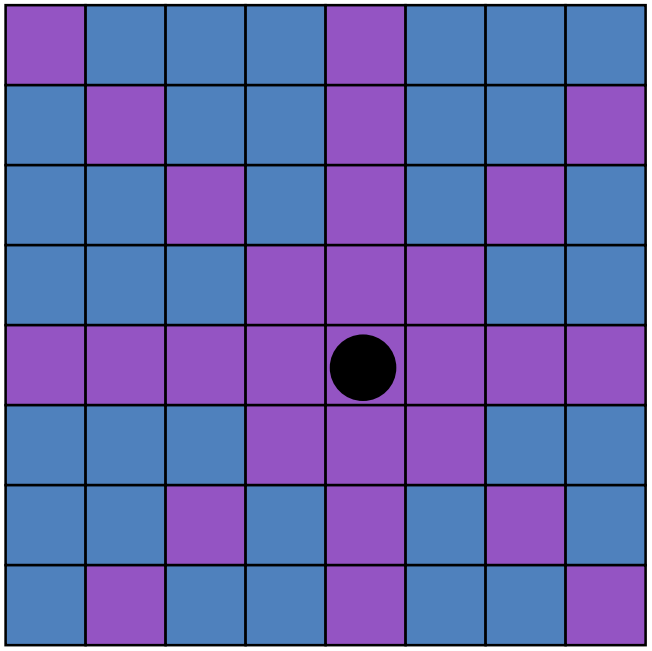


Pseudocode for a Typical Genetic Algorithm (GA)

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

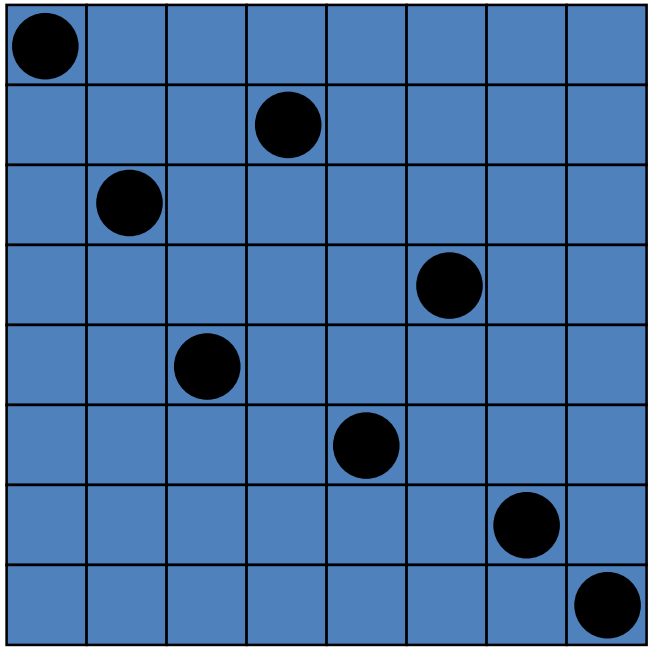
Example: The 8 Queens Problem ..

- Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other.



The 8 Queens Problem: *Representation ..*

Phenotype:
A board's configuration ..



Genotype:
A permutation of
the numbers 1 : 8

Obvious mapping

1	3	5	2	6	4	7	8

The 8 Queens Problem: *Fitness Function (Evaluation) ..*

- Penalty of one queen: the number of queens she can check.
- Penalty of a configuration: the sum of the penalties of all queens.
- Note: penalty is to be minimized
- Fitness of a configuration: inverse penalty to be maximized

The 8 Queens Problem: *Parents' Selection for Recombination (Crossover)..*

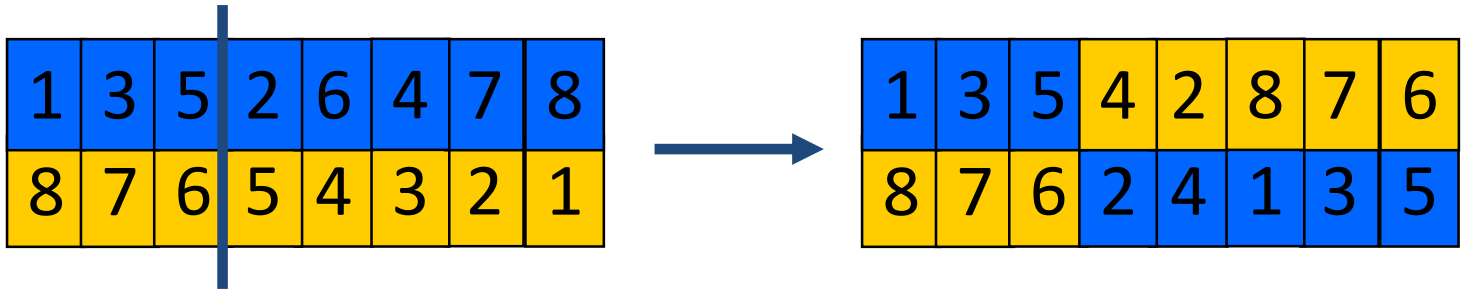
Selecting Parents for Crossover/Recombination:

- For e.g., by picking randomly 5 parents, and then select the best two to undergo crossover.

The 8 Queens Problem: *Recombination (Crossover) ..*

Combining two chromosomes *(in this case, permutations)* into two new permutations:

- Choose a random crossover point ..
- Copy first parts into both children ..
- Create the second part by inserting values from the other parent:
  - In the order they appear there (at that parent) ..
  - Beginning after the selected crossover point .. and ..
  - Skipping values already in the child.



The 8 Queens Problem: *Mutation ..*

- Small variation in one permutation:
- *For e.g.*, swapping values of two randomly chosen positions.

The 8 Queens Problem: *Parents' Selection for Replacement ..*

Survivors' Selection *(Replacement; "Creating a new generation")*

.. For Example: when inserting a new child into the population, choose an existing member to replace by:

- Sorting the whole population by decreasing fitness.
- Enumerating this list from high to low.
- Replacing the first with a fitness lower than the given child.

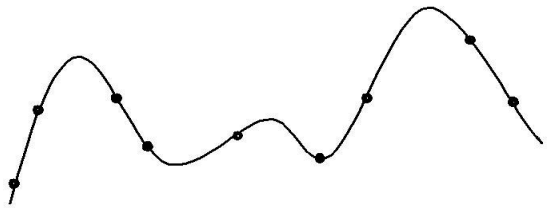
The 8 Queens Problem: *Summary ..*

Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

Note that this is ***only one possible***
set of choices of operators and parameters.

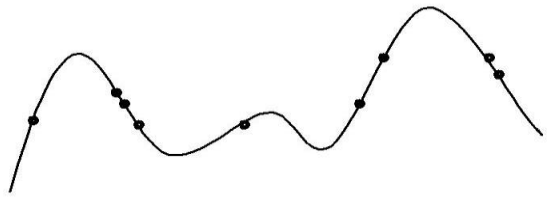
Typical Behaviour of an EA ..

Phases in optimising on a 1-dimensional fitness landscape:



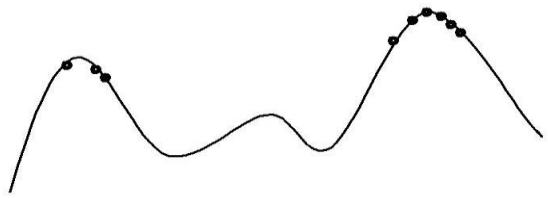
An Early Phase:

Quasi-random population distribution.



A Middle Phase:

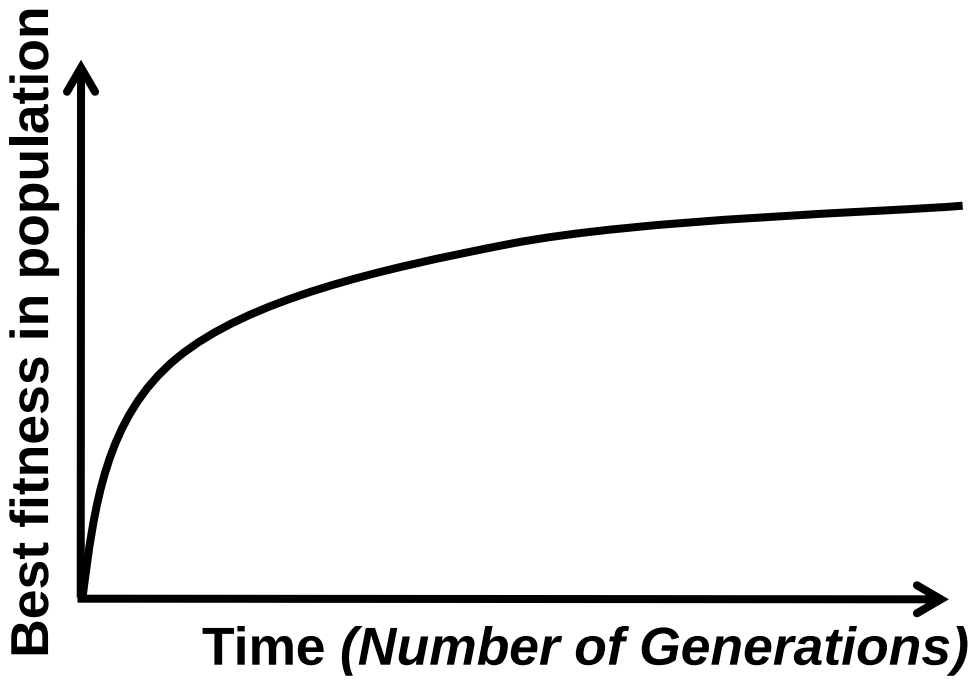
Population arranged around/on hills.



A Late phase:

