CS 581

Advanced Artificial Intelligence

January 31, 2024

Announcements / Reminders

- **Please follow the Week 04 To Do List instructions (if you haven't already)**
- **Written Assignment #01: to be posted soon**
- **Programming Assignment #01: to be posted soon**

Teaching Assistants

Name	e-mail	Office hours
Gawade, Vishal	vgawade@hawk.iit.edu	Tuesdays 12:30 PM - 01:30 PM CST in SB 108
Zhou, Xiaoting	xzhou70@hawk.iit.edu	Thursdays: 10:00 AM - 11:00 AM CST
Sandhu, Sukhmani	ssandhu3@hawk.iit.edu	**GRADING ONLY / NO OFFICE HOURS** (email if needed)

TAs will:

- assist you with your assignments,
- hold office hours to answer your questions,
- grade your assignments (**a specific TA will be assigned to you**).

Take advantage of their time and knowledge!

DO NOT email them with questions unrelated to lab grading.

Make time to meet them during their office hours.

Add a **[CS581 Spring 2024]** prefix to your email subject when contacting TAs, please.

Plan for Today

- Solving problems by Searching
  - Local Search Algorithms with memory
    - Local Beam Search
    - Tabu Search
  - Evolutionary Algorithms
    - Basic Genetic Algorithm

Local Beam Search

Local Beam Search: the Idea

The local beam search algorithm:

- **keeps track of k states rather than just one**
- **begins with k randomly generated states**
- **at each step, all the successors of all k states are generated.**
  - **if any one is a goal, the algorithm halts**
- **otherwise, it selects the k best successors from the complete list and repeats**

Local Beam Search: the Idea

In a local beam search useful information is passed among the k parallel search threads

For example, if one state generates several good successors and the other k–1 states all generate bad successors, then the effect is that the first state says to the others, "Come over here, the grass is greener!" The algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made

Tabu Search

Tabu Search: Key Features

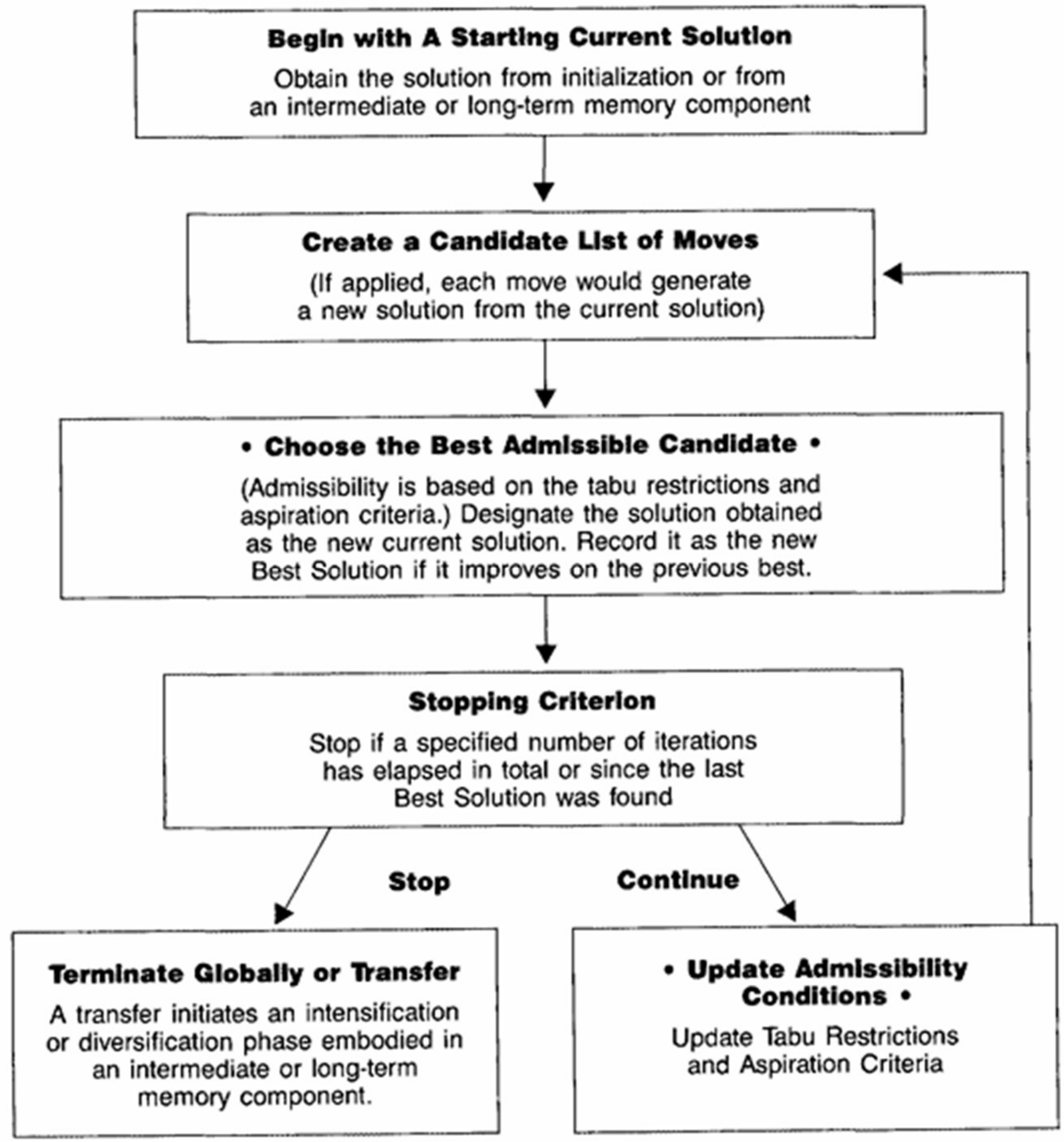
- **Always move to the best available neighborhood solution, even if it is worse than the current solution**
- **Maintain a list of solution points that must be avoided (not allowed) or a list of move features that are not allowed:**
  - **this is the Tabu List.**
- **Update the Tabu List based on some memory structure (short-term memory):**
  - **remove tabu moves after some time period has elapsed (Tenure).**
- **Allow for exceptions from the tabu list**
  - **Aspiration Criteria**
- **Expand the search area:**
  - **modify Tenure or Tabu List size**

Tabu Search: Memory Structures

The memory structures used in tabu search can be divided into three categories:

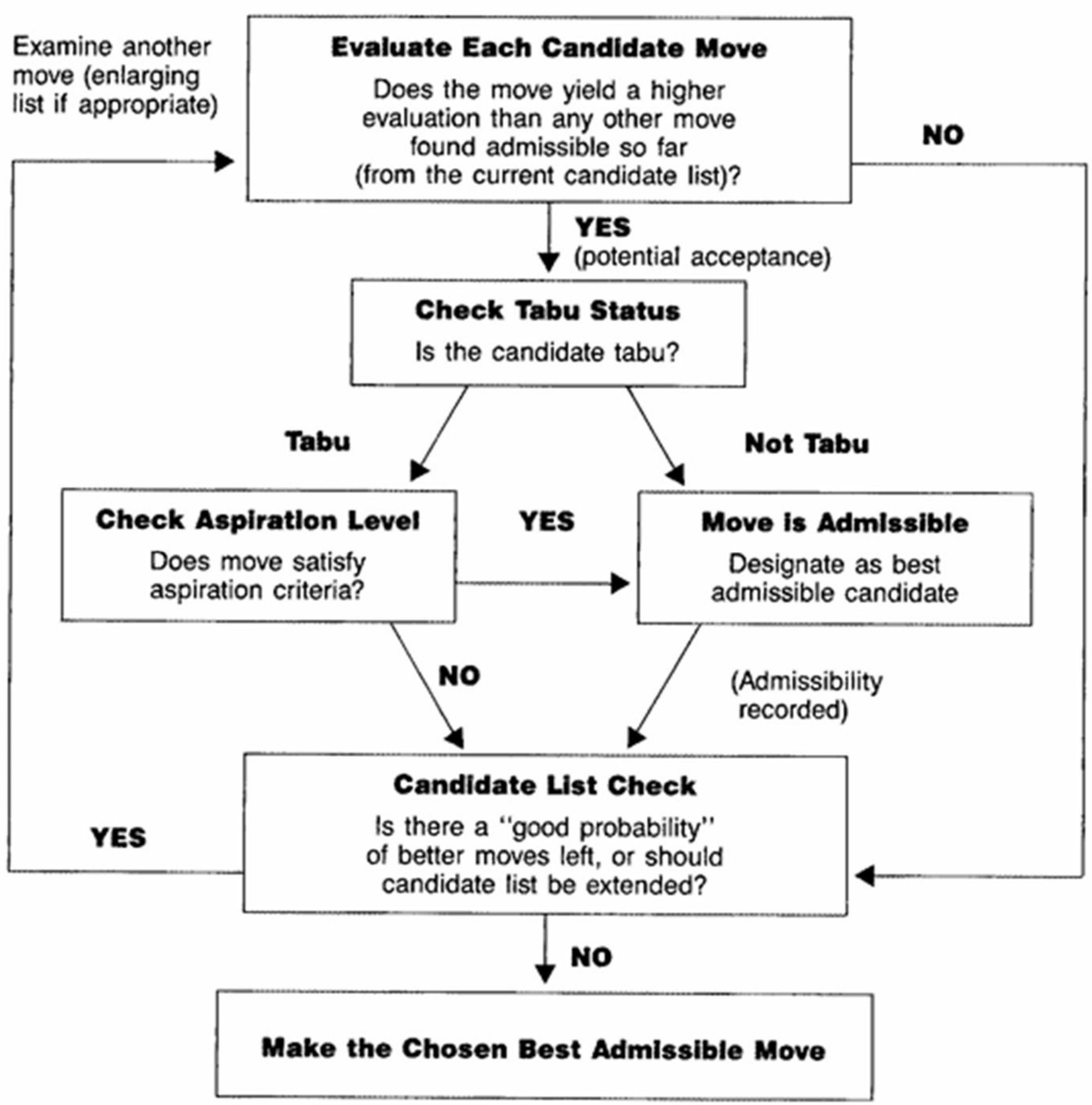
- **Short-term**: The list of solutions recently considered. If a potential solution appears on this list, it cannot be revisited until it reaches an expiration point (**Tenure**).
- **Intermediate-term**: A list of rules intended to bias the search towards promising areas of the search space.
- **Long-term**: Rules that promote diversity in the search process (i.e. regarding resets when the search becomes stuck in a plateau or a suboptimal dead-end).

Tabu Search: Short Memory Part



Source: Fred Glover – "Tabu Search: A Tutorial"

Tabu Search: Choose Admissible Move



Source: Fred Glover – "Tabu Search: A Tutorial"

Tabu Search: Aspiration Criteria

A criteria which allows a tabu move to be accepted under certain conditions.

Most common aspiration criterion:

If the move finds a new best solution, then accept the move even if the move is tabu.

Tabu Search: Tenure

The Tabu Tenure is the number of iterations that a move stays in the Tabu List

- **too small - risk of cycling**
- **too large - may restrict the search too much**

Tabu Search: Intensification

Search parameters can be locally modified in order to perform intensification and/or diversification

Intensification: usually applied when no configurations with a quality comparable to that of stored elite configuration(s), have been found in the last iterations

- choice rules for neighborhood moves are locally modified in order to encourage move combinations and solution properties historically found to be good
- jump to or initiate a return to regions in the configuration space in which some stored elite solutions lie: these regions can then be searched more thoroughly

Tabu Search: Diversification

Search parameters can be locally modified in order to perform intensification and/or diversification

Diversification: encourages the system to examine unvisited regions of the configuration space and thus to visit configurations that might differ strongly from all configurations touched before

- Random perturbation after stagnation or long-term cycling
- Coupling intensification and diversification: instead of jumping to one of the stored elite configurations, the system jumps to a configuration that has been created by changing one of the elite configurations in some significant way: slightly enough to search the neighborhood of the elite configuration and strongly enough so that the new configuration contains properties that are not part of the elite configuration anymore

Tabu Search: Stopping Criteria

Potential Stopping Criteria:

- **Number of iterations**
- **Number of iterations without improvement**
- **Execution time**
- **Objective function "good enough" threshold passed**

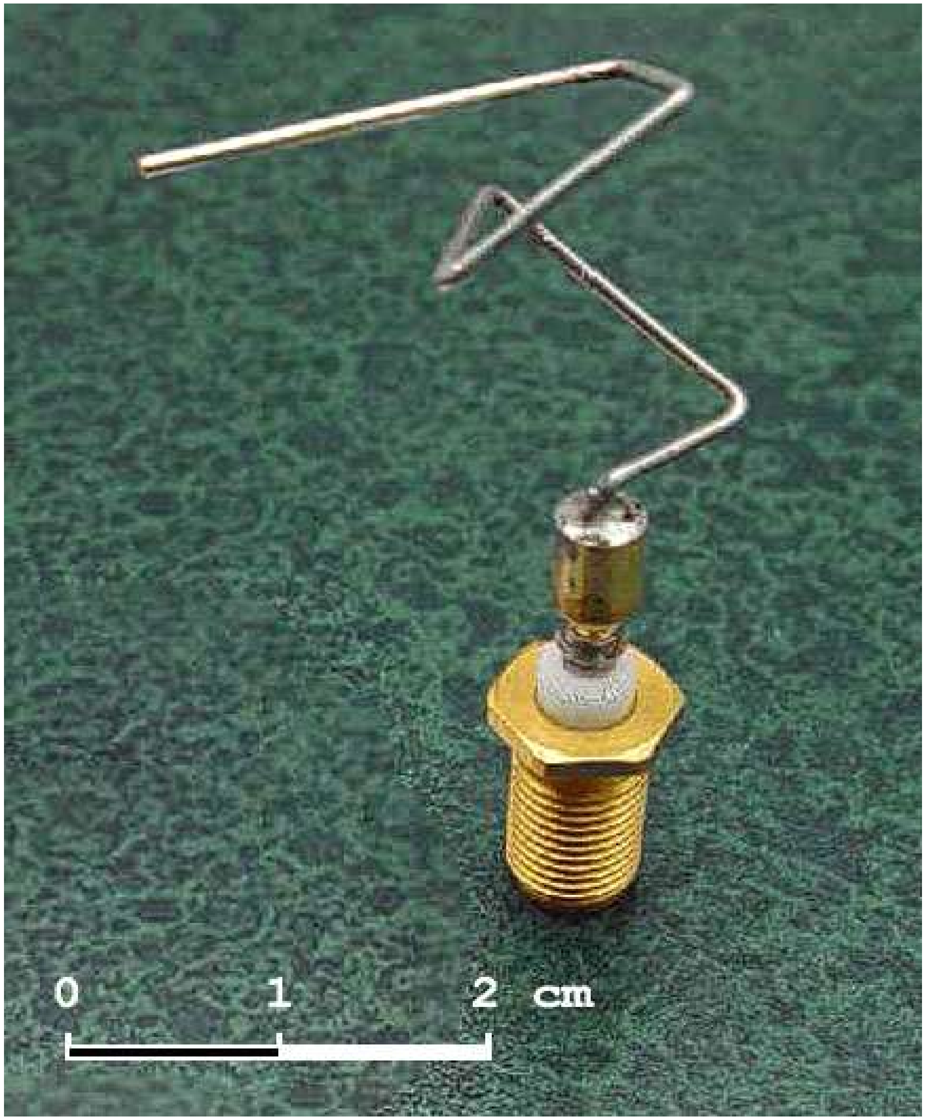
Tabu S: Advantages and Disadvantages

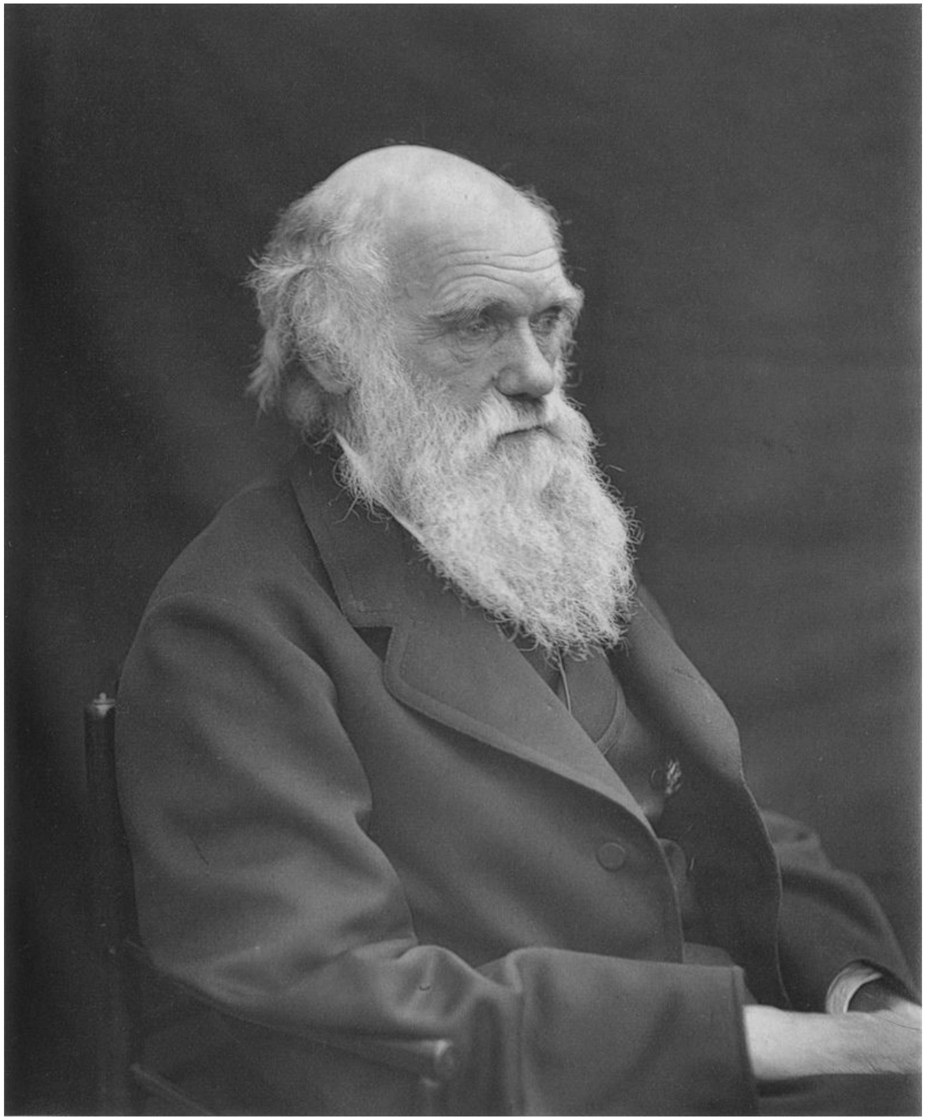
- Advantages:
  - Allows non-improving solution to be accepted in order to escape from a local optimum (similar to Simulated Annealing)
  - Keeping the Tabu list (prevents cycles and move reversals)
  - Works with both discrete and continuous solution spaces
  - For larger and more difficult problems (scheduling, vehicle routing, etc.), tabu search obtains solutions that rival and often surpass the best solutions previously found by other approaches