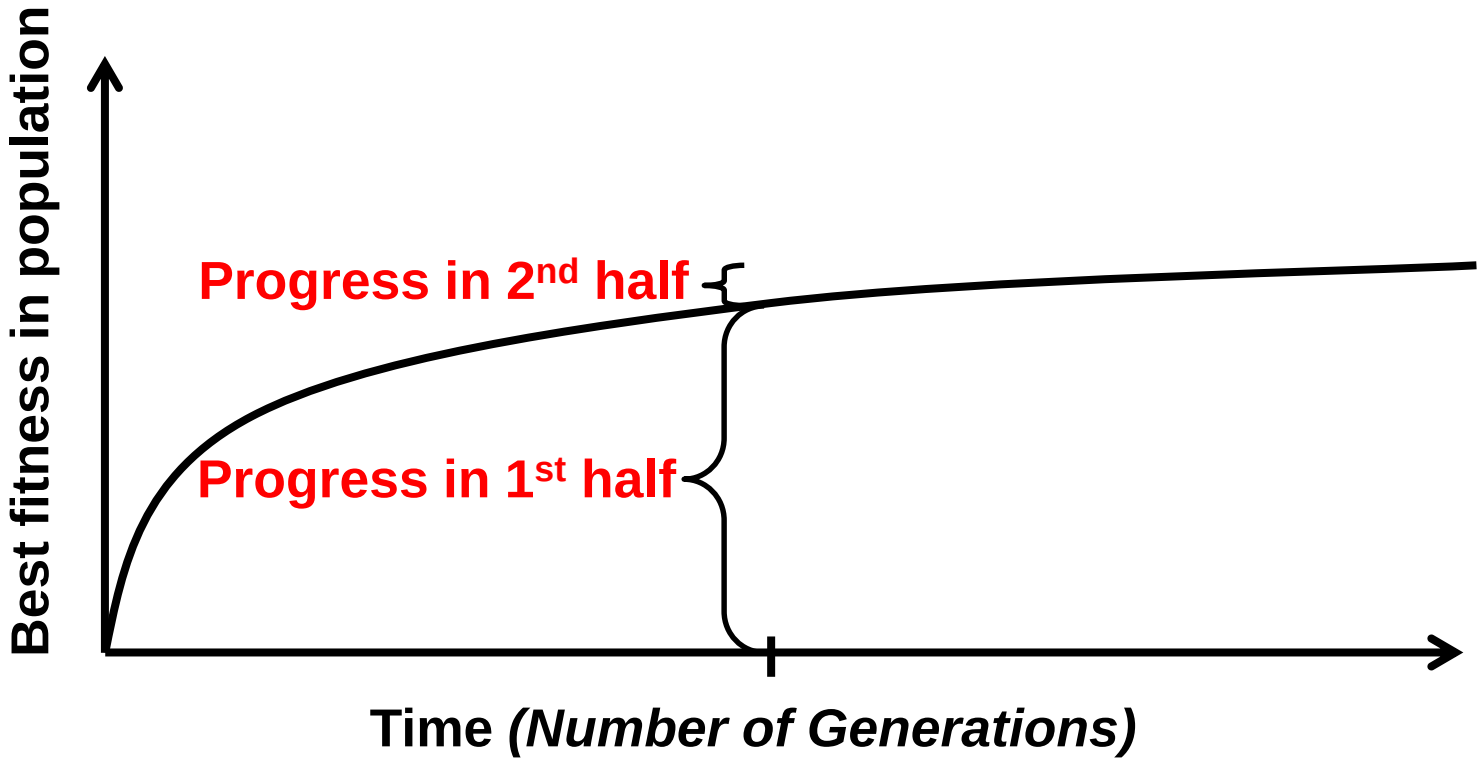
Population concentrated on high hills.

Typical Run: Progression of Fitness



Typical run of an EA shows so-called “anytime behavior” ..

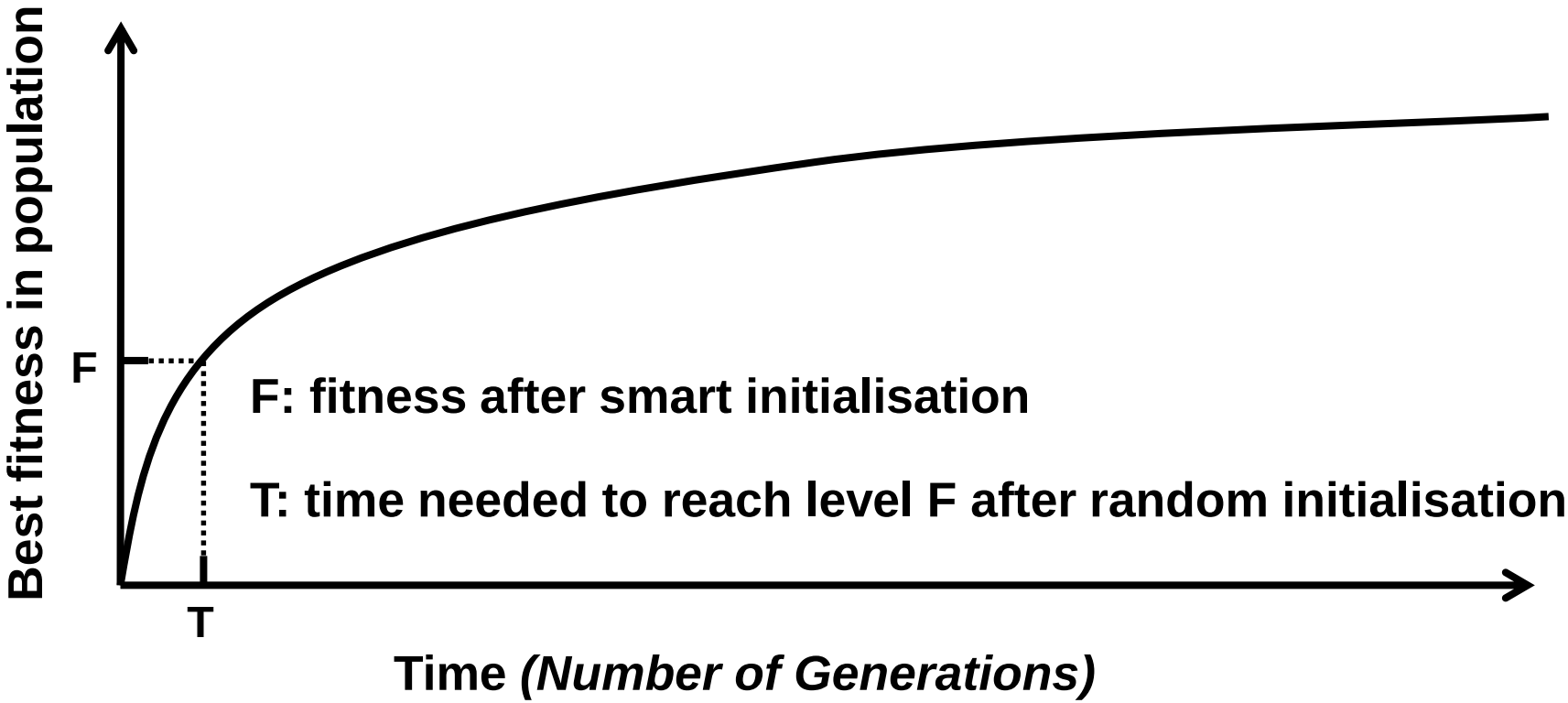
Are long runs beneficial?



Answer:

- *it depends on how much you want the last bit of progress ..*

Is it worth spending effort on smart initialisation?



Answer:

- it depends. Possibly, if good solutions/methods exist ..

Evolutionary Algorithms

Differential Evolution

A generic Nature-inspired population-based global-search metaheuristic optimization algorithm.

Based on vector differences and is therefore primarily suited for numerical optimization problems.

Requirements on a Practical Minimization Technique

- Ability to **handle** *non-differentiable*, *nonlinear*, and *multimodal* cost functions.
- **Parallelizability** to cope with computation intensive cost functions.
- **Ease of use**, in particular, there should be few control variables or parameters.
- **Good convergence** properties, i.e., consistent convergence to the global minimum (optimum) in consecutive independent trials.

Motivation for Introducing Differential Evolution (DE)

- DE was designed to be a stochastic direct search method. In other words, it satisfies the first requirement since it can be used to minimize any function for which a global minimum exists.
- DE uses a population of vectors where the stochastic perturbation of the vectors can be done independently. In other words, parallelization is easy.

Motivation for Introducing Differential Evolution (DE)

- **Ease of use:** To satisfy the third requirement, it is good if the minimization method is self-organizing so that it requires very little input from the user. DE alters the search space using information from within the vector population *(like Genetic Algorithms)*. DE's self-organizing scheme takes the difference of two randomly chosen population vectors to perturb an existing vector. The perturbation is done to every population vector.
- **Convergence:** DE has been found to converge very well in a large number of practical problems although theoretical discussions are not extensive.

Differential Evolution: *Basic Components ..*

- DE is a **nature-based** algorithm that **belongs to the class of genetic algorithms** since it uses selection, crossover, and mutation operators to optimize (**using a global-search metaheuristic**) an objective function over the course of successive generations (Suresh et al., 2009).
- DE is a **parallel population-based** direct search method where the population is comprised of ***NP* vectors** each of **dimension *D***, and is **primarily suited for numerical optimization** problems.
  - *NP* does not change during the algorithm's life time.
- The **initial population is chosen randomly** and should cover the entire parameter space.

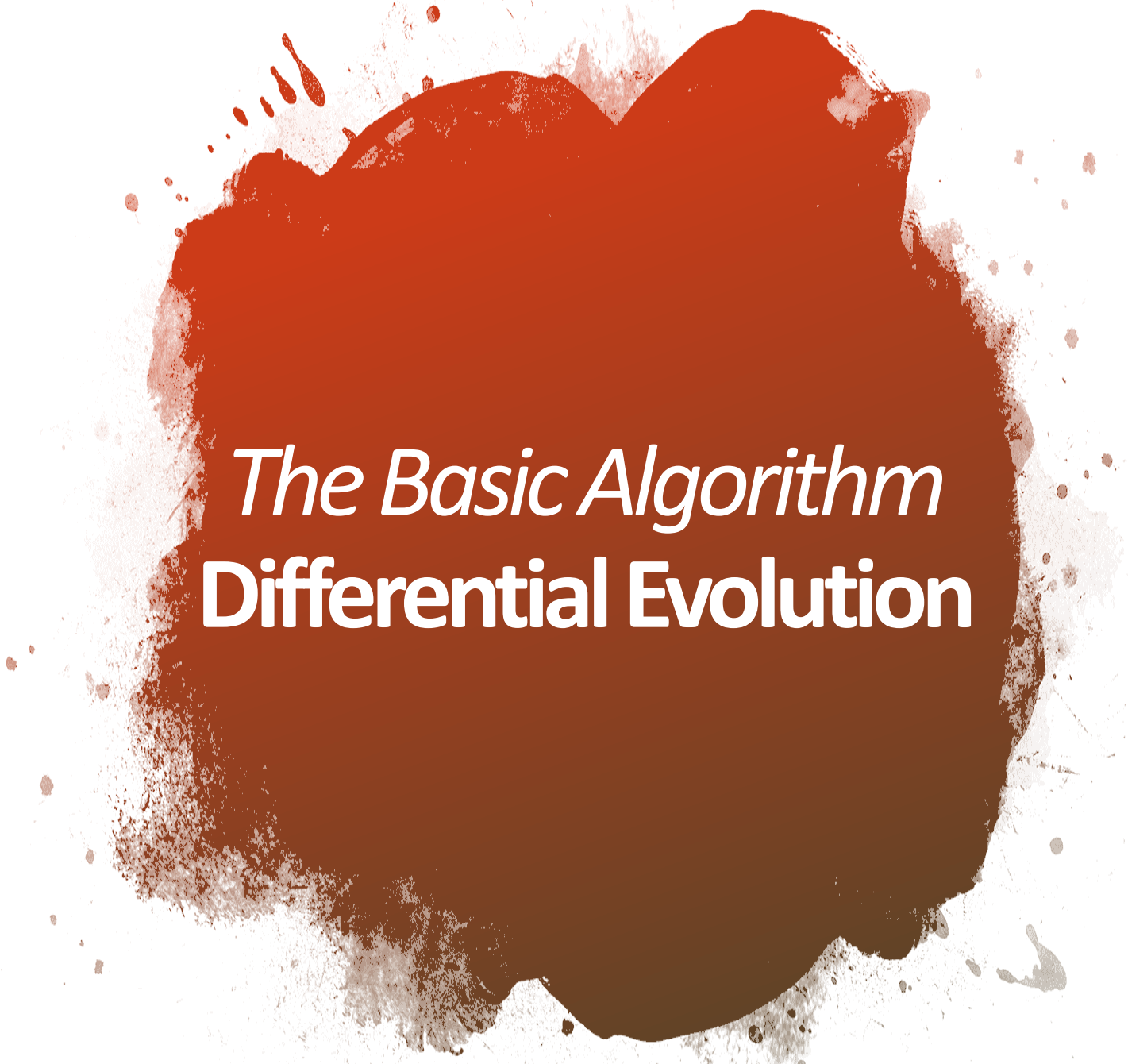
Differential Evolution: *Basic Components ..*

- **Mutation:** DE generates new individuals in the population by adding weighted difference between two population vectors to a third vector (the mutated vector).

- **Crossover:** The mutated vector's parameters are then mixed with the parameters of another pre-determined vector, the target vector, to yield a trial vector.

- **Selection:** If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the next generation.

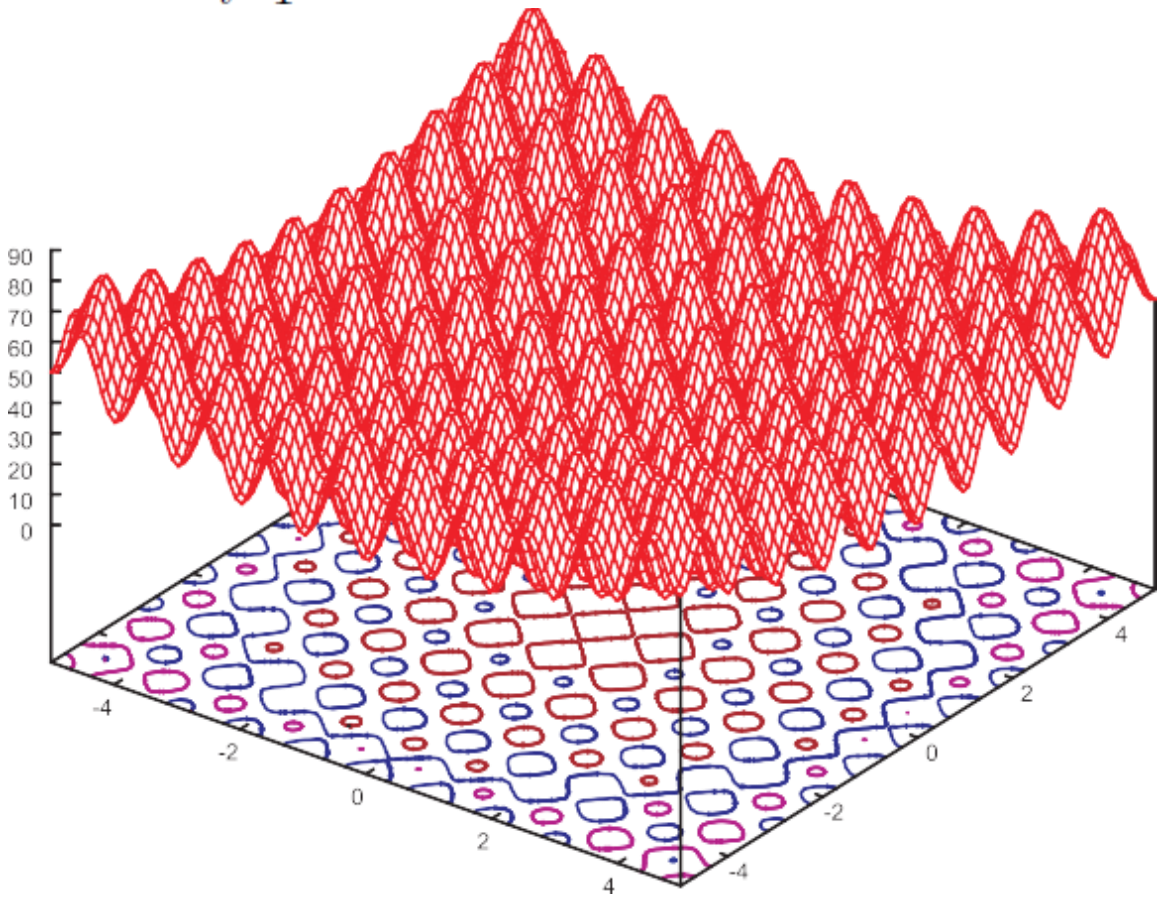
- Each population vector must serve once as the target vector so that NP competitions take place in one generation.


The Basic Algorithm
Differential Evolution

A Numerical Optimization Example:

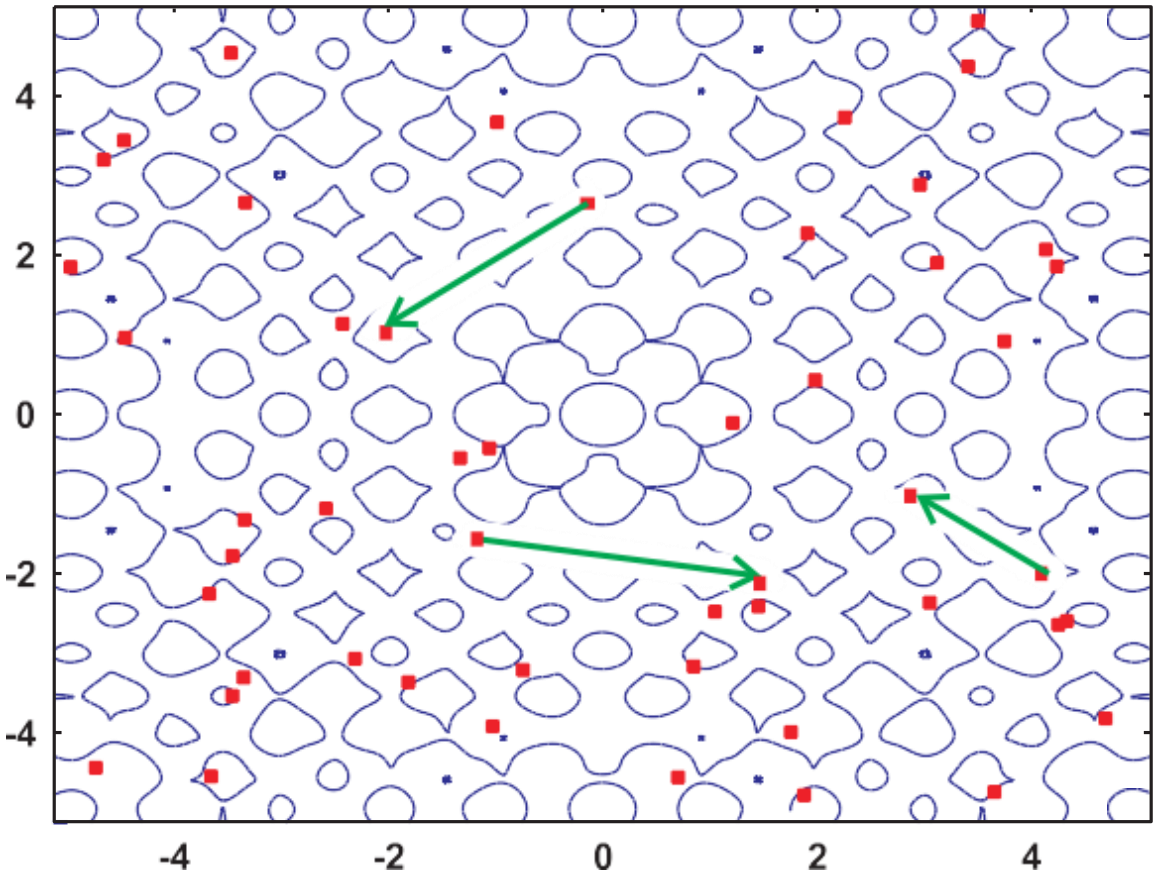
Minimizing a Function ..

$$\text{Minimize } f(\vec{x}) = 20 + \sum_{i=1}^{2} \left(x_i^2 - 10\cos\left(2 * \pi * x_i\right)\right) \text{ for } -5.12 \leq x \leq 5.12$$



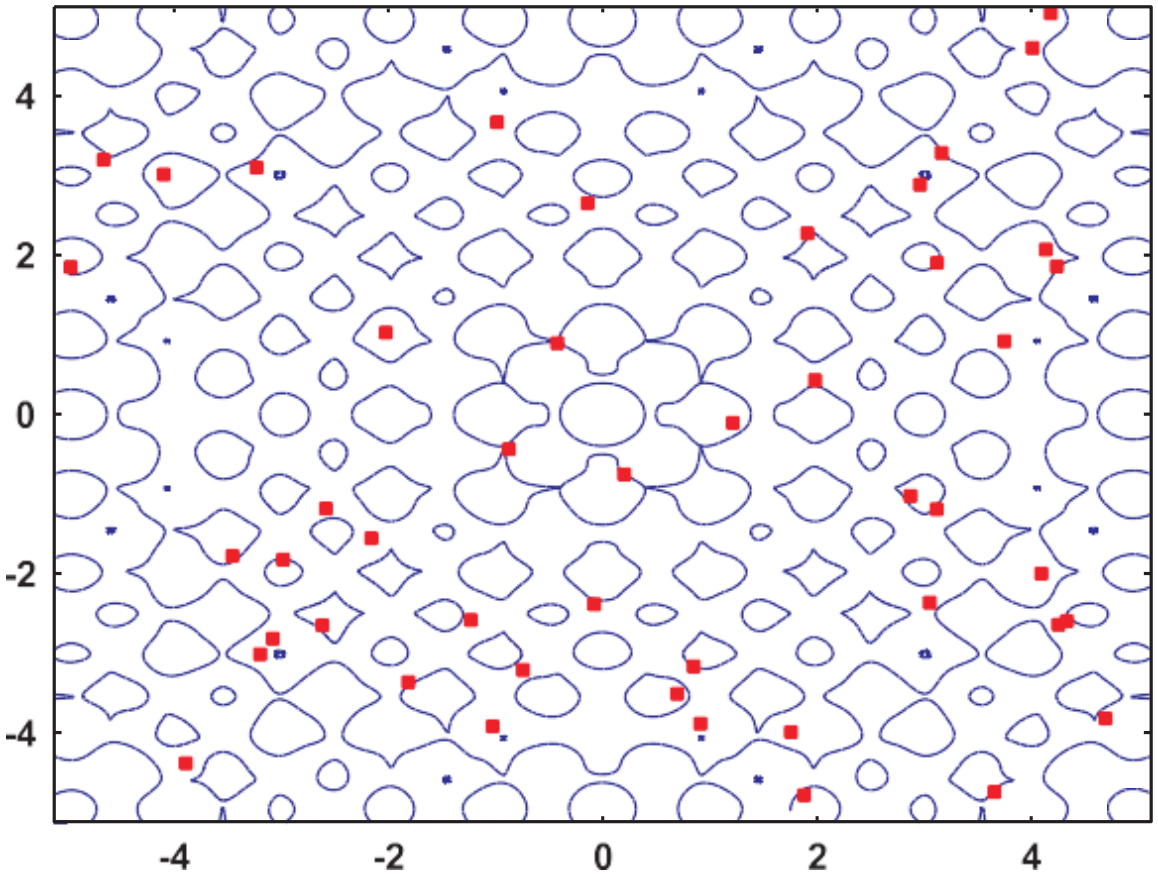
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 0: uniform spread, large differences**