- Disadvantages:
  - Quite a few parameters to be determined
  - Number of iterations could be very large
  - Global optimum may not be found, depends on parameter settings

Evolutionary Algorithms

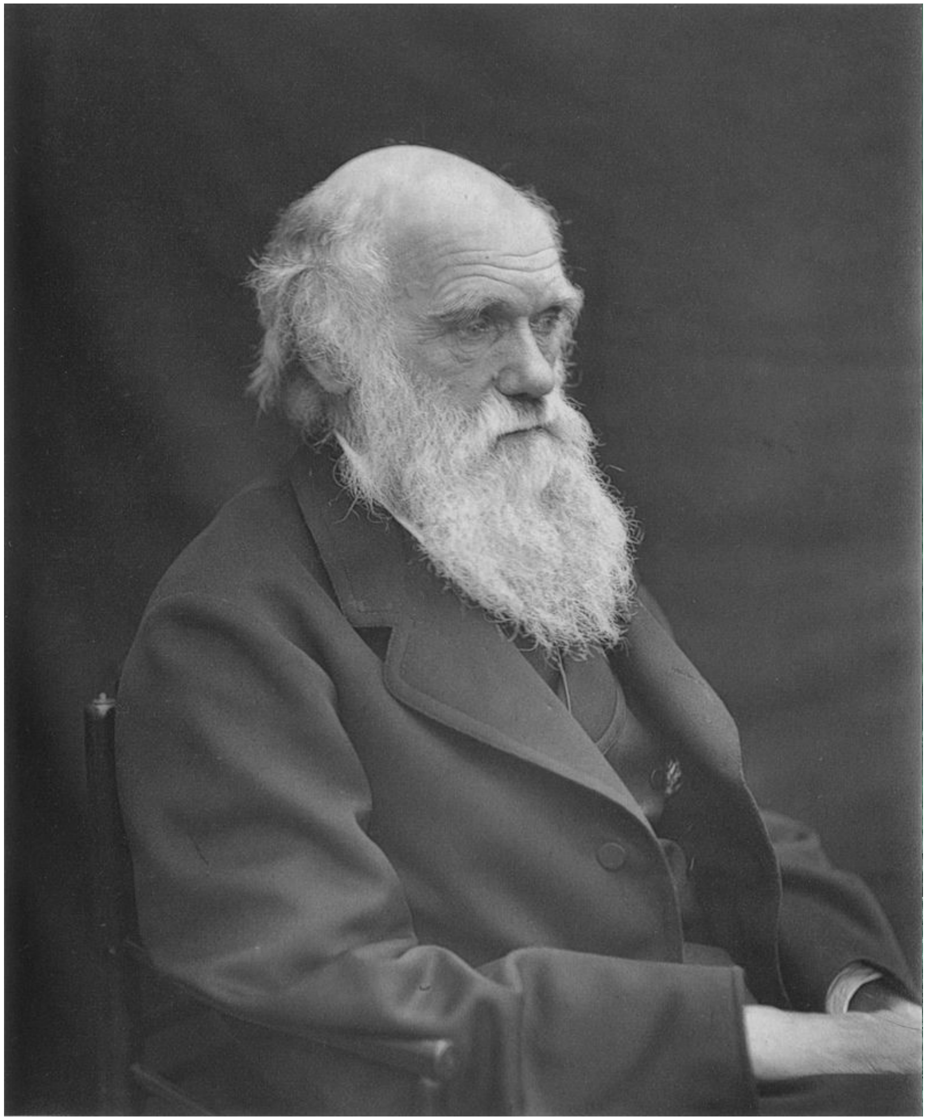
What’s the Connection Here?





Source: https://wikipedia.org/

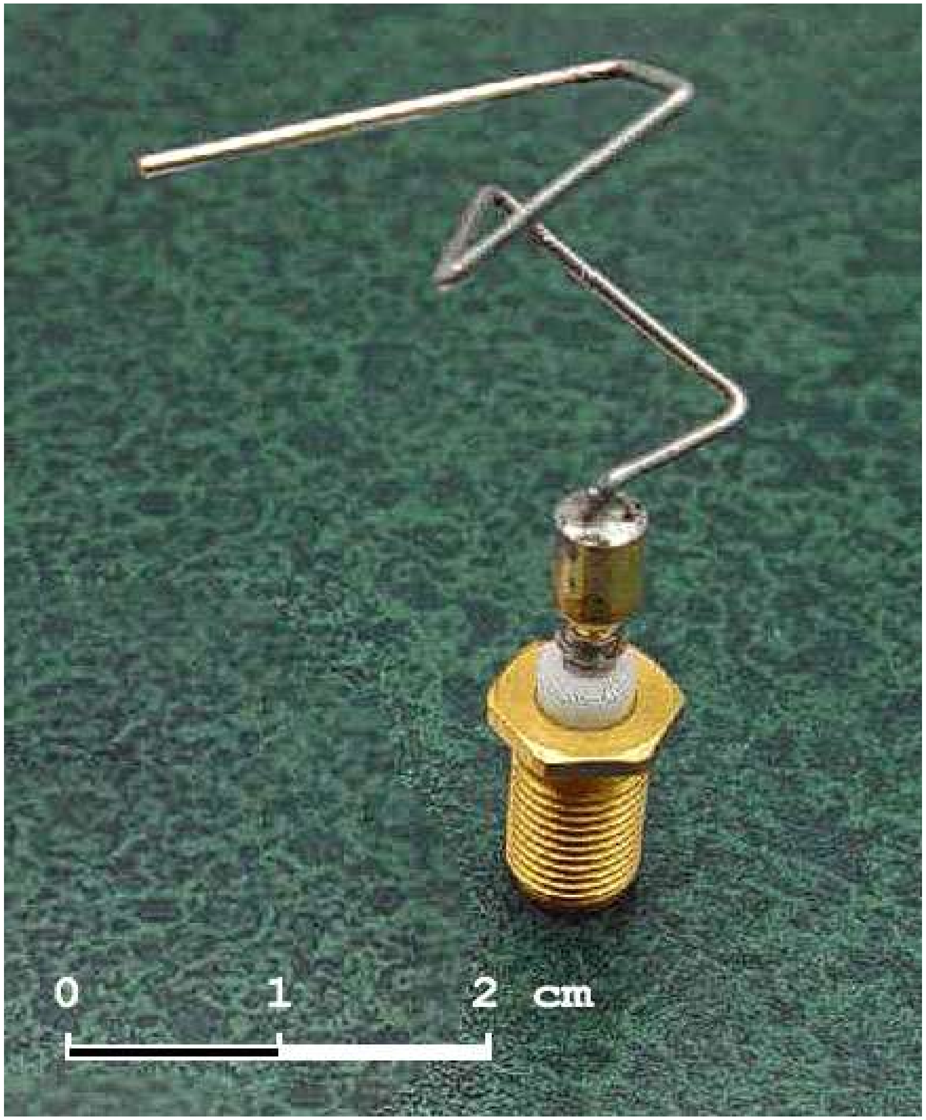
Charles Darwin



Source: https://wikipedia.org/

Charles Robert Darwin was an English naturalist, geologist and biologist, best known for his contributions to the **science of evolution**. His proposition that all species of life have descended over time from common ancestors is now widely accepted, and considered a foundational concept in science.

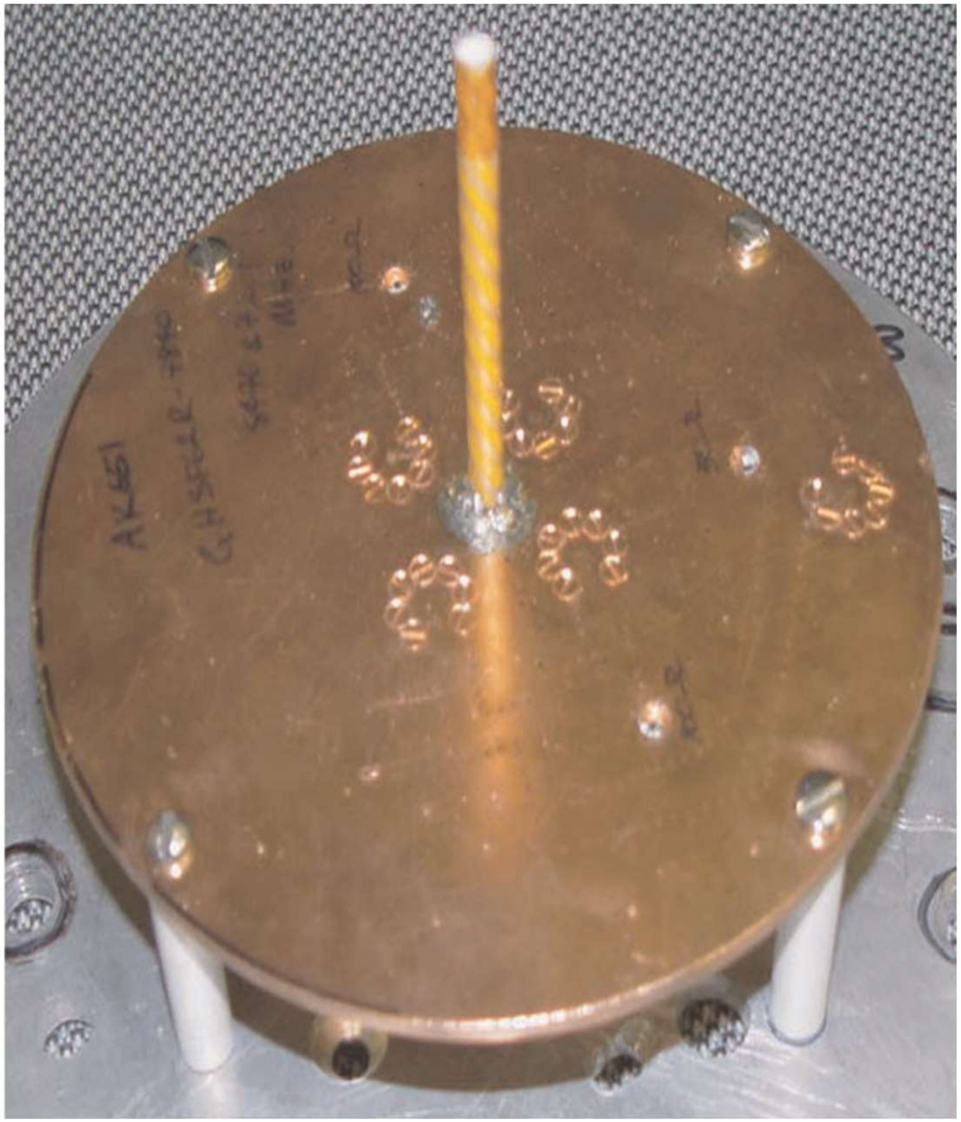
Evolved Antenna

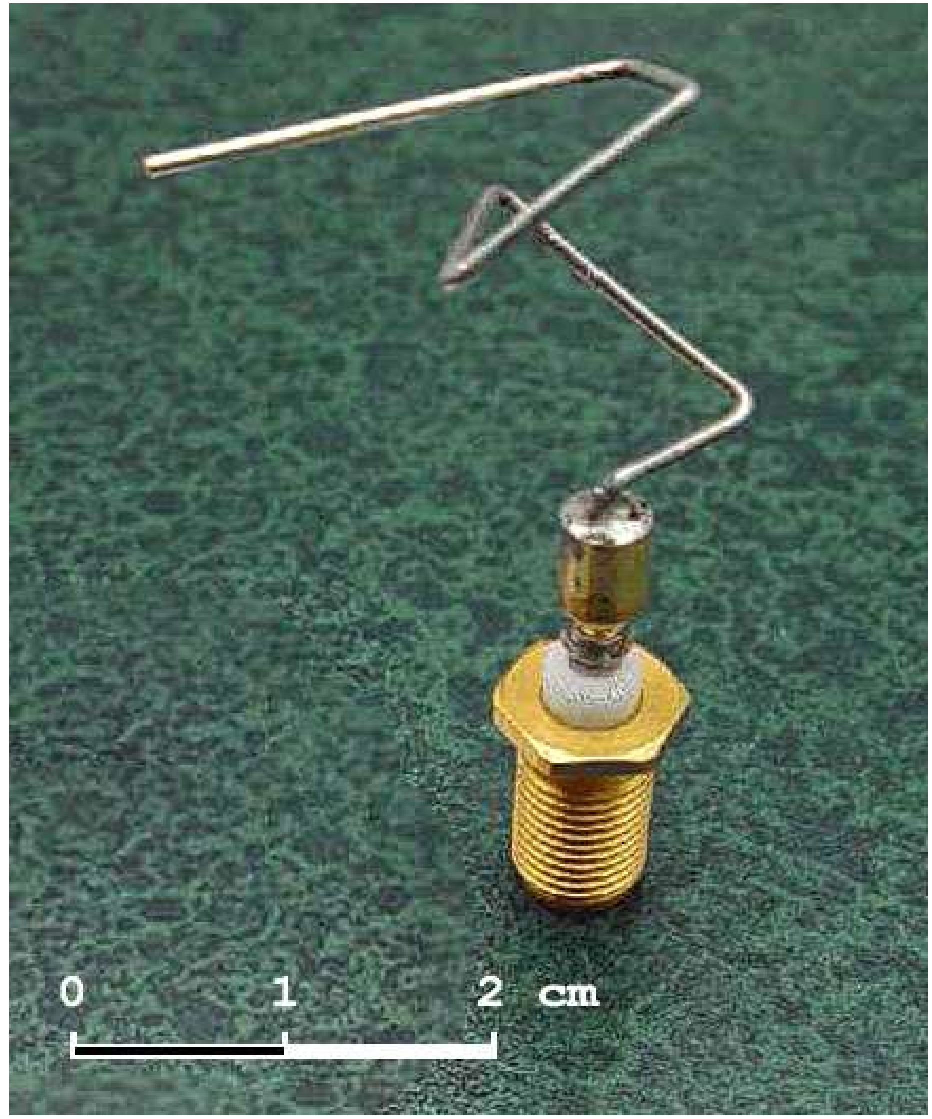


Source: https://wikipedia.org/

An evolved antenna is an antenna designed fully or substantially by an automatic computer design program that uses an evolutionary algorithm that mimics Darwinian evolution.

Engineered vs. Evolved Antenna





Source: Jason D. Lohn, Gregory S. Hornby, and Derek S. Linden - "Human-competitive evolved antennas"

Evolutionary Algorithms [Wikipedia]

An evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm.

An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions (see also loss function).

Evolution of the population then takes place after the repeated application of the above operators.

Biology and Evolutionary Algorithms Background

Chromosome

A chromosome is a package of DNA with part or all of the genetic material of an organism (source: Wikipedia).

It contains genes responsible for specific traits.

Artificial Chromosome

In Evolutionary Algorithms an artificial chromosome is a genetic representation of the task to be solved.

Typically:

1 individual = 1 chromosome = 1 solution

Also called a genotype.

Chromosome: Representation

Individuals / chromosomes can be represented as a string of values.

Typically:

Binary									
0	1	0	0	1	1	1	0	1	1

Integer									
2	1	11	2	3	78	1	0	111	33

Floating-point									
2.0	1.5	1.1	0.2	3.3	7.8	1.	0.0	11.1	3.3

Well-Suited Chromosome: Features

- It must allow the accessibility of all admissible points in the search space.
- Design the chromosome in such a way that it covers only the search space and no additional areas so that there is no redundancy or only as little redundancy as possible.
- Observance of strong causality: small changes in the chromosome should only lead to small changes in the phenotype. This is also called locality of the relationship between search and problem space.
- Design the chromosome in such a way that it excludes prohibited regions in the search space completely or as much as possible

Genotype vs. Phenotype

Genotype:

Organism's **full <u>hereditary information</u>**, even if not expressed. Directly inherited from parents.

Phenotype:

Organism's actual **observed properties**, such as morphology (structure), development, or behavior.

- Influenced by genotype
- Subject to environmental influence (including mutation)

Genes vs. Alleles

Genes:

Genes are **chunks of DNA** that contribute to particular **traits or functions**.

Alleles:

Alleles are **different versions of a gene**.

An **individual's combination of alleles** is known as **their genotype**.

- variations affect gene expressions: for example eye color

Chromosomes vs. Genes vs. Alleles

Species DNA structure

Gene 1			Gene 2			Gene 3			

Individual A chromosome

Gene 1			Gene 2			Gene 3			
0	1	0	1	1	0	1	0	1	1

Chromosomes
Genotypes
Individuals

Individual B chromosome

Gene 1			Gene 2			Gene 3			
1	0	1	1	1	0	1	0	1	1

Chromosomes vs. Genes vs. Alleles

Species DNA structure

Gene 1			Gene 2			Gene 3			

Individual A chromosome

Gene 1			Gene 2			Gene 3			
0	1	0	1	1	0	1	0	1	1

Alleles

Individual B chromosome

Gene 1			Gene 2			Gene 3			
1	0	1	1	1	0	1	0	1	1

Artificial Chromosome

Problem solution structure

Feature 1			Feature 2			Feature 3			

Individual A chromosome

Feature 1			Feature 2			Feature 3			
0	1	0	1	1	0	1	0	1	1

Encoded feature values

Individual B chromosome

Feature 1			Feature 2			Feature 3			
1	0	1	1	1	0	1	0	1	1

Artificial Chromosome

Problem solution structure

Variable 1			Variable 2			Variable 3			

Individual A chromosome

Variable 1			Variable 2			Variable 3			
0	1	0	1	1	0	1	0	1	1

Encoded variable values

Individual B chromosome

Variable 1			Variable 2			Variable 3			
1	0	1	1	1	0	1	0	1	1

Population

The set of solutions (individuals / chromosomes / genotypes) is called a population.