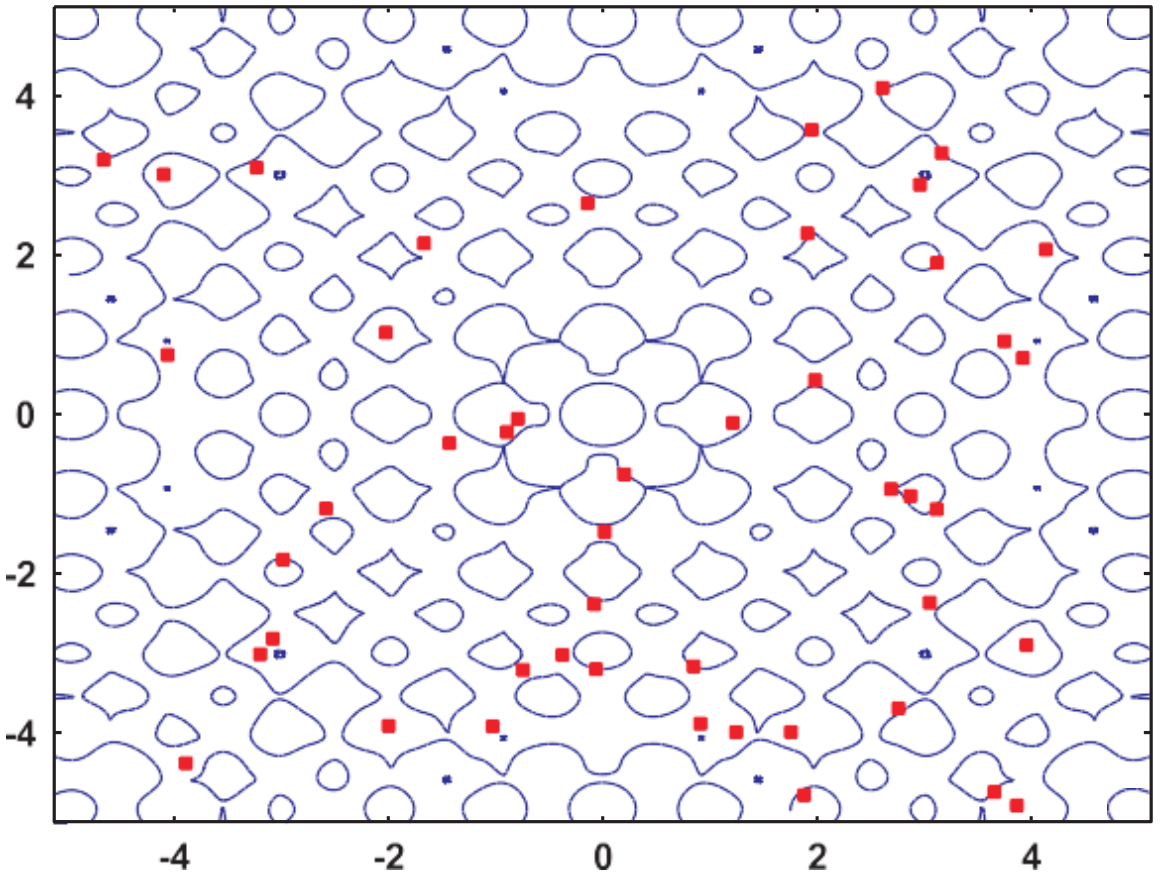
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 1 ..**



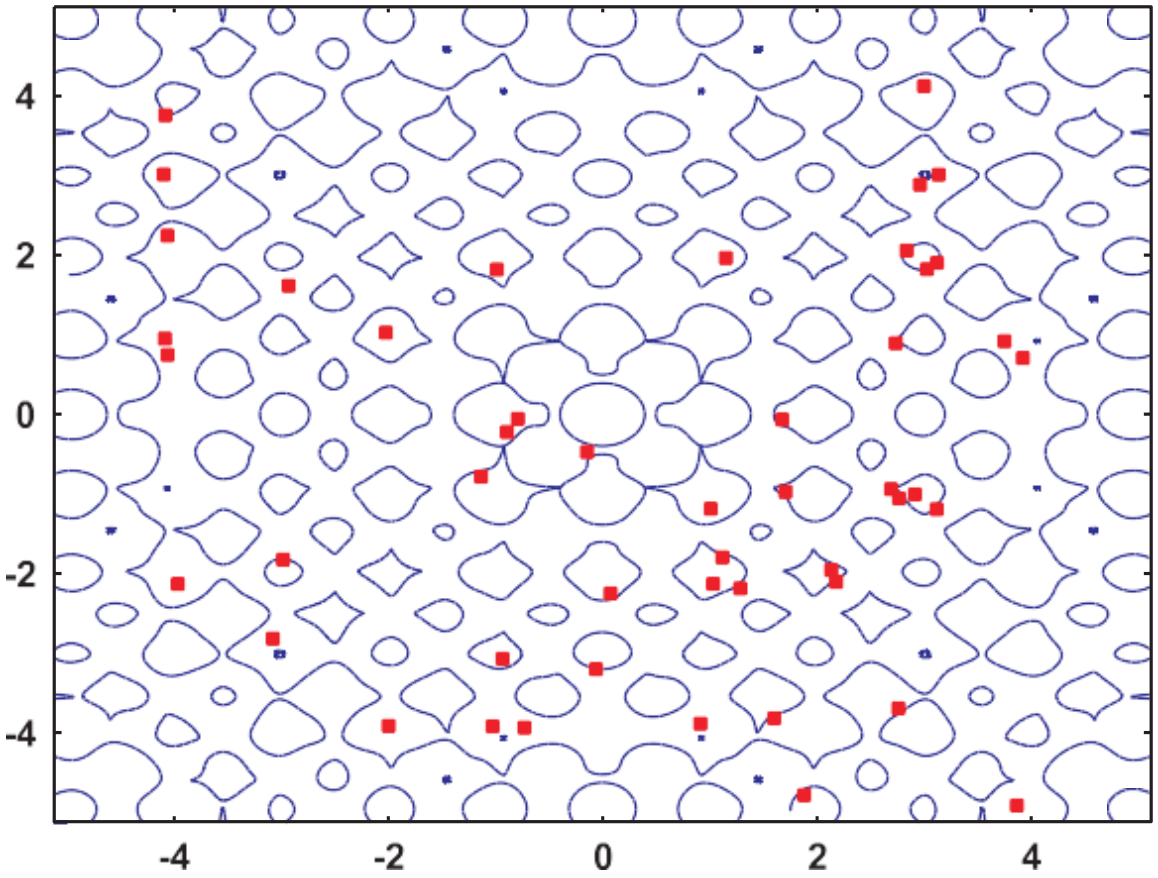
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 2 ..**



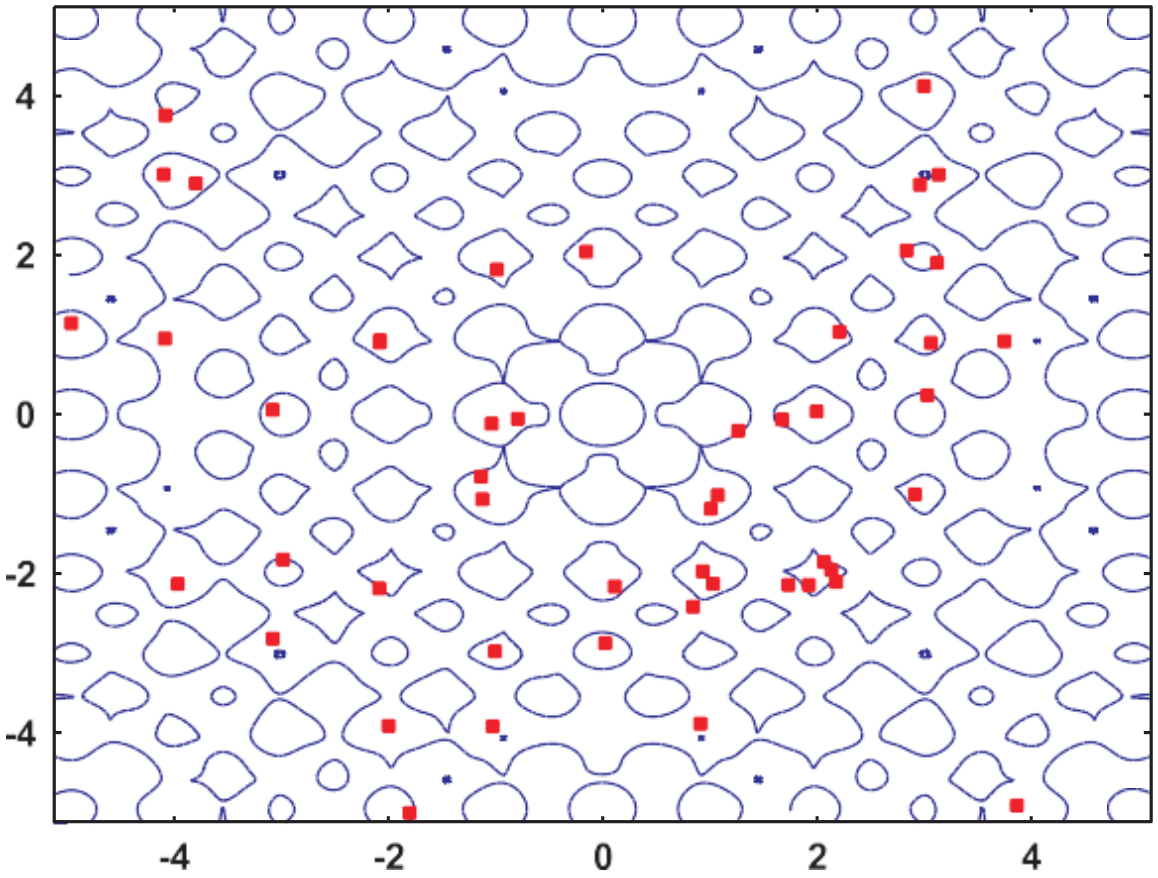
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 5 ..**