Example: Coordinates as Genes

Population of points (solutions)

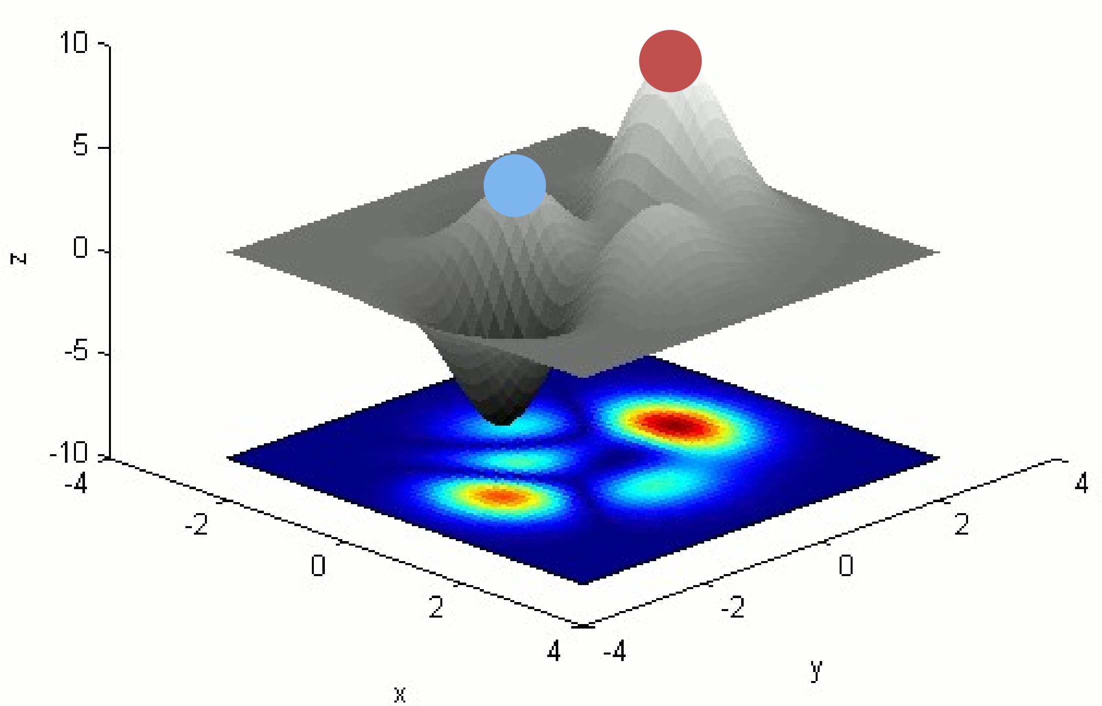
x			y		
0	1	0	1	0	1

x			y		
0	1	1	1	0	1

x			y		
0	0	0	1	0	1

x			y		
0	1	0	1	0	0

x			y		
0	0	1	0	0	0



y = f(x,y) - fitness function

"Good enough" fit / local maximum

Best fit / global maximum

Example: Coordinates as Genes

Individuals / Chromosomes / Genotypes

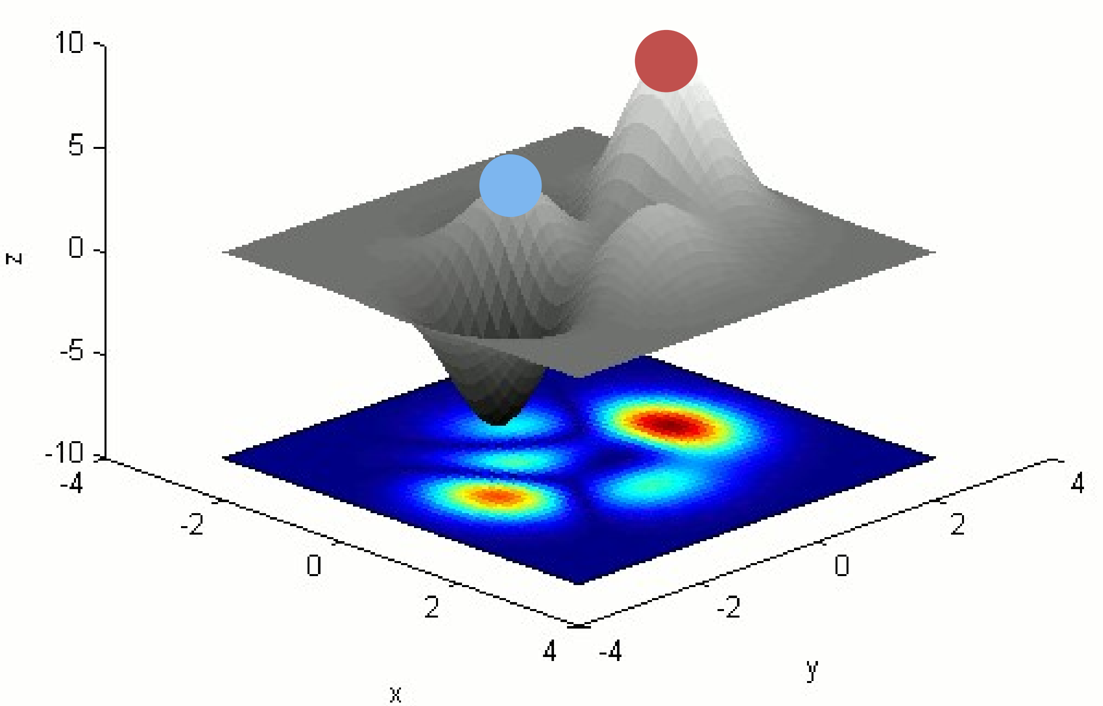
x			y		
0	1	0	1	0	1

x			y		
0	1	1	1	0	1

x			y		
0	0	0	1	0	1

x			y		
0	1	0	1	0	0

x			y		
0	0	1	0	0	0



y = f(x,y) - fitness function

"Good enough" fit / local maximum

Best fit / global maximum

Example: Coordinates as Genes

X gene

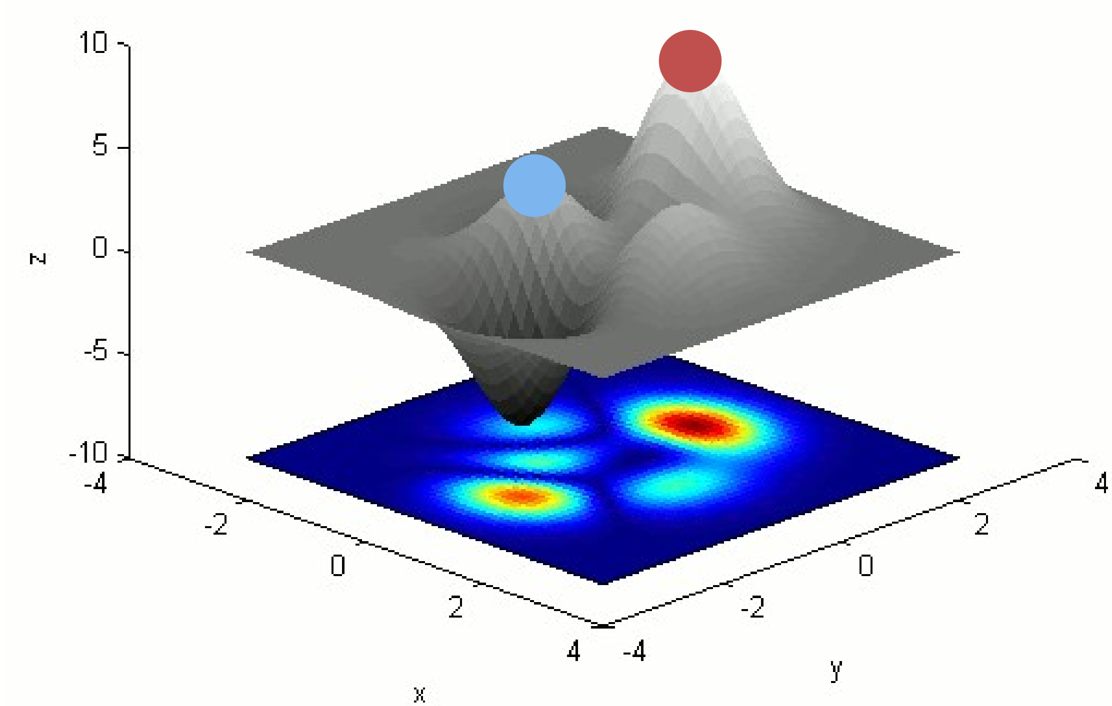
	x			y	
0	1	0	1	0	1

	x			y	
0	1	1	1	0	1

	x			y	
0	0	0	1	0	1

	x			y	
0	1	0	1	0	0

	x			y	
0	0	1	0	0	0



y = f(x,y) - fitness function

“Good enough” fit / local maximum

Best fit / global maximum

Example: Coordinates as Genes

Y gene

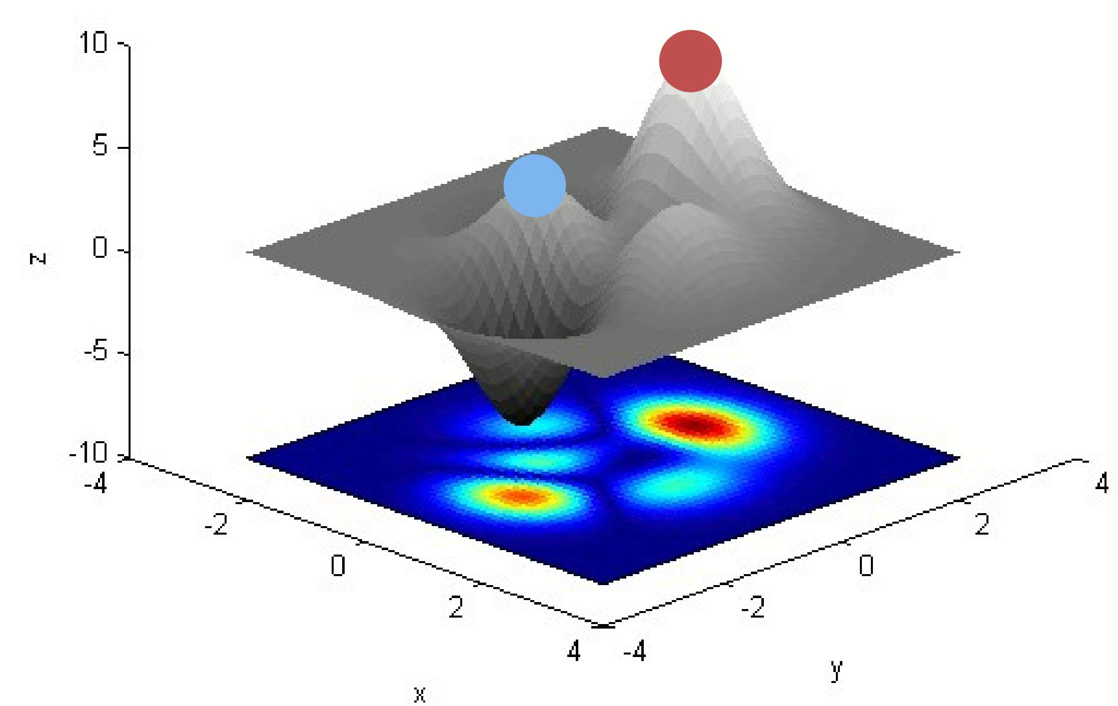
x			y		
0	1	0	1	0	1

x			y		
0	1	1	1	0	1

x			y		
0	0	0	1	0	1

x			y		
0	1	0	1	0	0

x			y		
0	0	1	0	0	0



y = f(x,y) - fitness function

"Good enough" fit / local maximum

Best fit / global maximum

Example: Coordinates as Genes

Population of points (solutions)

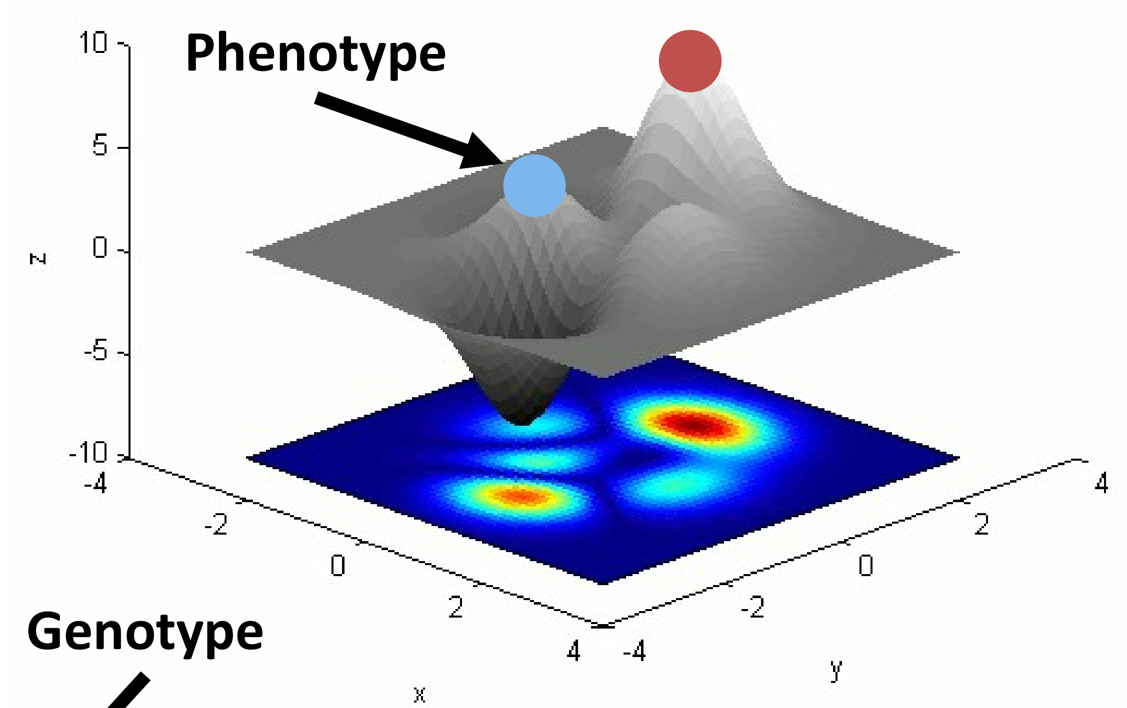
x			y		
0	1	0	1	0	1

x			y		
0	1	1	1	0	1

x			y		
0	0	0	1	0	1

x			y		
0	1	0	1	0	0

x			y		
0	0	1	0	0	0



Genotype

y = f(x,y) - fitness function

"Good enough" fit / local maximum

Best fit / global maximum

Genetic Algorithm

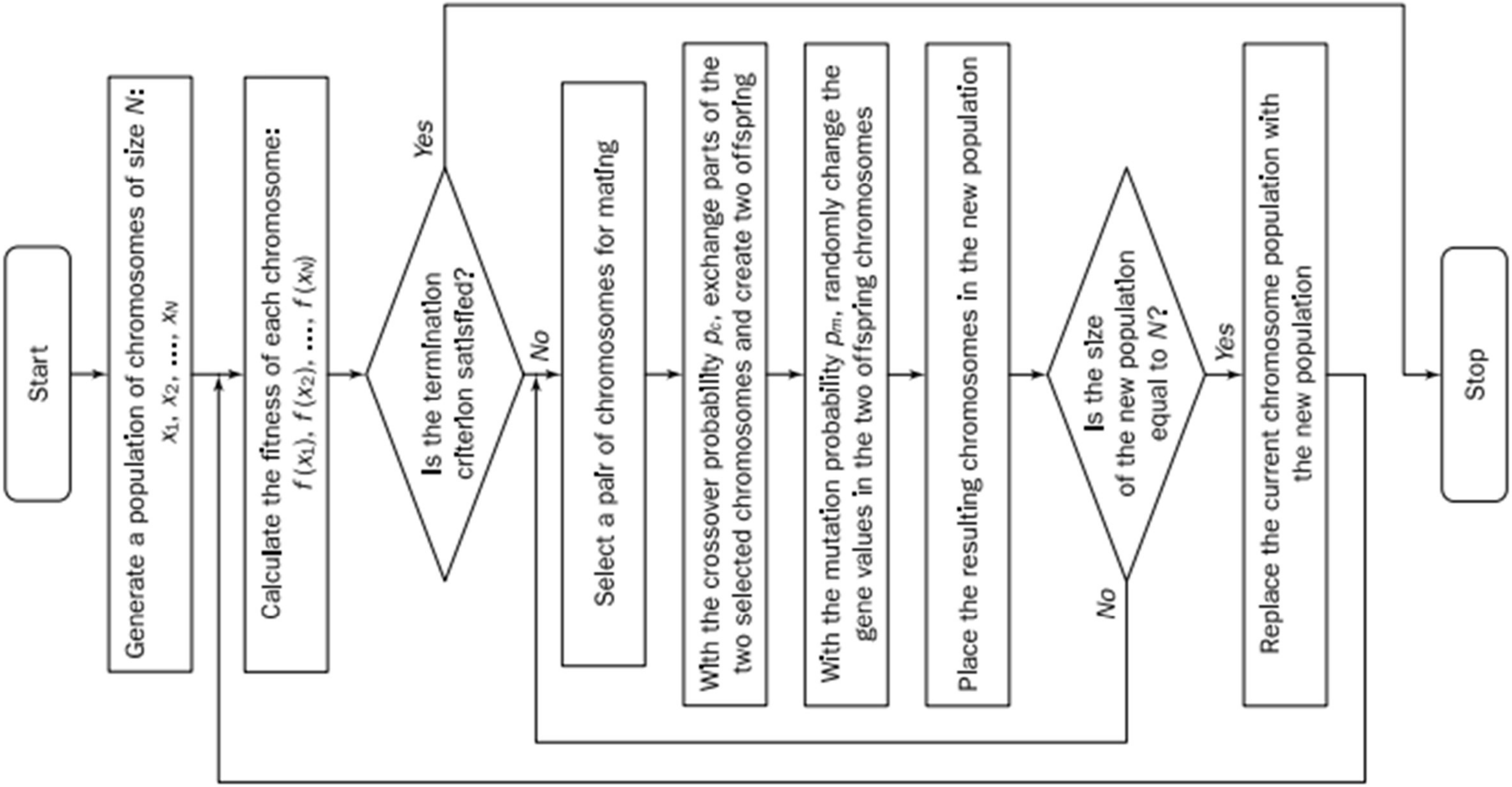
Genetic Algorithm: Roots

Directed search algorithms based on the concept of biological evolution

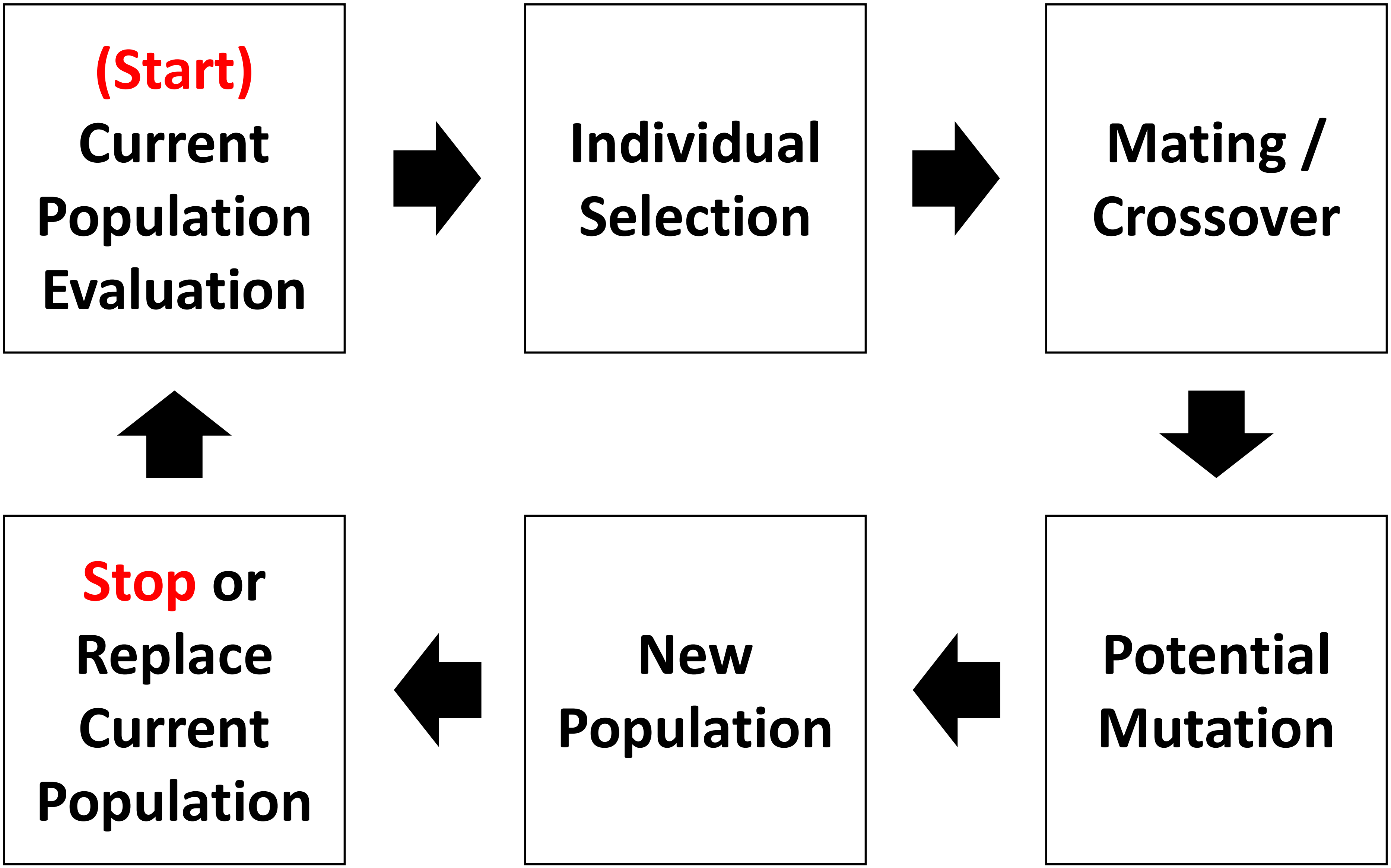
Developed by John Holland, University of Michigan (1970’s)