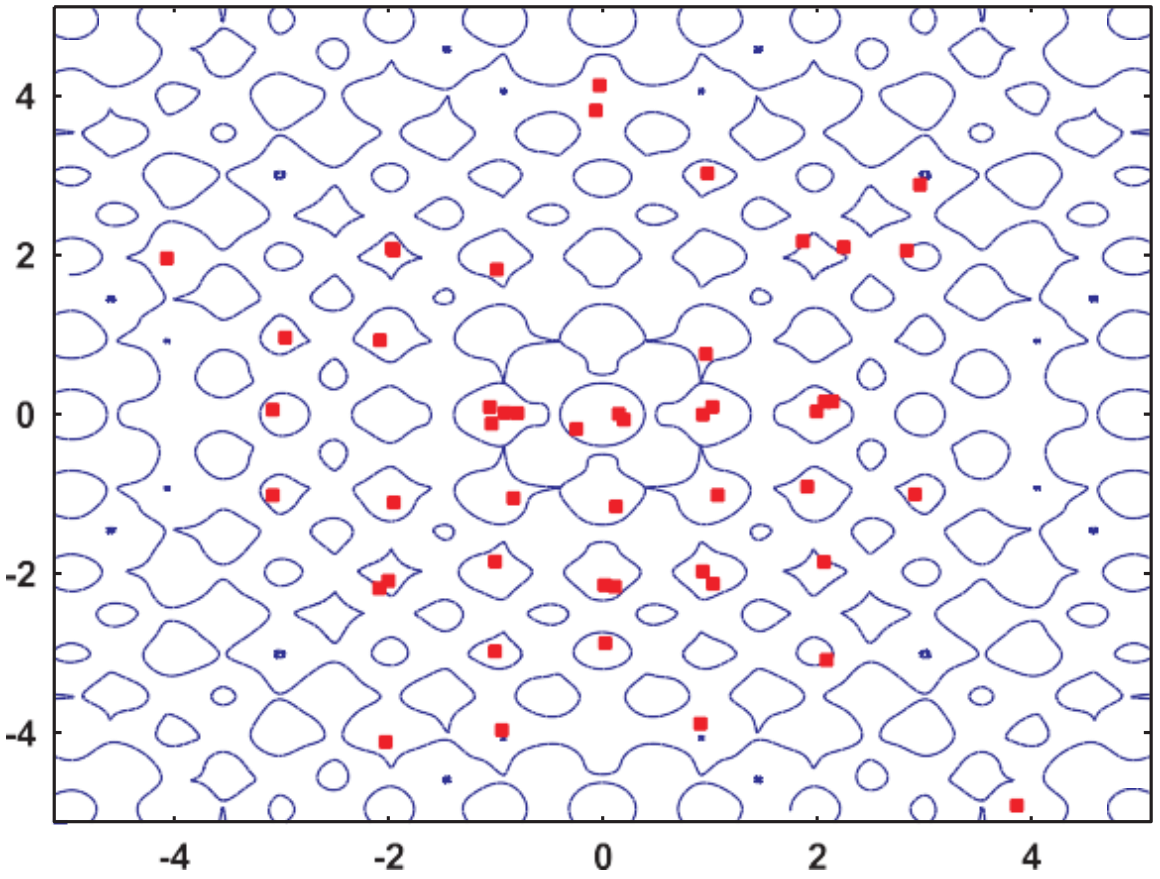
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 10 ..**



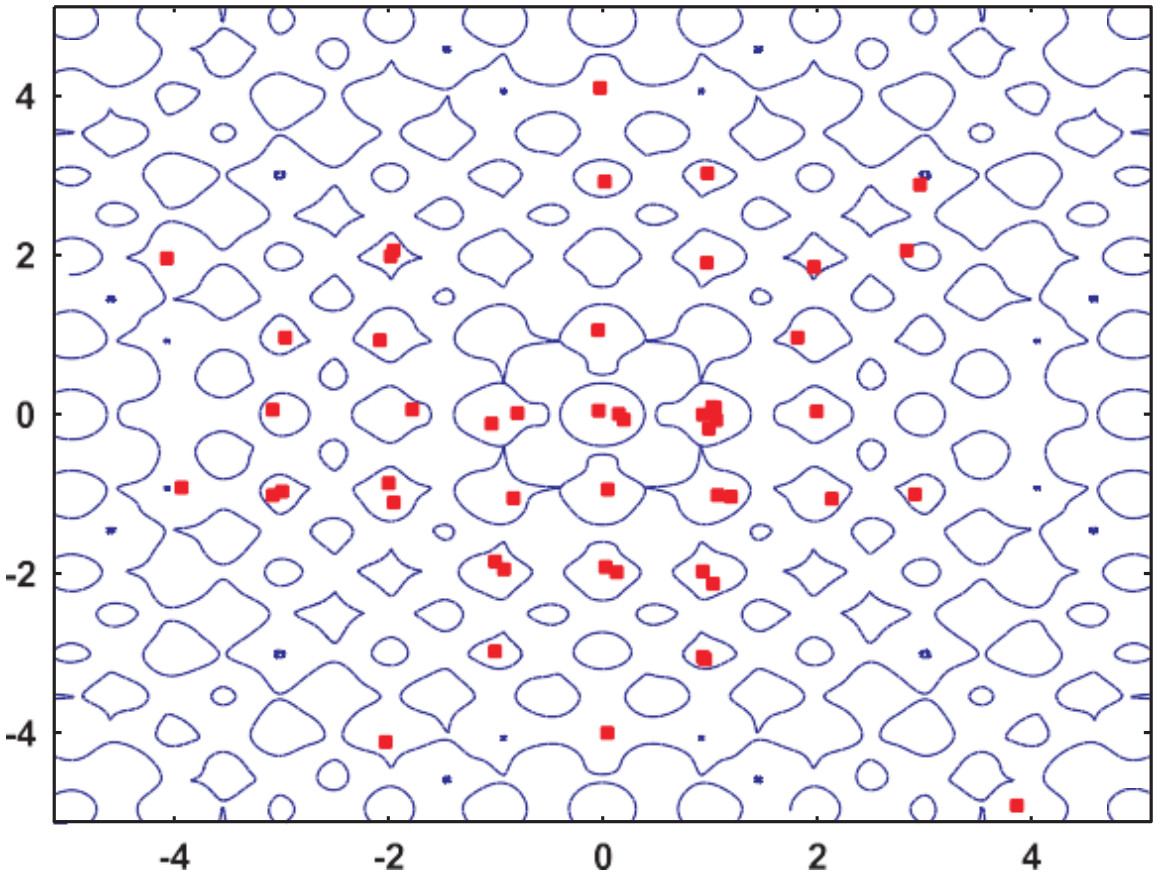
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 20: concentration around the center**



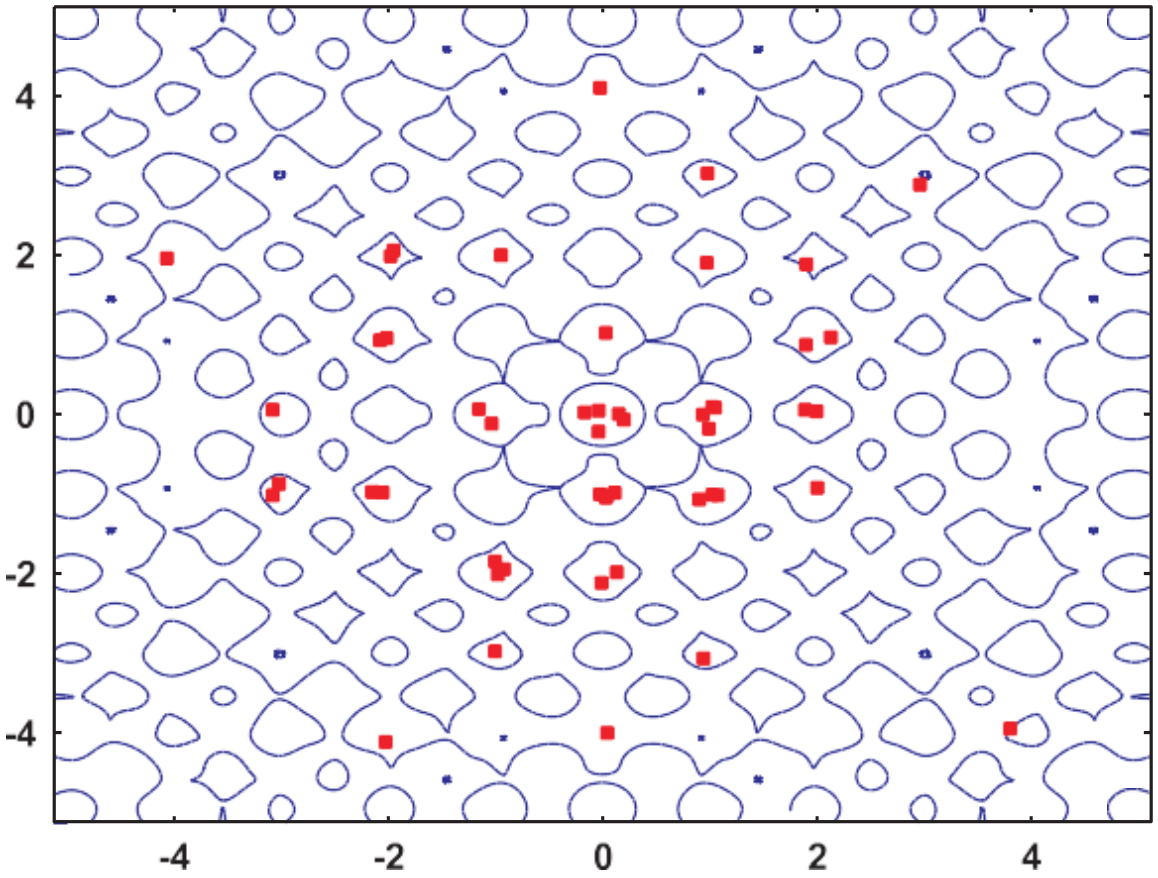
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 30 ..**



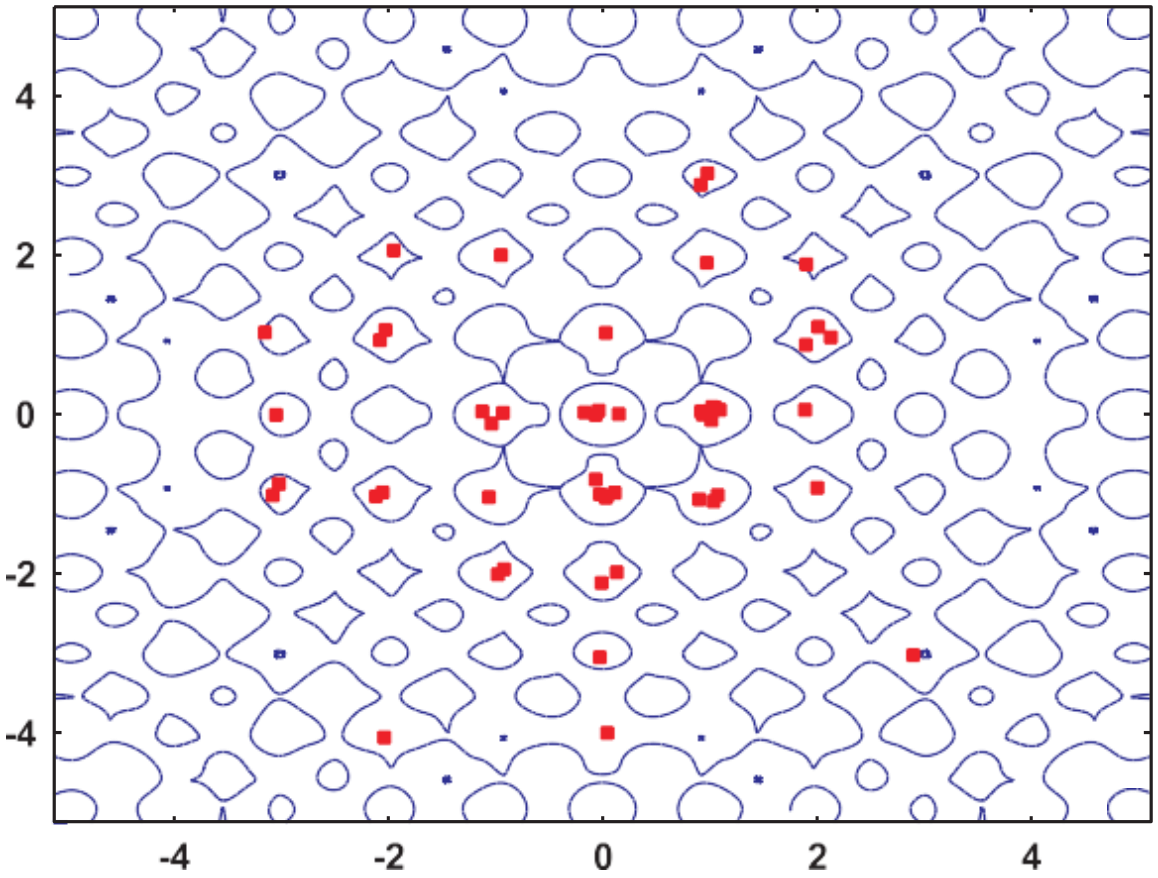
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 40 ..**