- **to understand the adaptive processes of natural systems**
- **to design artificial systems software that retains the robustness of natural systems**

Genetic Algorithm: Flowchart



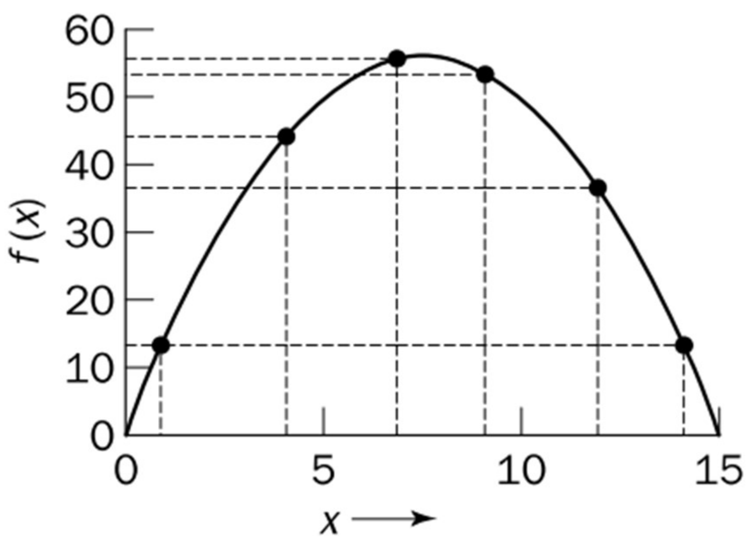
Genetic Algorithm: Process



Current Population Evaluation

Example Problem / Population

Individual	Chromosome				Decoded value	Individual fitness	Fitness ratio [%]
X1	1	1	0	0	12	36	16.5
X2	0	1	0	0	4	44	20.2
X3	0	0	0	1	1	14	6.4
X4	1	1	1	0	14	14	6.4
X5	0	1	1	1	7	56	25.7
X6	1	0	0	1	9	54	24.8

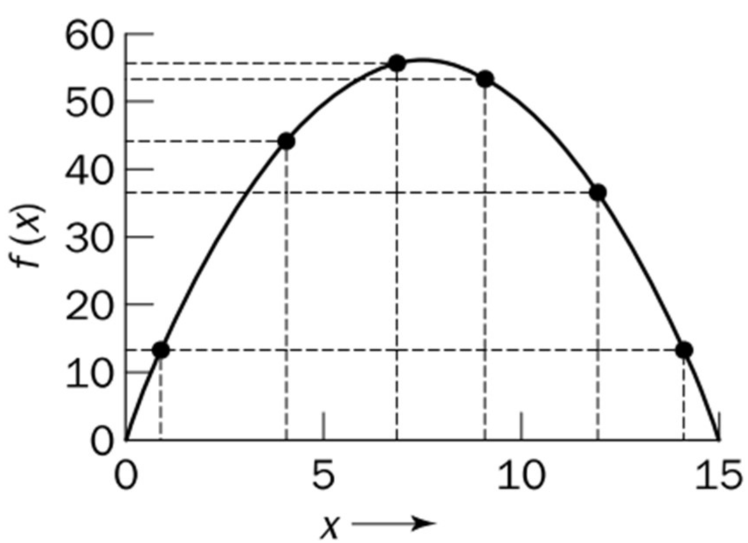


Fitness function f(x)

$$f(x) = 15x - x^2$$

Example Problem / Population

Individual	Chromosome				Decoded value	Individual fitness	Fitness ratio [%]
X1	1	1	0	0	12	Use the fitness function to calculate for each individual	16.5
X2	0	1	0	0	4		20.2
X3	0	0	0	1	1		6.4
X4	1	1	1	0	14		6.4
X5	0	1	1	1	7		25.7
X6	1	0	0	1	9		24.8

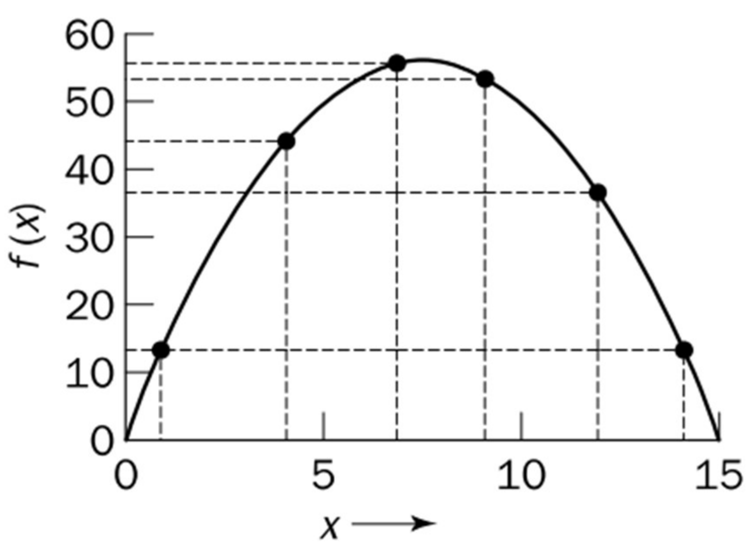


Fitness function f(x)

$$f(x) = 15x - x^2$$

Example Problem / Population

Individual	Chromosome	Decoded value	Individual fitness	Fitness ratio [%]
X1	1 1 0 0	12	36	Individual fitness / over total population fitness: $f(i)/\sum_{i}^{N} f(i)$
X2	0 1 0 0	4	44	
X3	0 0 0 1	1	14	
X4	1 1 1 0	14	14	
X5	0 1 1 1	7	56	
X6	1 0 0 1	9	54	

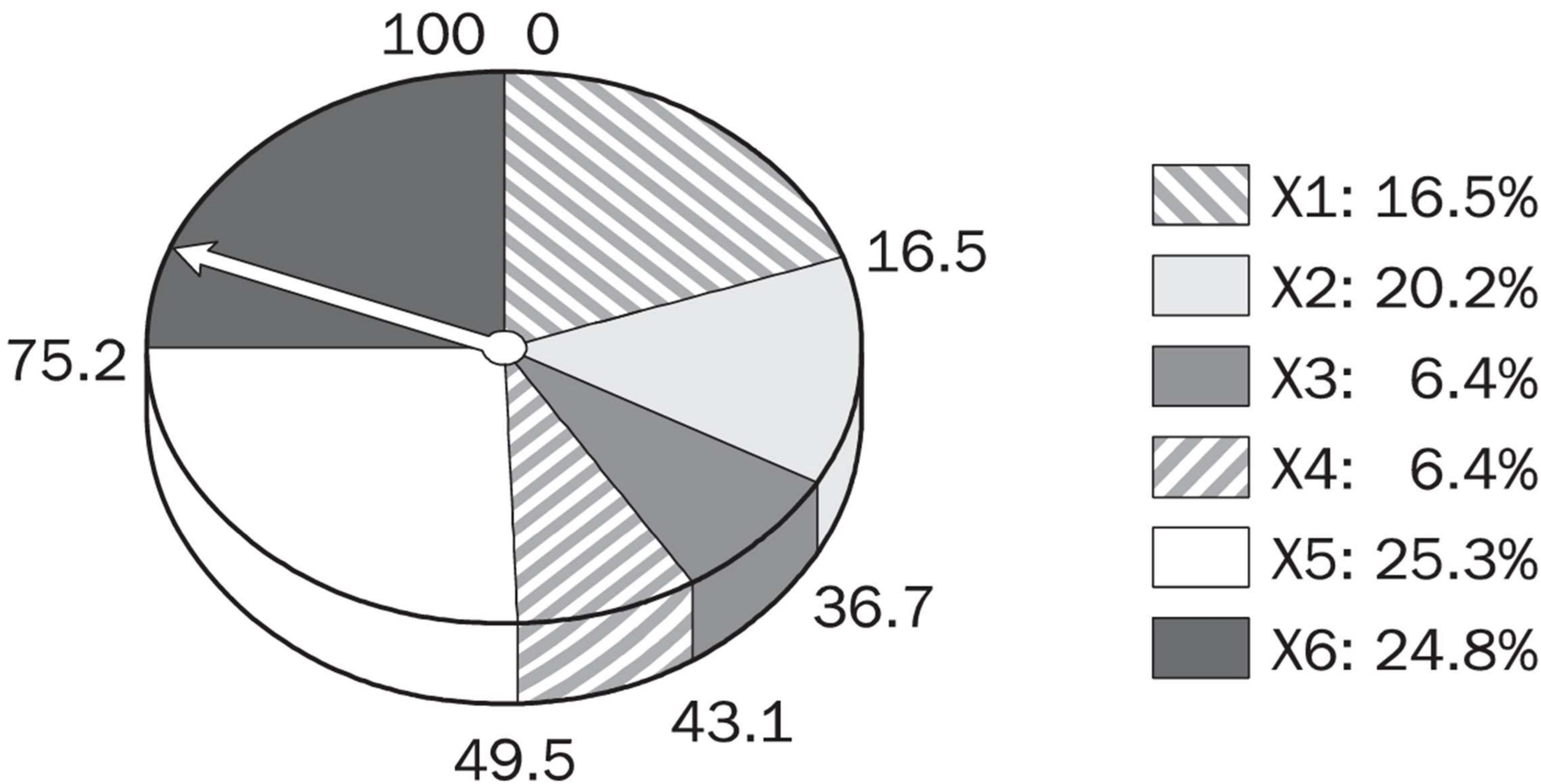


Fitness function f(x)

$$f(x) = 15x - x^2$$

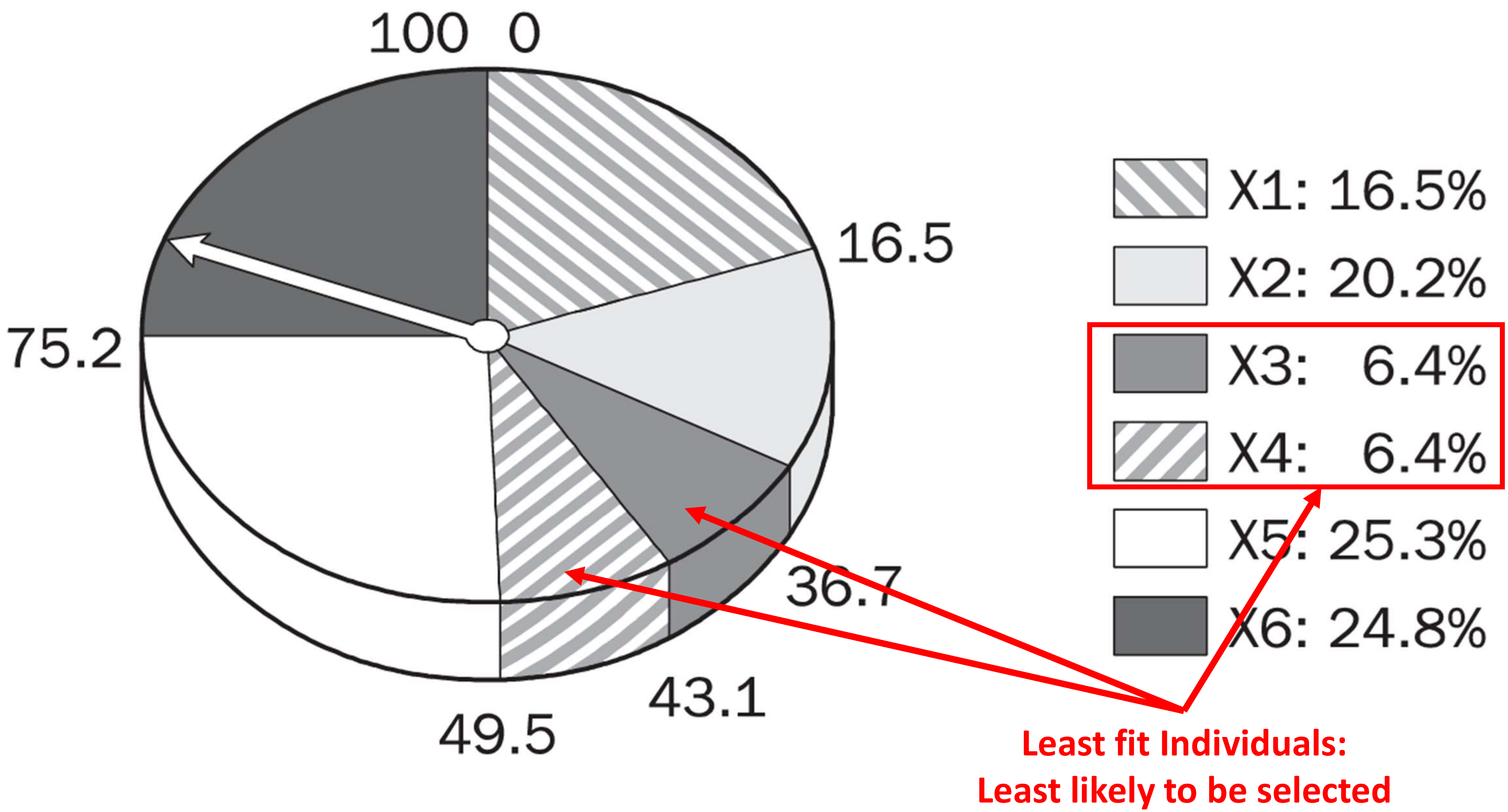
(Individual) Selection Mechanisms

Individual Selection: Roulette Wheel



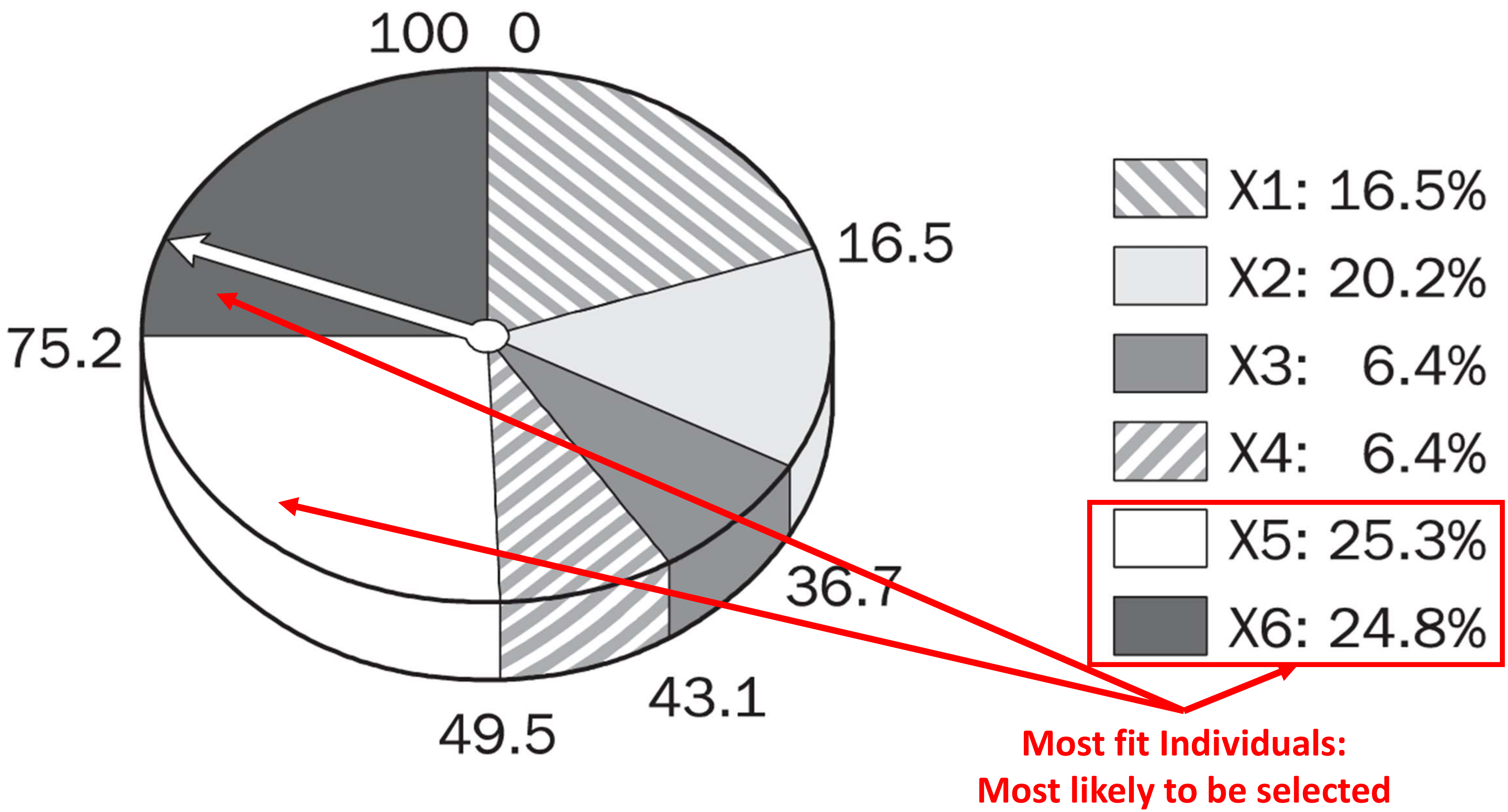
Source: Michael Negnevitsky – "Artificial Intelligence: A Guide to Intelligent Systems"

Individual Selection: Roulette Wheel



Source: Michael Negnevitsky – "Artificial Intelligence: A Guide to Intelligent Systems"

Individual Selection: Roulette Wheel



Source: Michael Negnevitsky – "Artificial Intelligence: A Guide to Intelligent Systems"

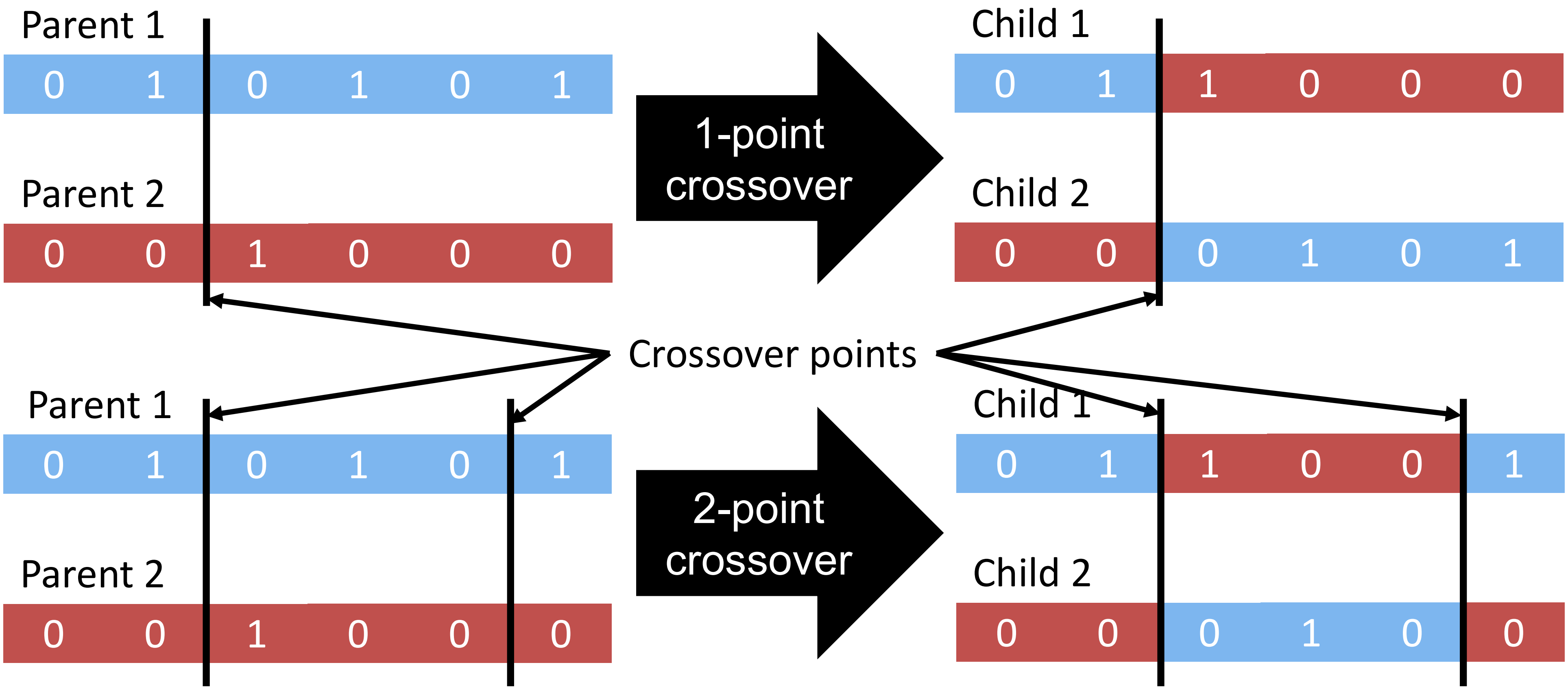
Individual Selection: Tournament

```
function tournament_selection(population, k):
  best = null

  for i = 1 to k
    individual = pick one randomly* from population
    if (best == null) or
            or fitness(individual) > fitness(best)
       best = individual

  return best
```

* could be with or without replacement

Crossover / Reproduction / Mating Mechanisms

Crossover Mechanisms



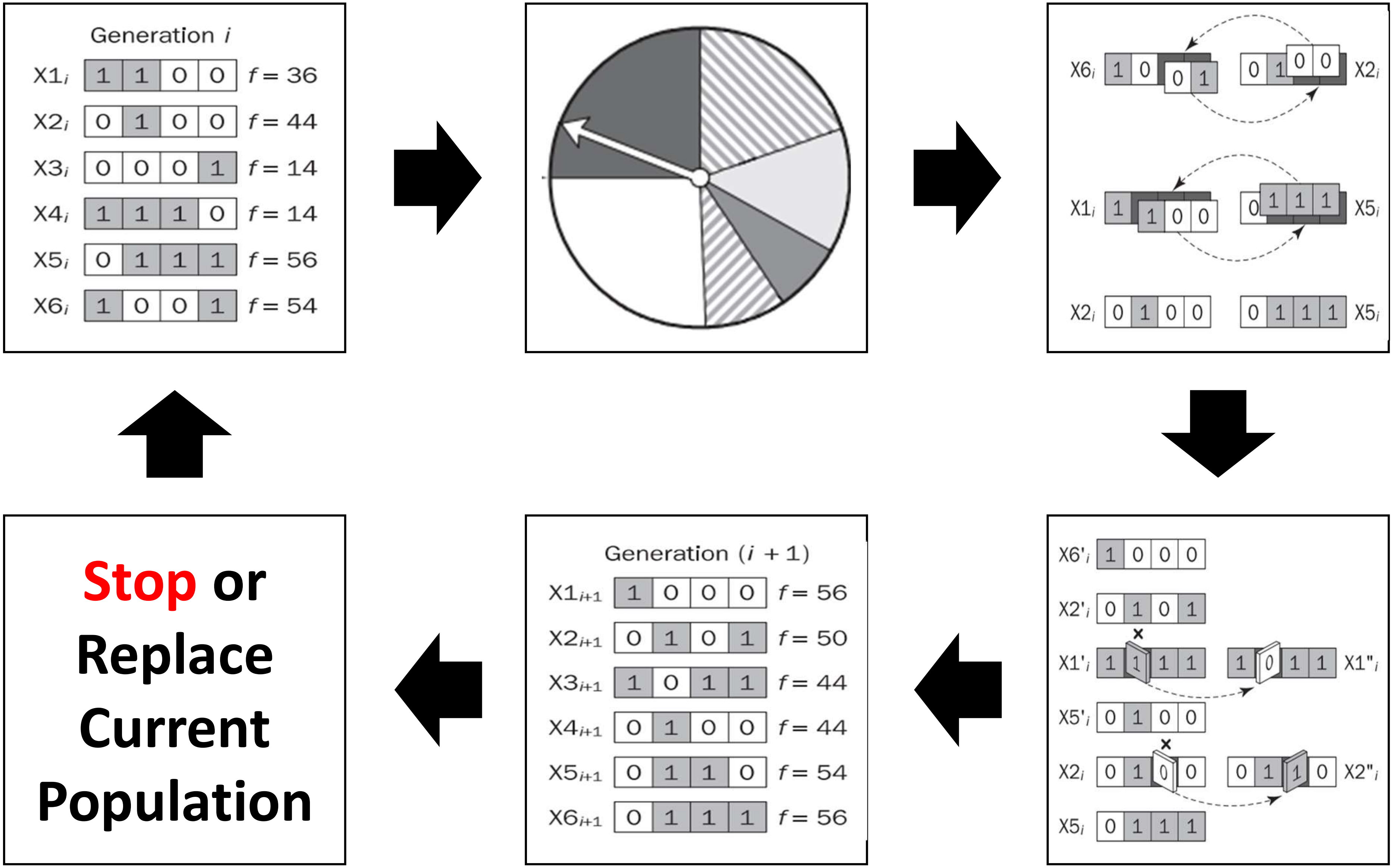
- **Uniform crossover: each bit is chosen from either parent with equal probability**
- **Probability of crossover P_c**
- **Other**

Potential Mutation

Mutation / Probability of Mutation

- Each component (bit, etc.) of every individual / chromosome is modified with
  - mutation probability P_m
- Mutation is the main operator for global search (looking at new areas of the search space)
- P_m is usually small: between 0.001 and 0.01
  - rule of thumb = 1/no. of bits in chromosome
- Individuals not mutated are carried over in population

Genetic Algorithm: Process



Textbook Pseudocode

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

population: **an ordered list of individuals / chromosomes (could be a matrix of 0,1 values)**

Population

Individual	Genotype				Phenotype	Phenotype fitness	Fitness ratio [%]
X1	1	1	0	0	12	36	16.5
X2	0	1	0	0	4	44	20.2
X3	0	0	0	1	1	14	6.4
X4	1	1	1	0	14	14	6.4
X5	0	1	1	1	7	56	25.7
X6	1	0	0	1	9	54	24.8

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, $\boxed{\textit{fitness}}$) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, $\boxed{\textit{fitness}}$)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to $\boxed{\textit{fitness}}$

fitness: **fitness ("objective") function**

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c + 1$, n))

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

will return last best ("most fit") individual / chromosome. It may or may not be the global maximum

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

weights: **list (or vector) of corresponding fitness values for each individual (matches *population*)**

Weights

Individual	Genotype				Phenotype	Phenotype fitness	Fitness ratio [%]
X1	1	1	0	0	12	36	16.5
X2	0	1	0	0	4	44	20.2
X3	0	0	0	1	1	14	6.4
X4	1	1	1	0	14	14	6.4
X5	0	1	1	1	7	56	25.7
X6	1	0	0	1	9	54	24.8

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
     weights ← WEIGHTED-BY(population, fitness)
     population2 ← empty list
     for i = 1 to SIZE(population) do
        parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
        child ← REPRODUCE(parent1, parent2)
        if (small random probability) then child ← MUTATE(child)
        add child to population2
     population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

"evolution" loop

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← $\boxed{\text{WEIGHTED-BY}(\textit{population}, \textit{fitness})}$
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c + 1$, n))

weights: **go through every individual / chromosome in** *population* **and evaluate its fitness according to** *fitness* **function**

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

population2: **temporary list of individuals that will REPLACE current** *population* **in the next round / iteration (initialized as empty)**

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

population2: **temporary list of individuals that will REPLACE current** *population* **in the next round / iteration (initialized as empty)**

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

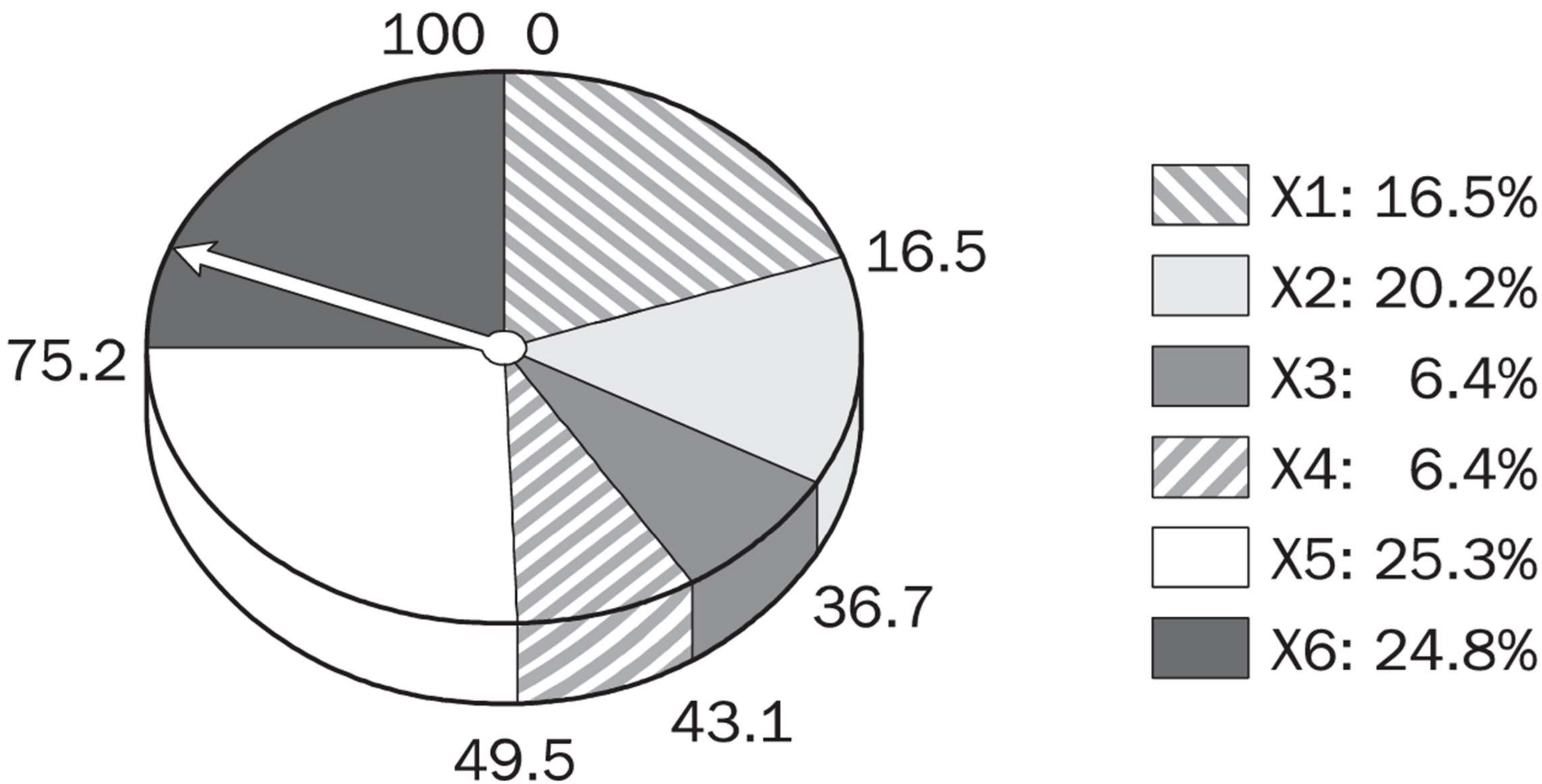
"breed new population" loop

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Select two parents (according to their fitness) from current *population* for reproduction

Weighted-Random-Choices: Could Be



Source: Michael Negnevitsky – "Artificial Intelligence: A Guide to Intelligent Systems"

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      $\boxed{\textit{parent1}, \textit{parent2} \leftarrow \text{WEIGHTED-RANDOM-CHOICES}(\textit{population}, \textit{weights}, 2)}$
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

Could be through Roulette Wheel selection

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

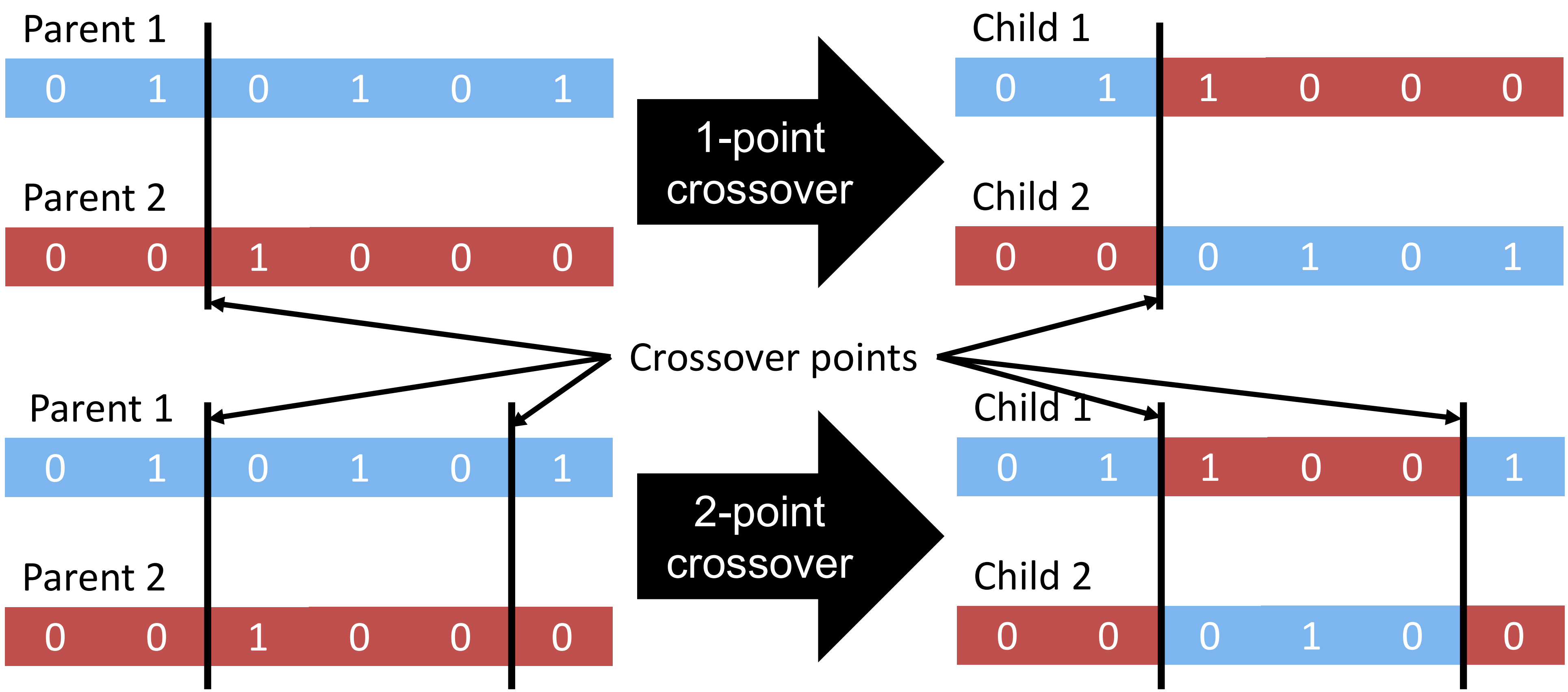
Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