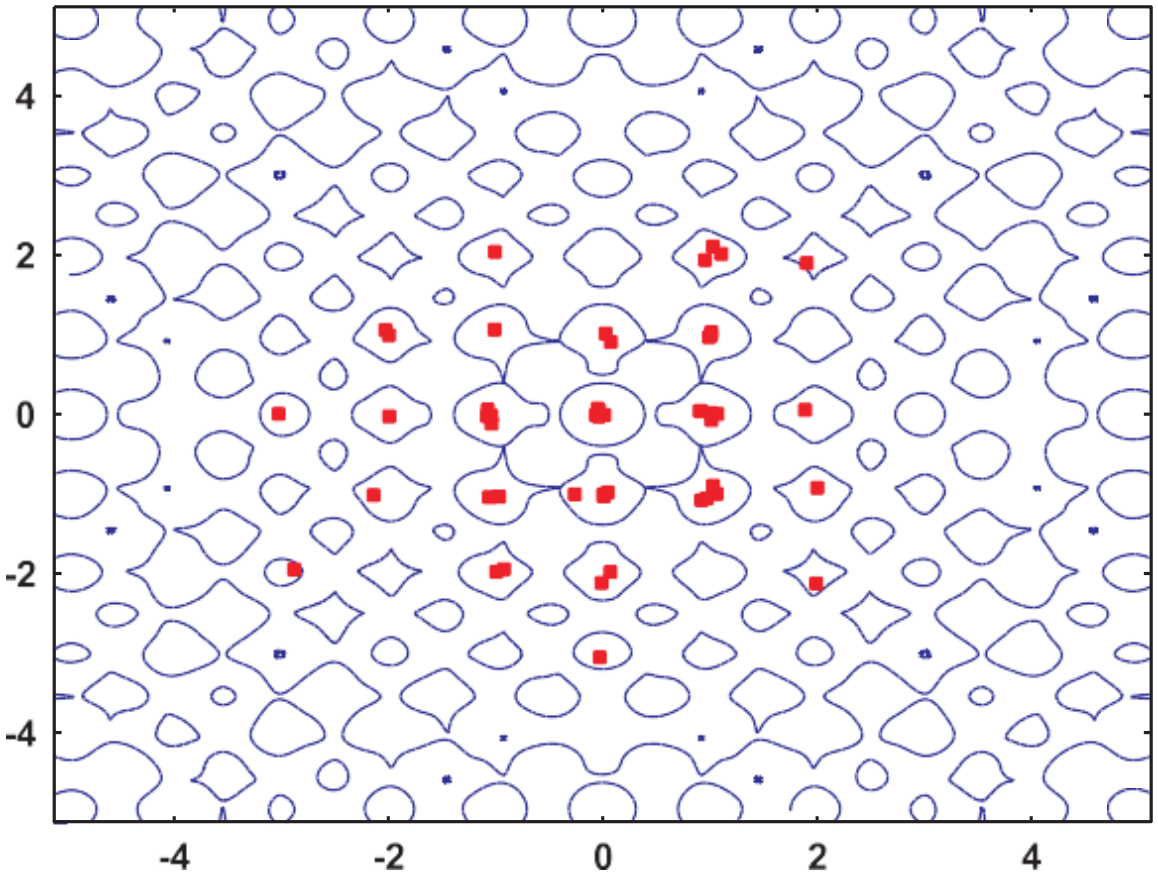
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 50 ..**



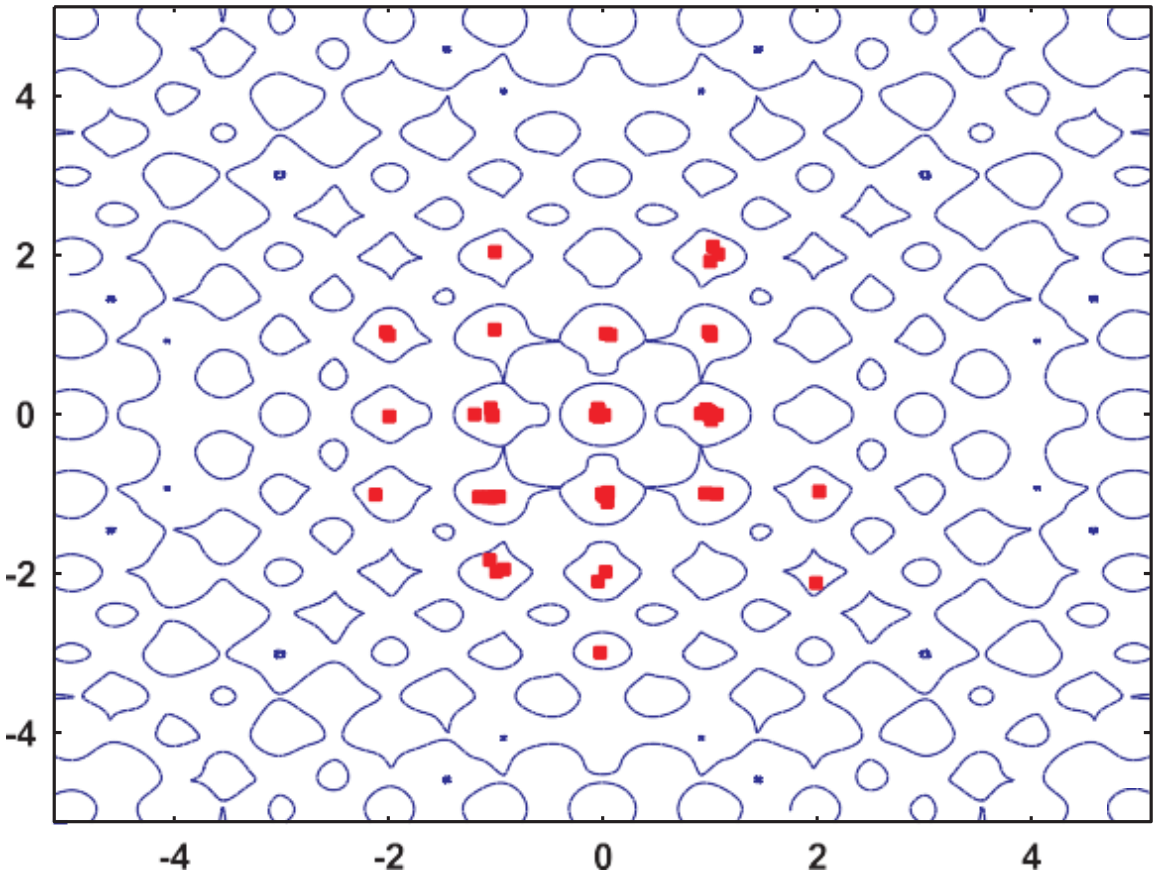
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 100 ..**



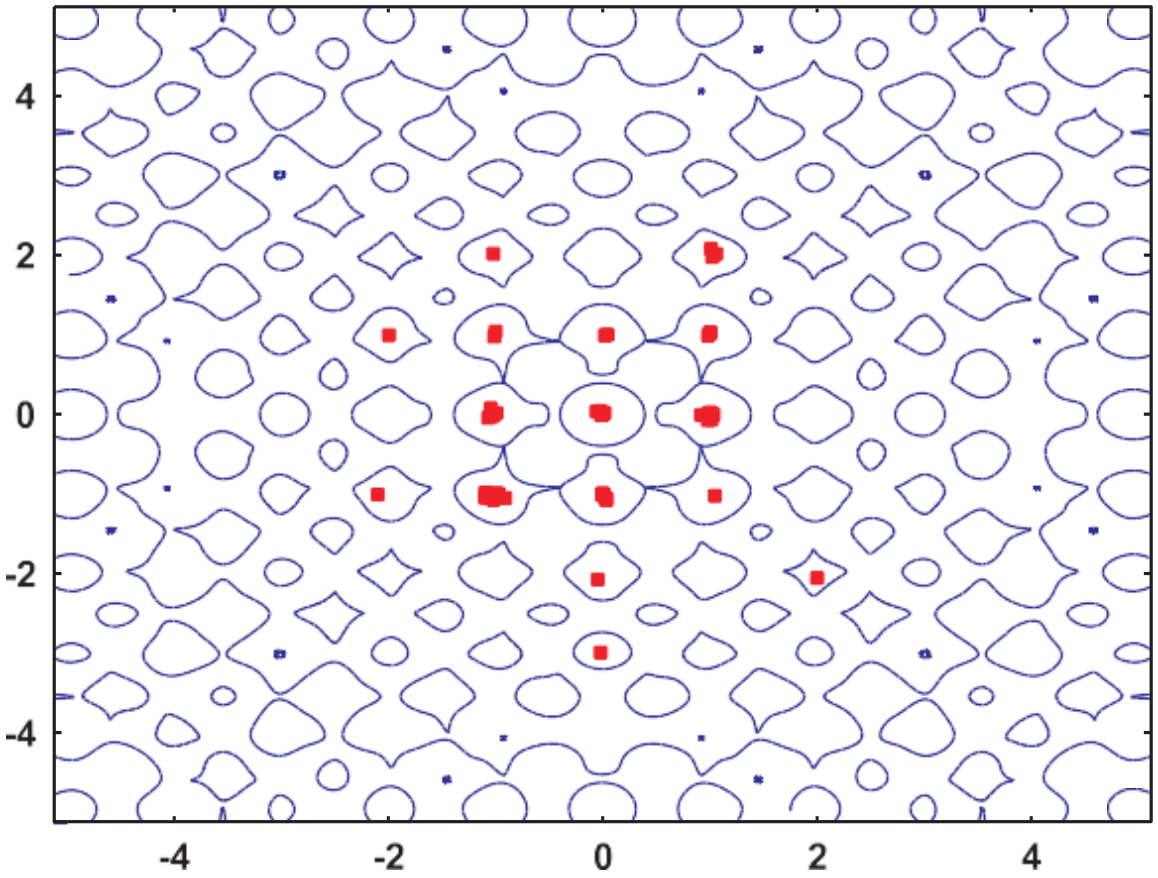
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 125: Similar individuals in local optima**