Reproduction / crossover step: a new individual (*child*) is created [some algorithms produce TWO] based on *parent1*, *parent2*

Crossover Mechanisms



- **Uniform crossover: each bit is chosen from either parent with equal probability**
- **Probability of crossover P_c**
- **Other**

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Random mutation step (typically 1 bit flipped) on the newly produced individual *child* [may or may not happen]

Mutation / Probability of Mutation

- Each component (bit, etc.) of every individual / chromosome is modified with
  - mutation probability P_m
- Mutation is the main operator for global search (looking at new areas of the search space)
- P_m is usually small: between 0.001 and 0.01
  - rule of thumb = 1/no. of bits in chromosome
- Individuals not mutated are carried over in population

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for $i = 1$ **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

Add *child* to the "next iteration/round" population *population2*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  $n \leftarrow$ LENGTH(*parent1*)
  $c \leftarrow$ random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for $i = 1$ **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

Replace current population *population* with "next iteration/round" population *population2*

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  $n \leftarrow$ LENGTH(*parent1*)
  $c \leftarrow$ random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
      weights ← WEIGHTED-BY(population, fitness)
      population2 ← empty list
      for i = 1 to SIZE(population) do
          parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
          child ← REPRODUCE(parent1, parent2)
          if (small random probability) then child ← MUTATE(child)
          add child to population2
      population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Terminate the "evolution" process. Can be iteration-, fitness-, and time-based

Genetic Algorithm: Pseudocode

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
     weights ← WEIGHTED-BY(population, fitness)
     population2 ← empty list
     for i = 1 to SIZE(population) do
        parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
        child ← REPRODUCE(parent1, parent2)
        if (small random probability) then child ← MUTATE(child)
        add child to population2
     population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness
```

Reproduction / crossover step function: splice two individuals / chromosomes

```
function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

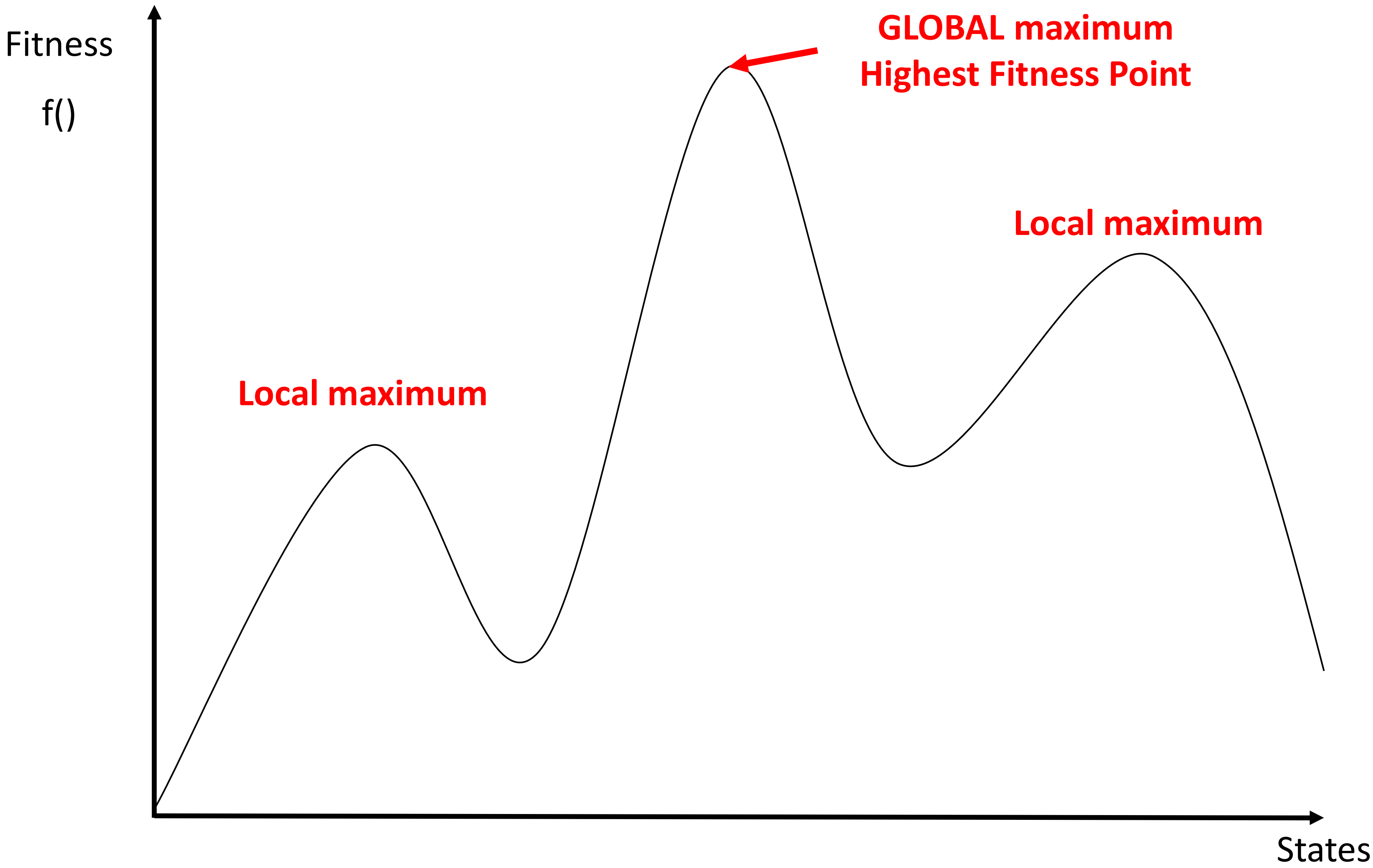
Genetic Algorithm: Pseudocode

function GENETIC-ALGORITHM(*population*, *fitness*) **returns** an individual
  repeat
    weights ← WEIGHTED-BY(*population*, *fitness*)
    population2 ← empty list
    for i = 1 **to** SIZE(*population*) **do**
      parent1, *parent2* ← WEIGHTED-RANDOM-CHOICES(*population*, *weights*, 2)
      child ← REPRODUCE(*parent1*, *parent2*)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *population2*
    population ← *population2*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to *fitness*

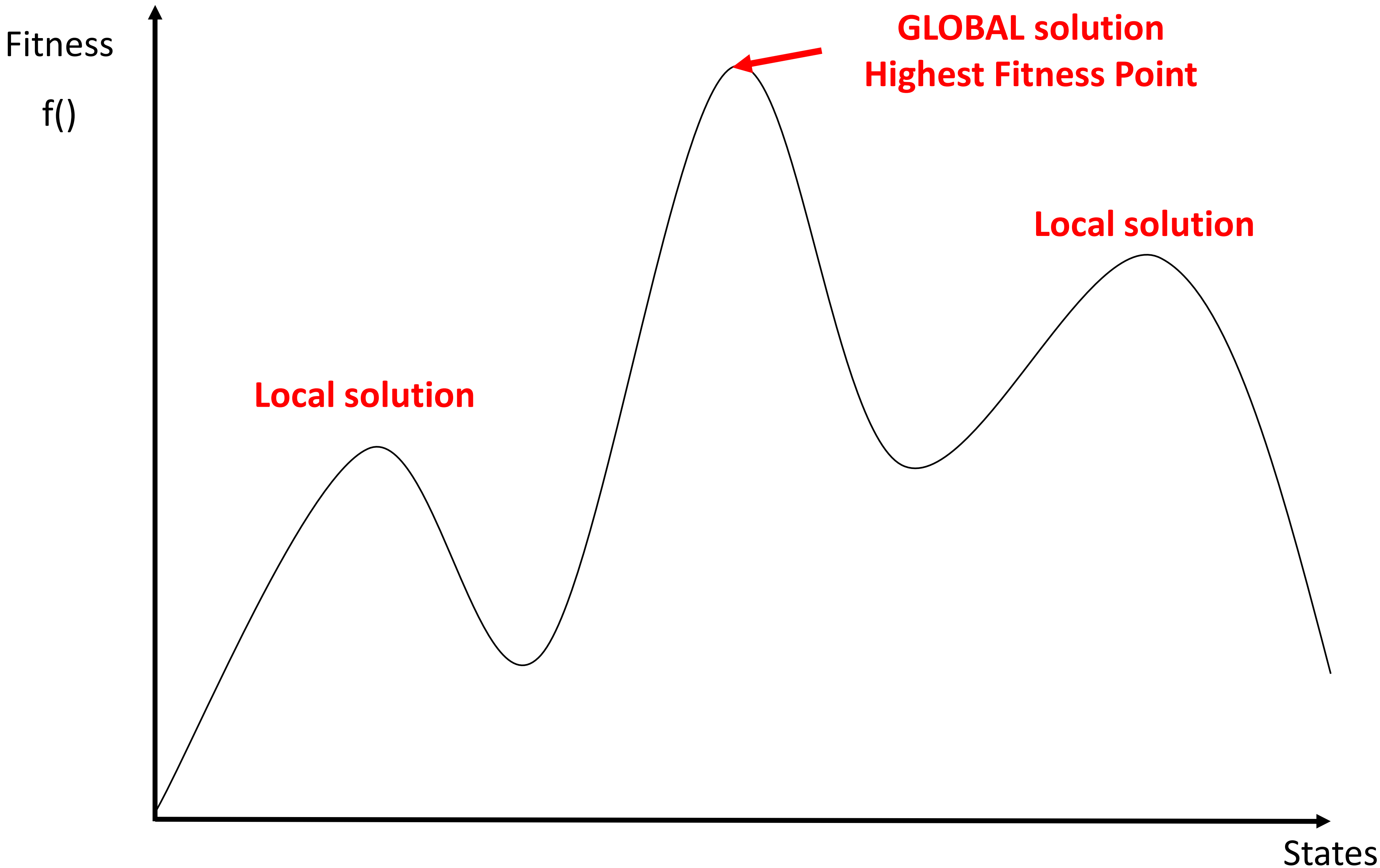
Reproduction / crossover step function: splice two individuals / chromosomes

function REPRODUCE(*parent1*, *parent2*) **returns** an individual
  n ← LENGTH(*parent1*)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(*parent1*, 1, c), SUBSTRING(*parent2*, $c+1$, n))