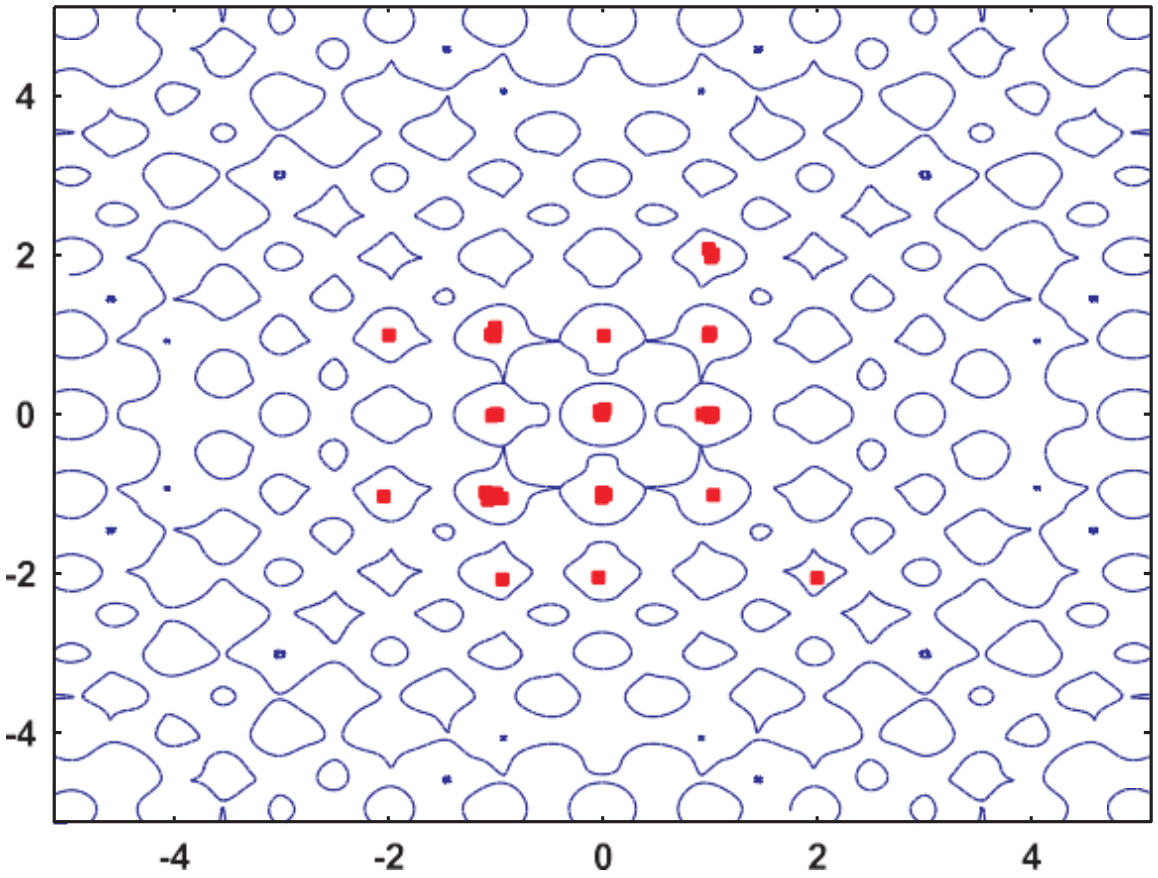
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 200 ..**



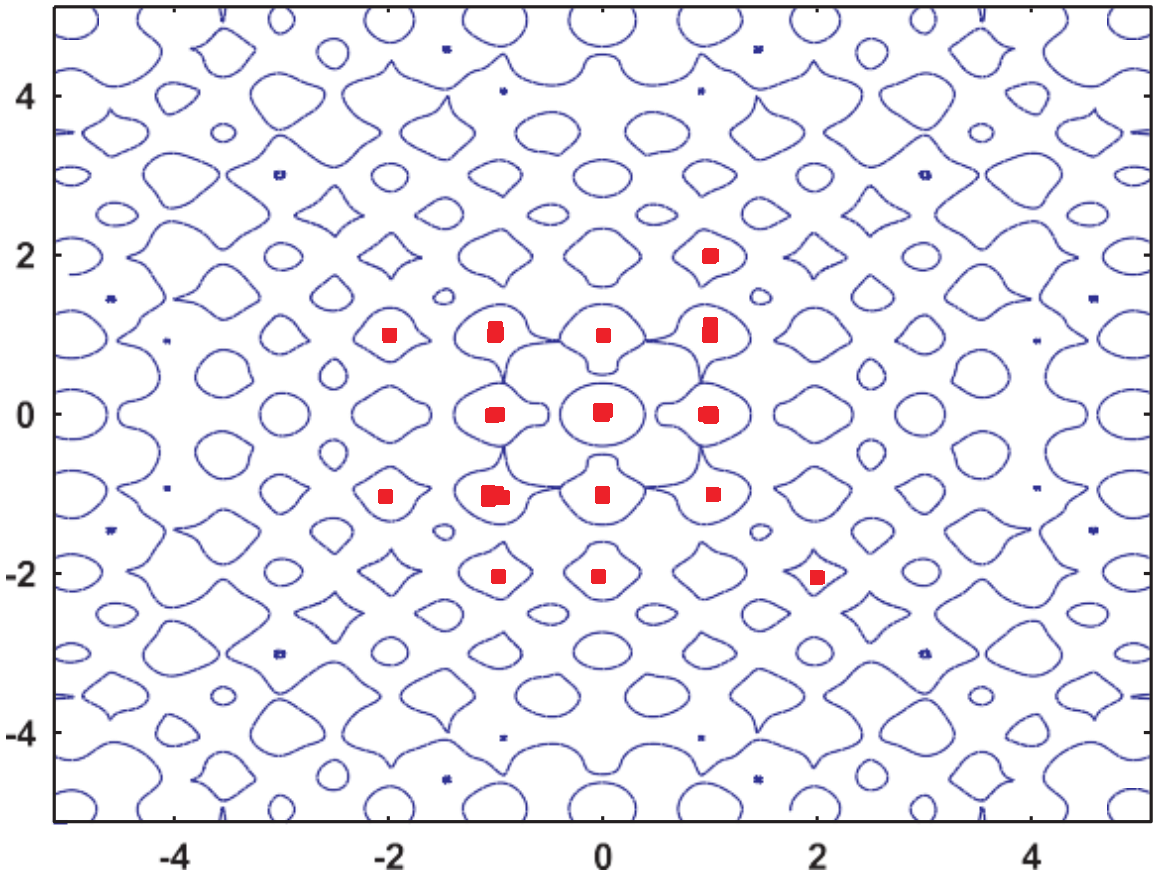
A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 250 ..**



A Numerical Optimization Example:
Minimizing a Function ..

- **Population after generation 300: Concentration at optima**



DE (*Differential Evolution*): *the* Algorithm

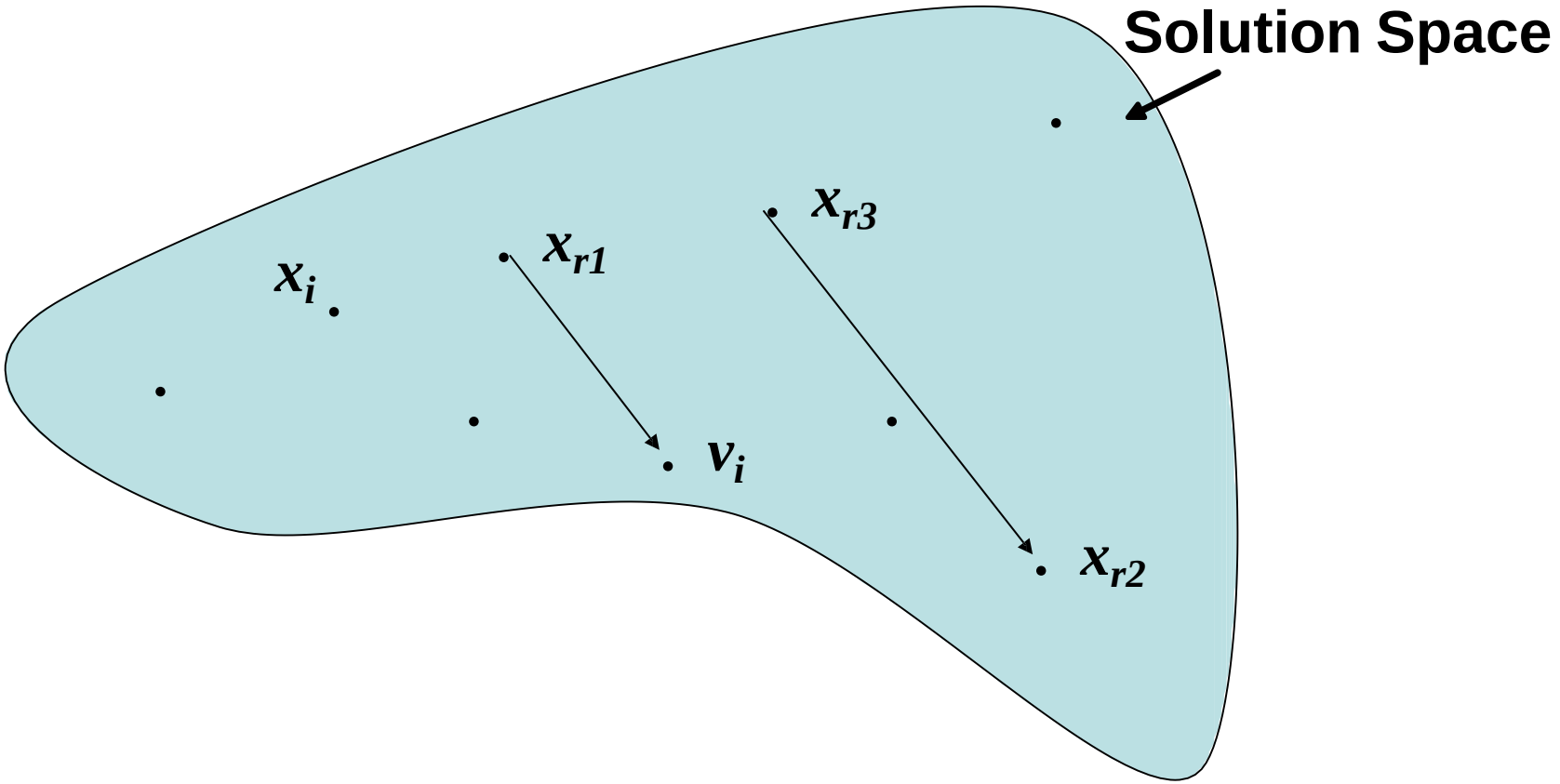
- Select four parents (p_1, p_2, p_3, p_4)
- Calculate the difference vector between two parents: $p_1 - p_2$
- Add the difference to the third parent vector with weight F:
  - $p_3 + F(p_1 - p_2)$
- Crossover between the new vector and fourth parent with gene selection probability "crossover rate" CR
  - **If (Math.random() > CR)** $x_i = x_i (p_4)$
  - **else** $x_i = x_i (p_3 + F (p_1 - p_2))$
- Compare the new child with fourth parent (p_4), the more fit will reach the next generation.

DE: *the* Algorithm ..