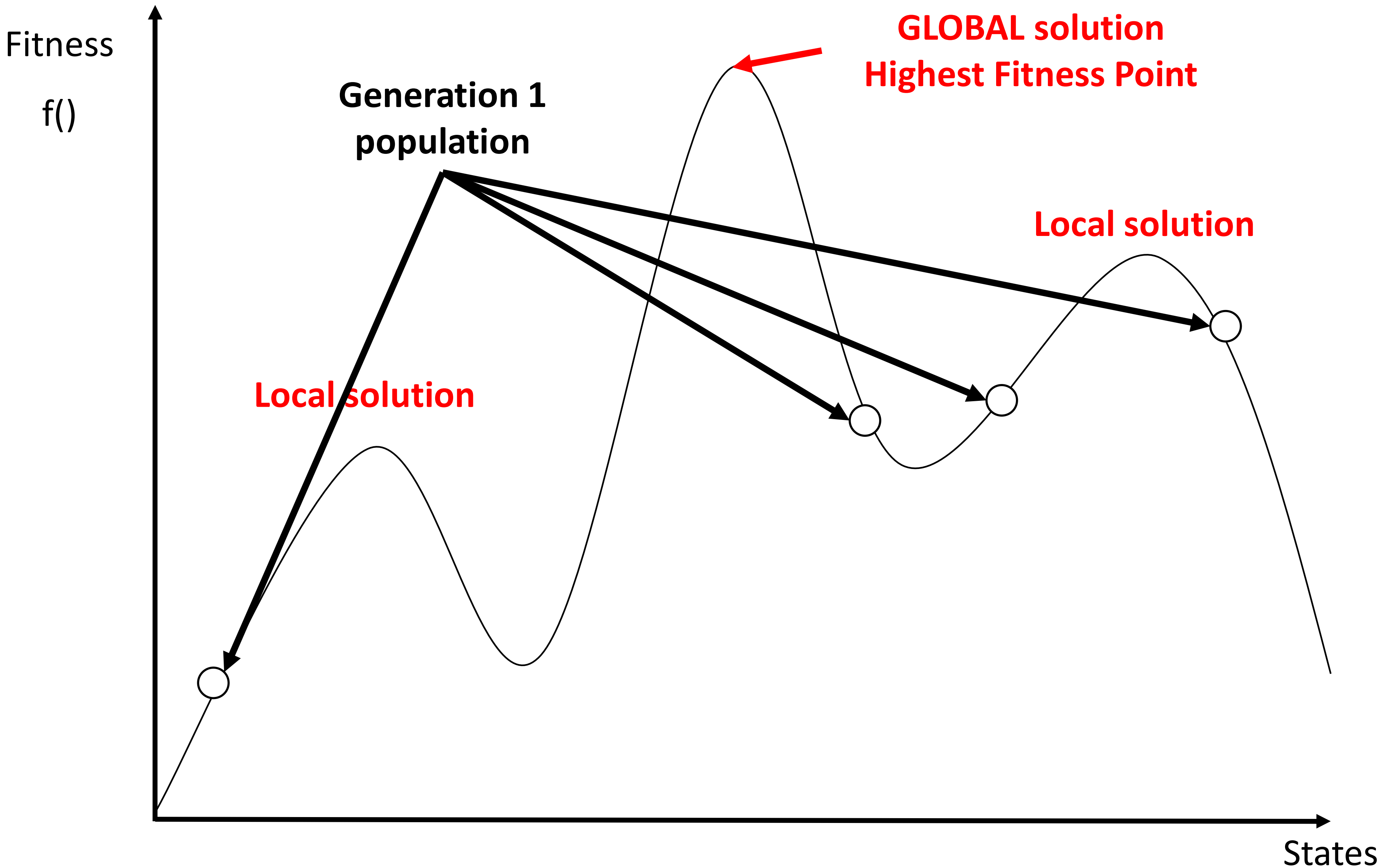
Genetic Algorithm: Progress



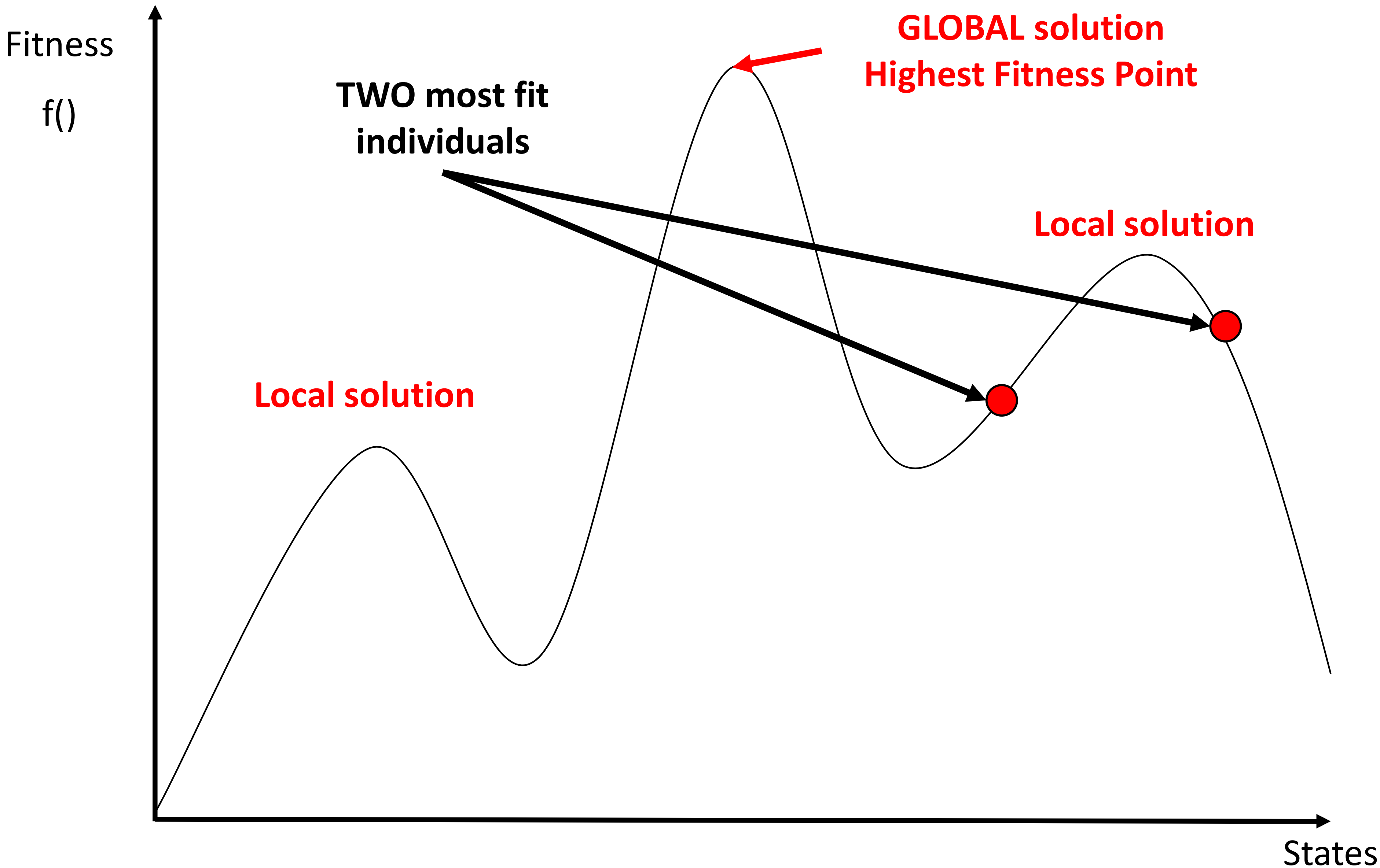
Genetic Algorithm: Progress



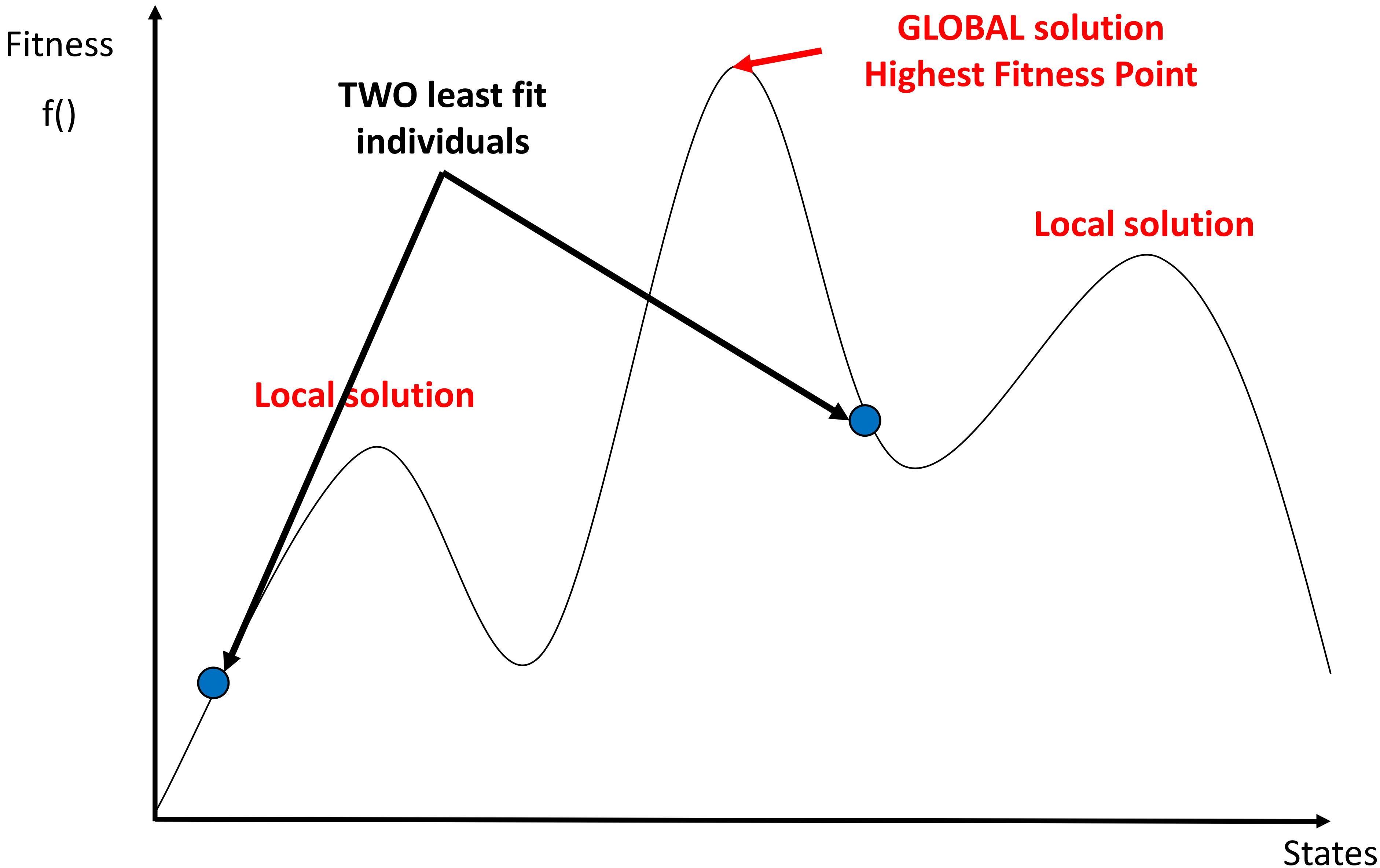
Genetic Algorithm: Progress



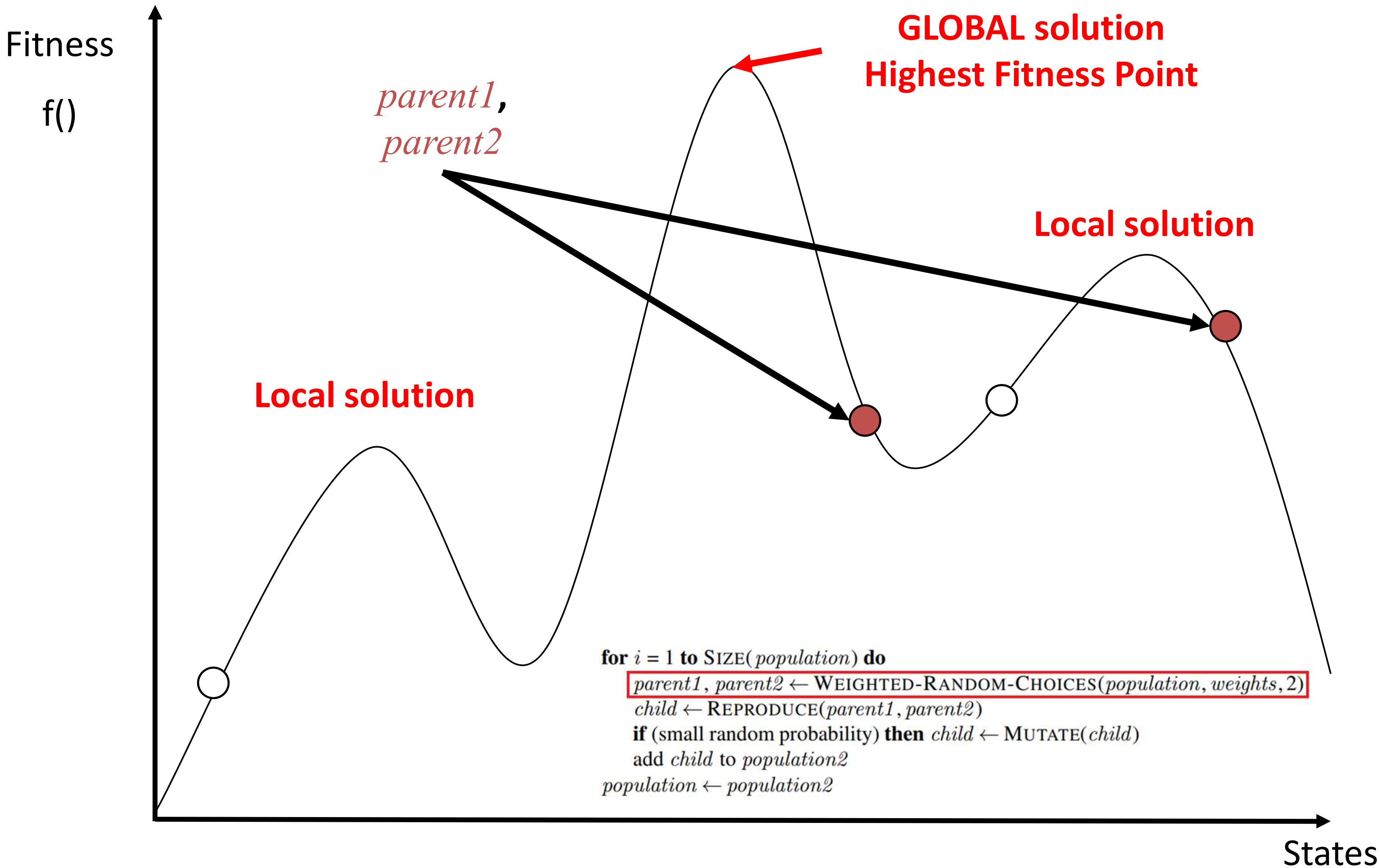
Genetic Algorithm: Progress



Genetic Algorithm: Progress

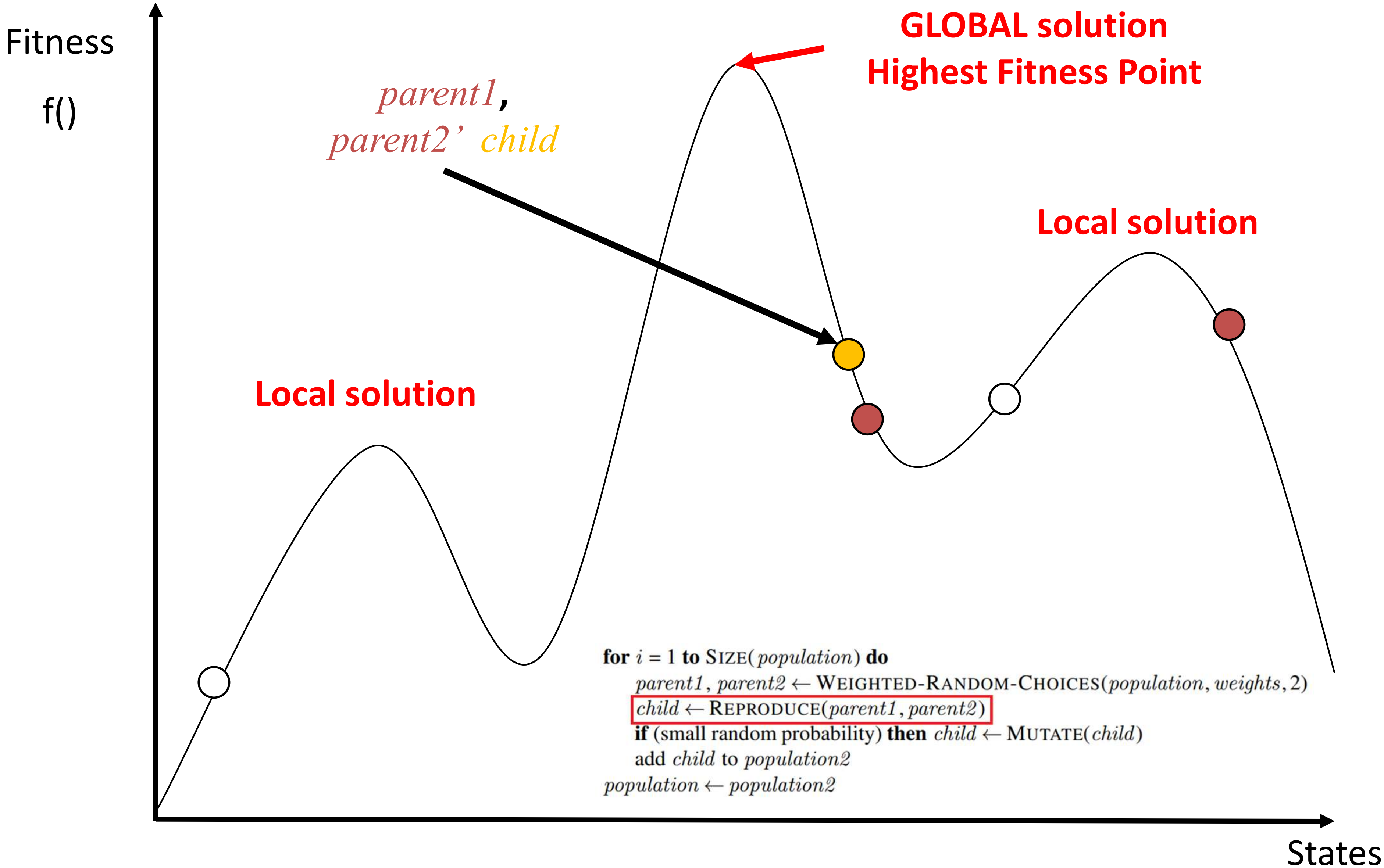


Genetic Algorithm: Progress



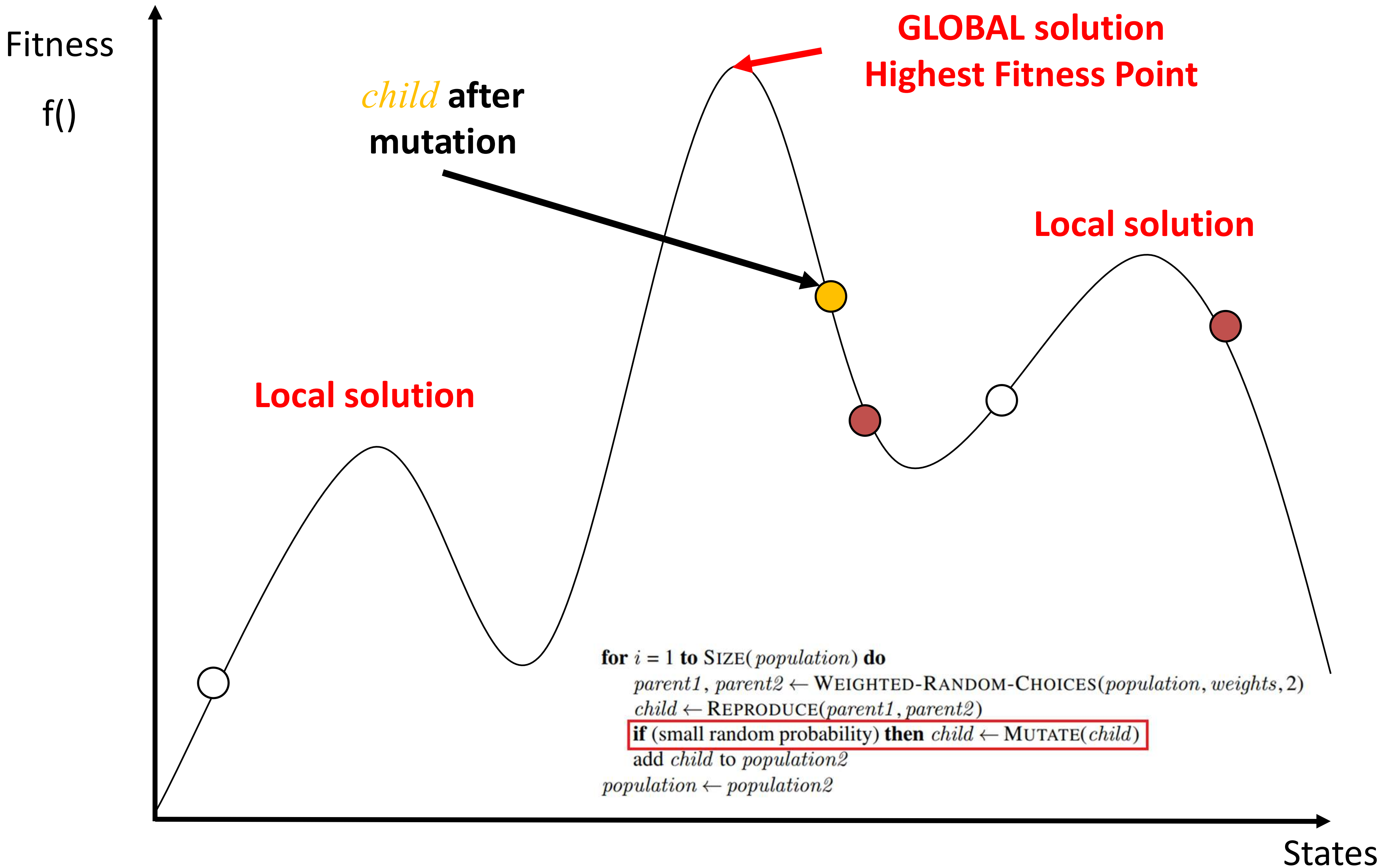

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress



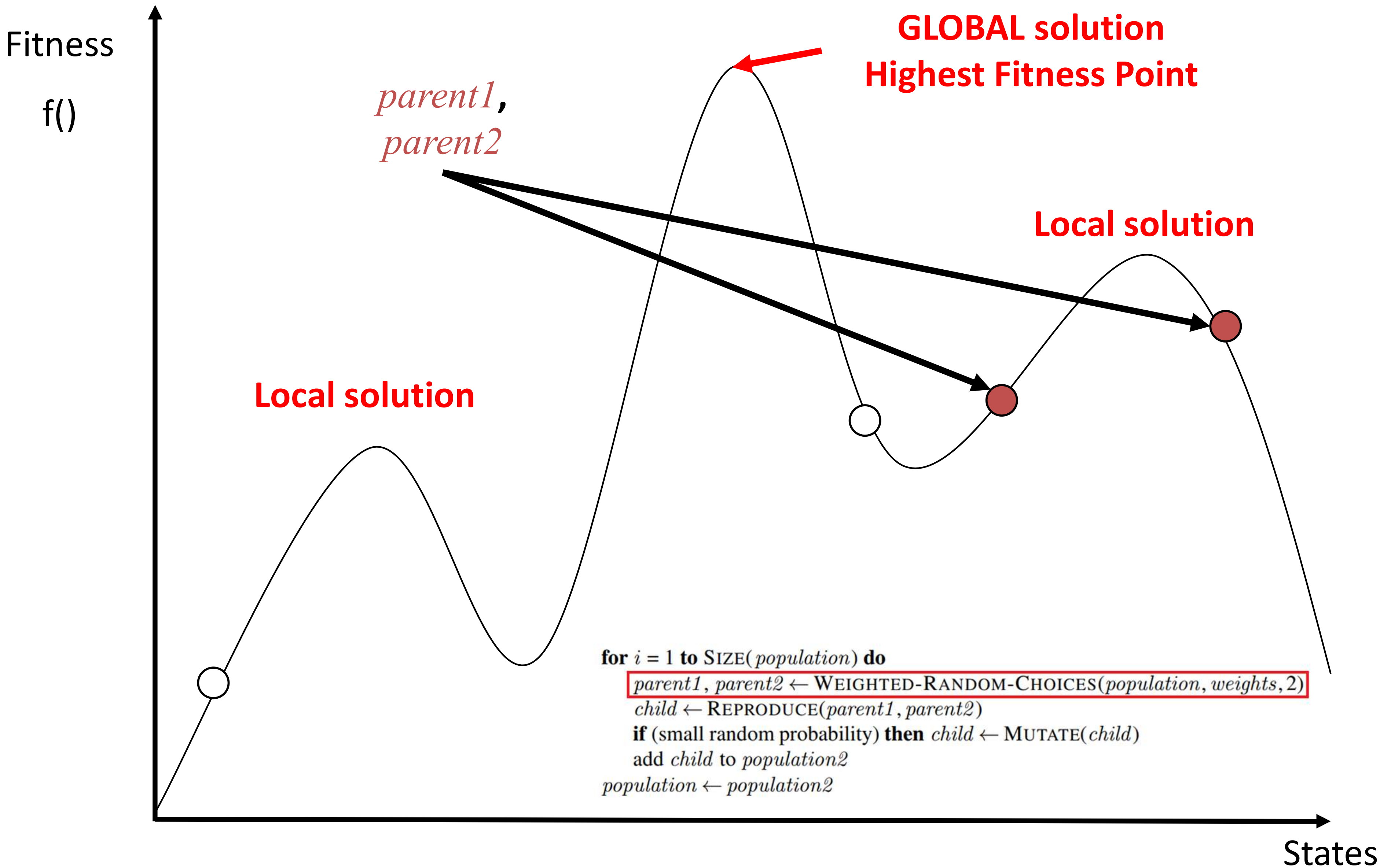

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress



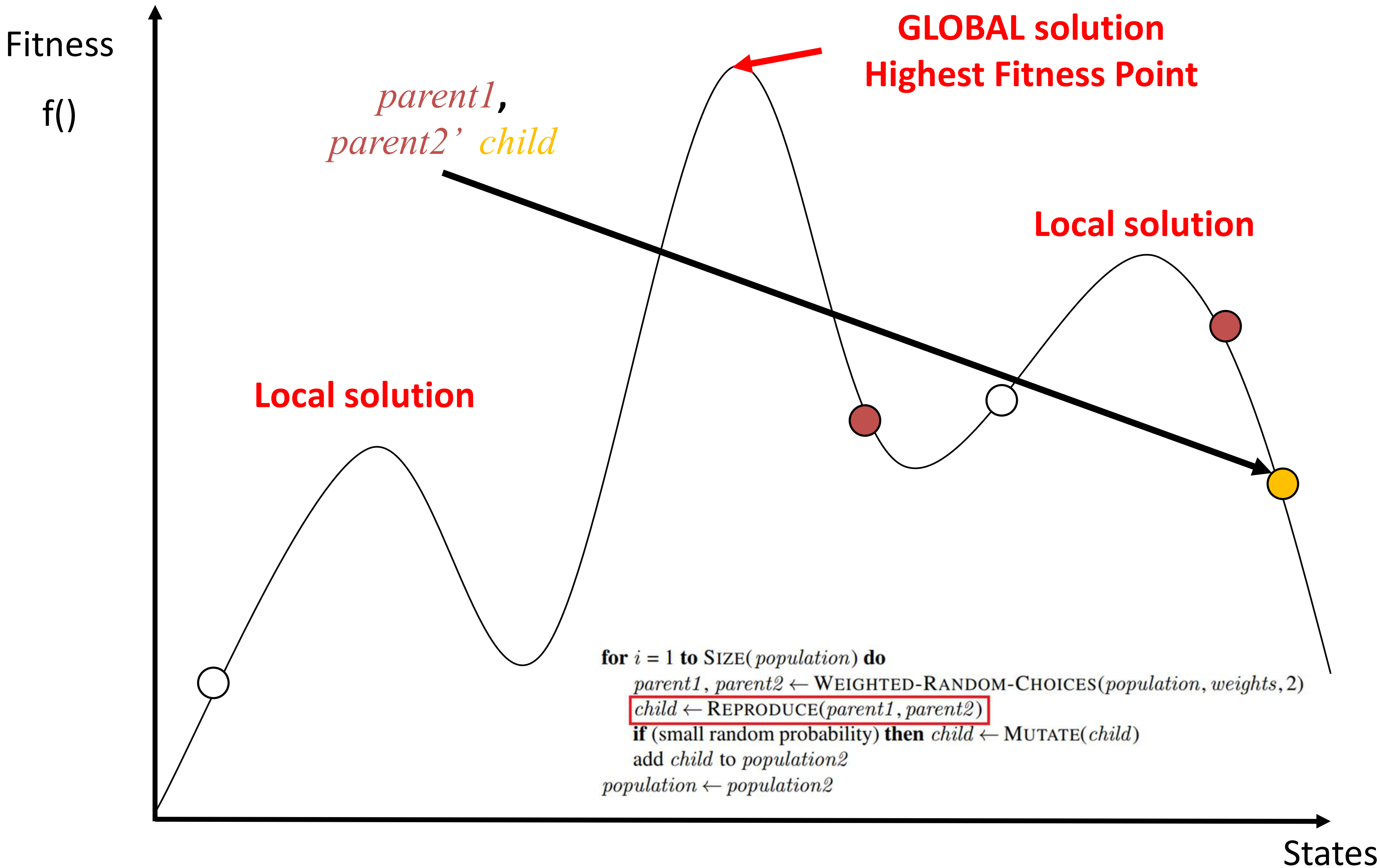

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress