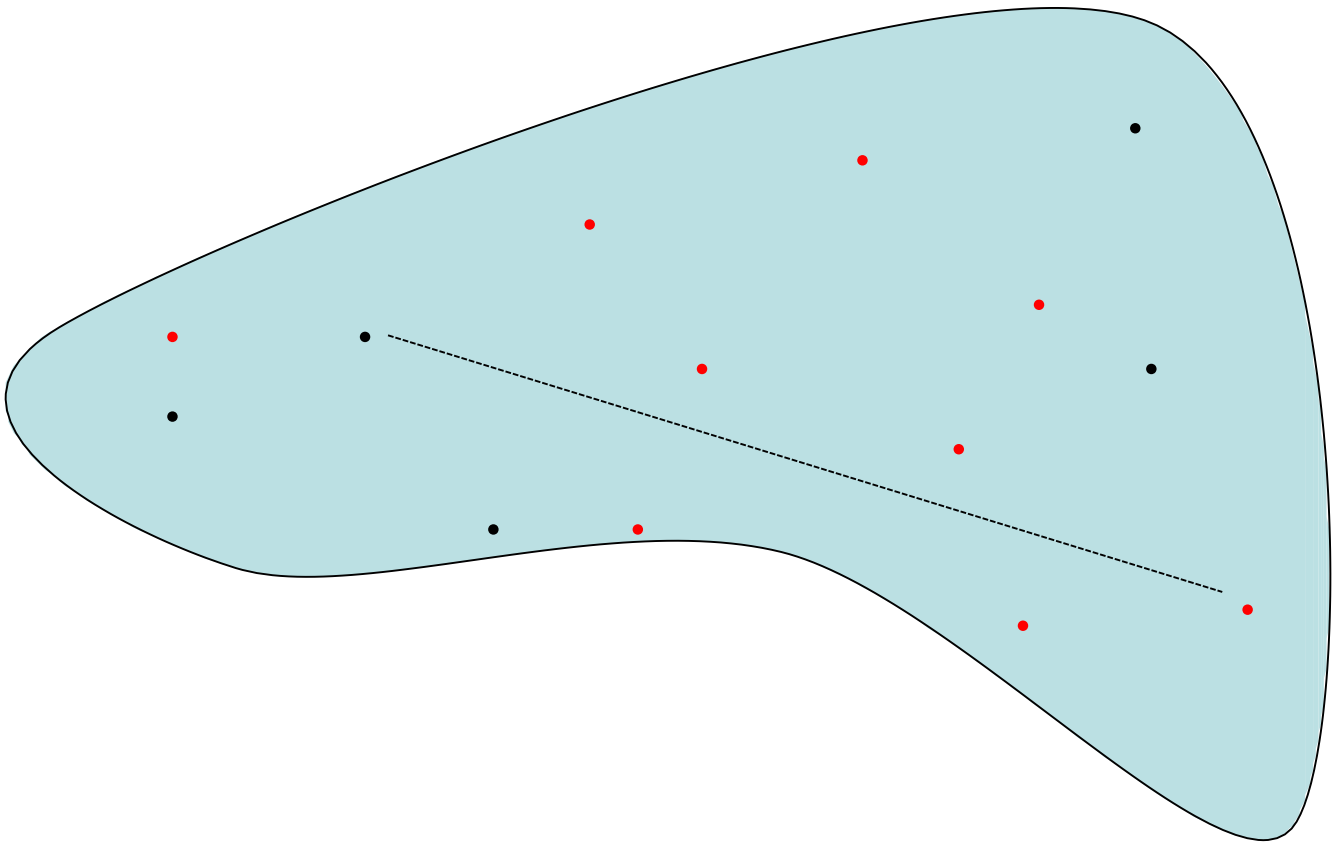
- Start with ***NP*** randomly chosen solution vectors.
- For each ***i*** in ***(1, .., NP)***, form a ‘mutant vector’
- $\boldsymbol{v_i = x_{r1} + F \cdot (x_{r2} - x_{r3})}$
- Where ***r1***, ***r2***, and ***r3*** are three mutually distinct randomly drawn indices from ***(1, .., NP)***, and also distinct from ***i***, and $\mathbf{0 < F \leq 2}$ *(F is a real and constant value).*

DE: *forming* a Mutant Vector ..

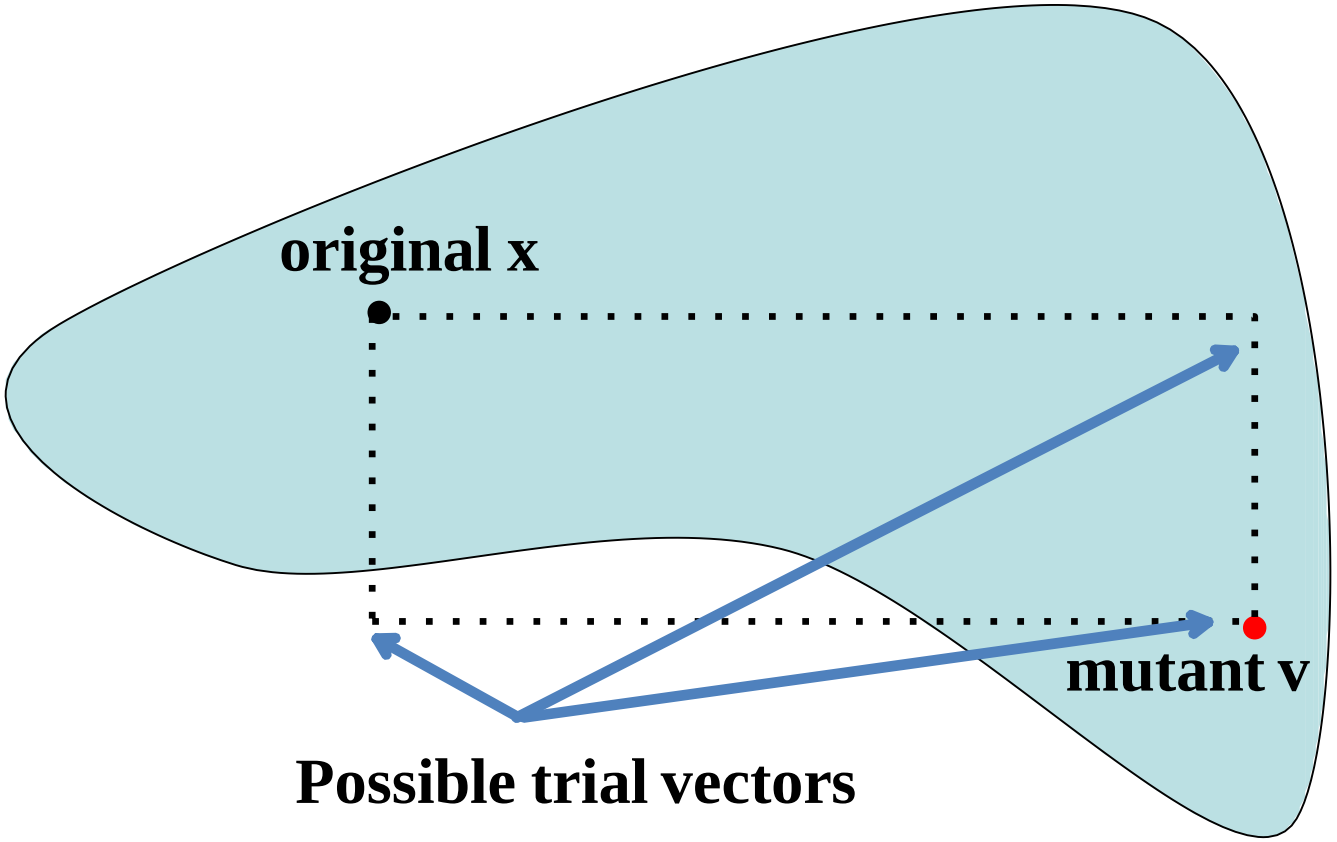
$$v_i = x_{r1} + F \cdot (x_{r2} - x_{r3})$$



DE: *from* Old Points to Mutants ..



DE: Crossover x_i & v_i to form a Trial Vector ..



DE: Crossover x_i & v_i to form a Trial Vector ..

$$x_i = (x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5})$$
$$v_i = (v_{i1}, v_{i2}, v_{i3}, v_{i4}, v_{i5})$$
$$u_i = (\_, \_, \_, \_, \_) \ ?$$

- For each component of the vectors, draw a random number in $U[0,1]$, call this $rand_j$.
- CR *(Crossover Rate)* is the crossover-constant / cutoff which has to be determined by the user. Let $0 < CR < 1$.
- If $rand_j \leq CR$, then $u_{ij} = v_{ij}$, else $u_{ij} = x_{ij}$.
- To ensure at least some crossover, one (or more) component(s) of u_i is/are selected at random to be from v_i .
  - So, $rnbr_j$ is a randomly chosen index $\in$ 1, 2, .., D.
  - If $rnbr_j = j$, then $u_{ij} = v_{ij}$

DE: Crossover x_i & v_i to form a Trial Vector ..

$$x_i = (x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5})$$

$$v_i = (v_{i1}, v_{i2}, v_{i3}, v_{i4}, v_{i5})$$

E.g., Index 1 is randomly selected as a definite crossover.

So, for example, we may have:

$rand_5$ <= CR, so it crossed over too.

$$u_i = (v_{i1}, x_{i2}, x_{i3}, x_{i4}, v_{i5})$$

DE: *the* Selection ..

If the objective-value *(fitness-function)* COST(u_i) is better than COST(x_i)

then u_i replaces x_i in the next generation

otherwise, we keep x_i

DE: *an* Example ..

Parent 1:	0.12	0.65	0.45	0.32	0.77
Parent 2:	0.16	0.02	0.77	0.34	0.31
Difference *(Parent 1 – Parent 2)*:	-0.04	0.63	-0.32	-0.02	0.46
Difference * F (= 0.5 *for example*)	-0.02	0.32	-0.16	-0.01	0.23
Parent 3:	0.45	0.33	0.23	0.77	0.81
Donor Vector (Mutant Vector):	0.43	0.65	0.07	0.76	1.00* (*overflow)
Parent 4 (Target Vector):	0.23	0.77	0.43	0.88	0.96
New child (Trial Vector): *Crossover Mutant & Target Vectors*	0.43	0.77	0.07	0.76	0.96

The *basic* DE algorithm ..

Initialize all agents *X* with random positions in search space

repeat

for each agent *X* in the population **do**

Pick three agents r_1, r_2, r_3

Pick a random index between 1 and *NP*

Compute the agent's potentially new position

for each component of the agent **do**

Copy or not copy based on the cases discussed earlier

end for .. end for

until termination criterion is met *(e.g. number of iterations performed, or adequate fitness reached).*

Pick the agent from the population that has the lowest cost and return it as the best found candidate solution.

The *highlights* of DE ..

- In the beginning, all candidate solutions are spread uniformly in search space X.
- The differential information (x_{r2}-x_{r3}) is large ⇒ large search steps.
- As the population converges, the candidate solutions are located closer to each other.
- The distances (x_{r2}-x_{r3}) decrease ⇒ the search steps get smaller too.
- Very simple way to perform self-adaptation without needing any additional parameter or operation.
- The step-width is self-organizing from the structure of the population and self-adapting along the optimization process.

The *highlights* of DE ..

- A population of NP solution vectors are successively updated by addition, subtraction, and component swapping, until the population converges, hopefully to the optimum.
- No derivatives are used.
- Very few parameters to set.
- A simple and apparently very reliable method.

Hybridization of EAs ..

- The common trend of EAs is hybrids between different evolutionary algorithms.
- These hybrids are relatively easy to do, since most EAs can use the same population table and the same gene presentation.
- Hybrids are mainly done in order to avoid the shortcomings of one algorithm:
  - – E.g. In the GA/DE hybrid, GA acts like a global searcher, and DE acts as a local searcher. Thus, GA finds peaks, and DE finds the highest point of that peak.

More information:

- DE homepage *(including practical advice, source codes, etc.)*:

  http://www.icsi.berkeley.edu/~storn/code.html

- Practical Advice:

  *(e.g., start with NP = 10 * D, and CR = 0.9, F = 0.8)*

Thanks! … *Questions?*