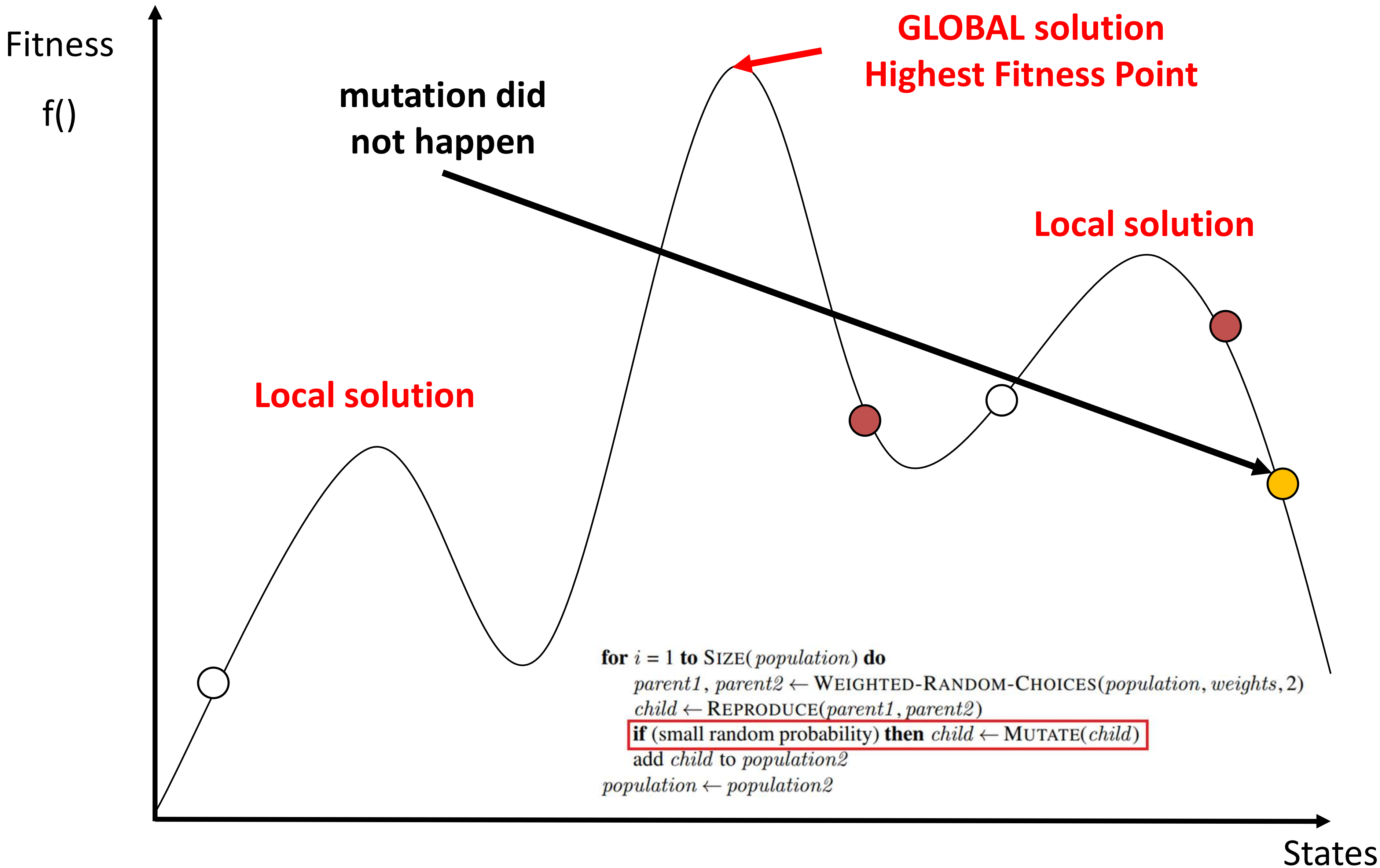
for i = 1 **to** SIZE(*population*) **do**
  $\boxed{parent1, parent2 \leftarrow \text{WEIGHTED-RANDOM-CHOICES}(population, weights, 2)}$
  child ← REPRODUCE(*parent1*, *parent2*)
  if (small random probability) **then** *child* ← MUTATE(*child*)
  add *child* to *population2*
population ← *population2*

Genetic Algorithm: Progress



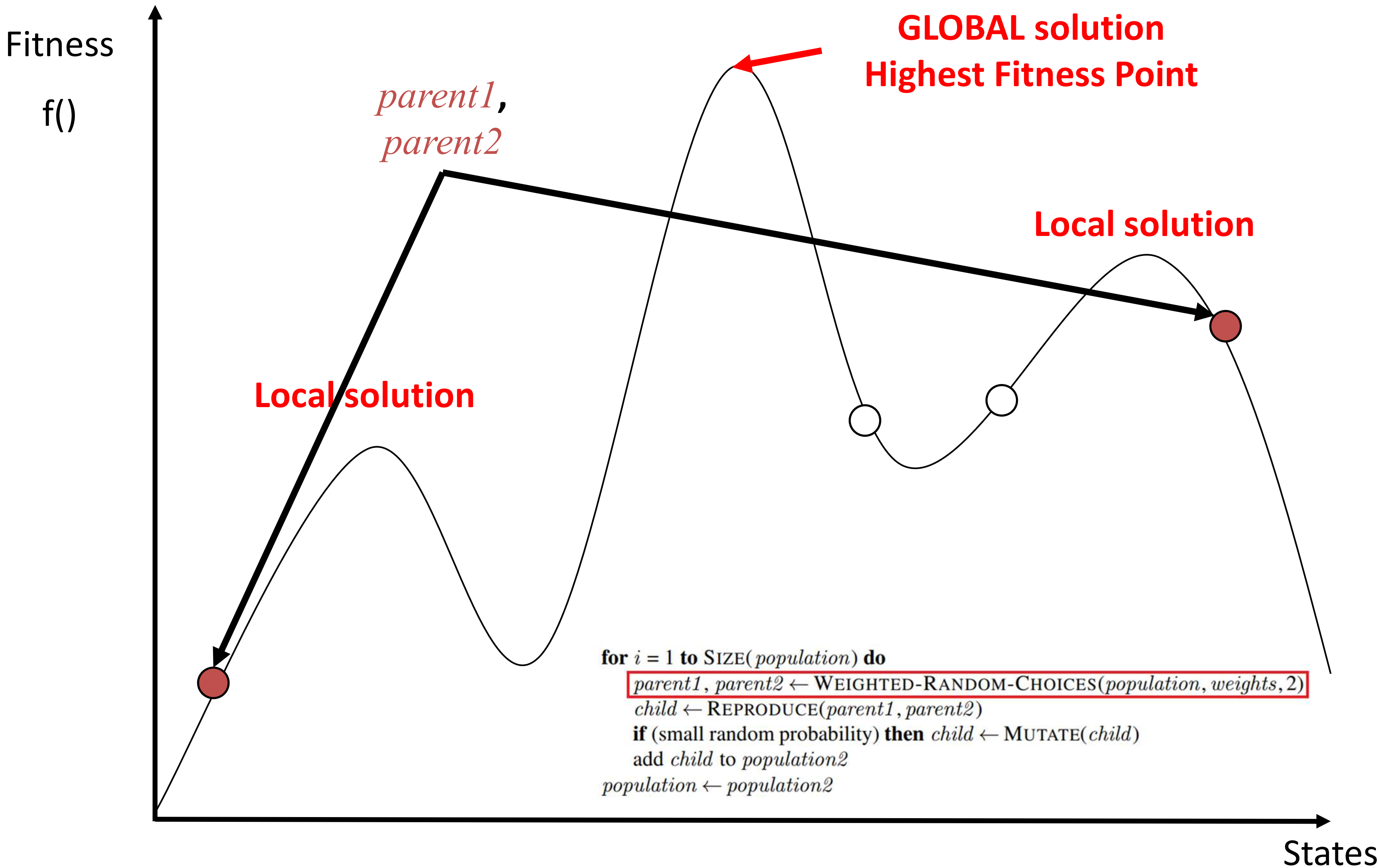

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress

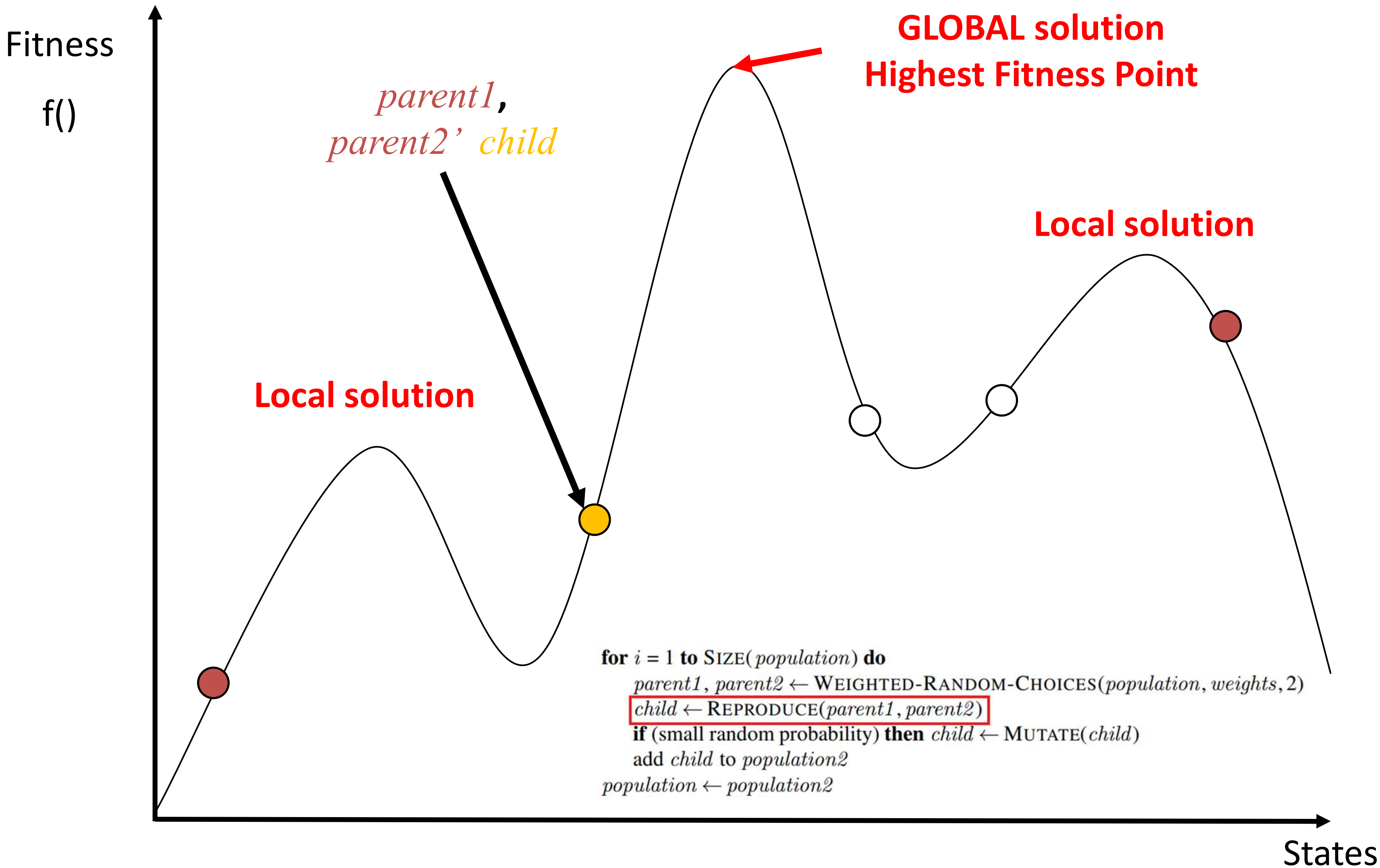



```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

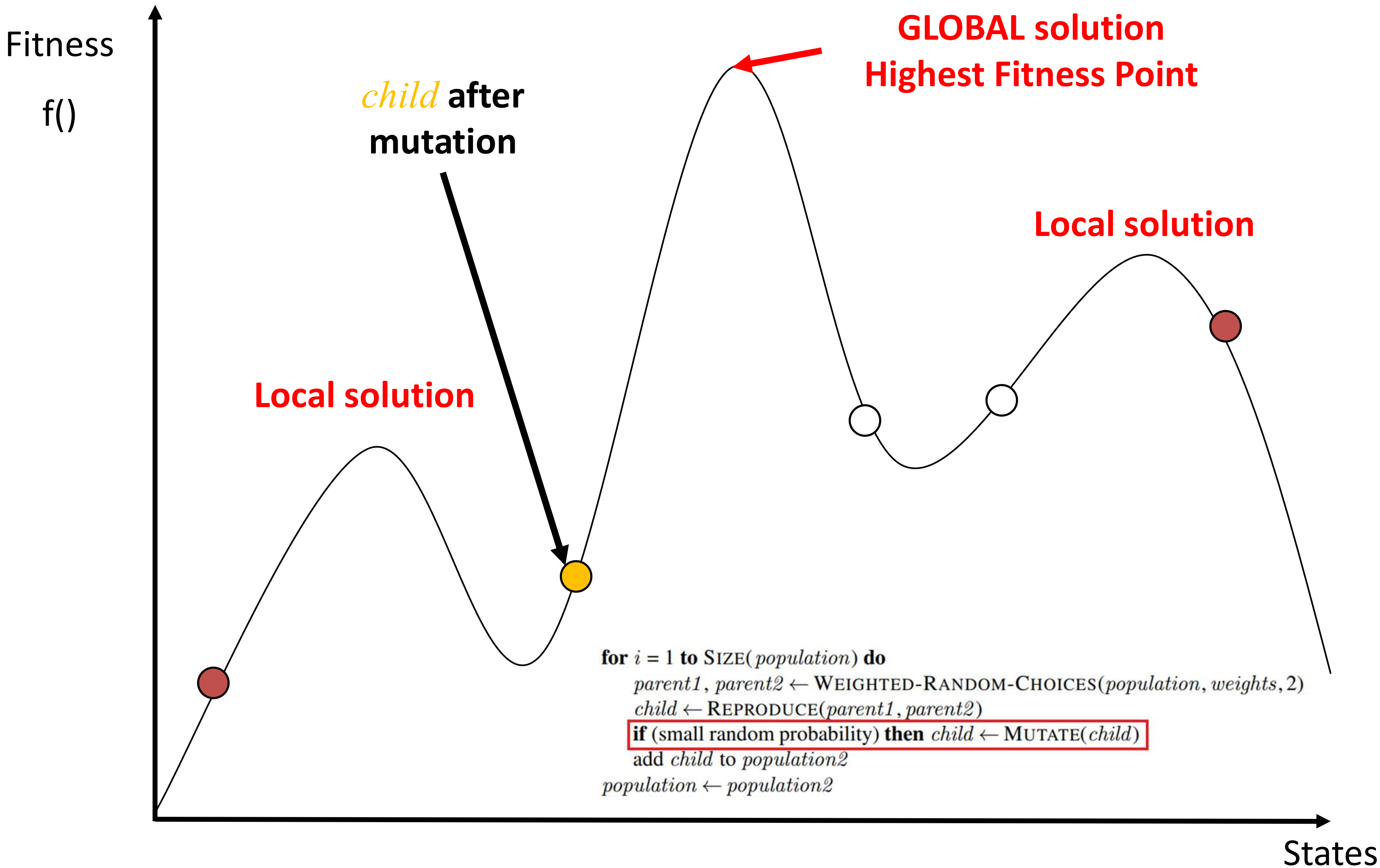
Genetic Algorithm: Progress



Genetic Algorithm: Progress

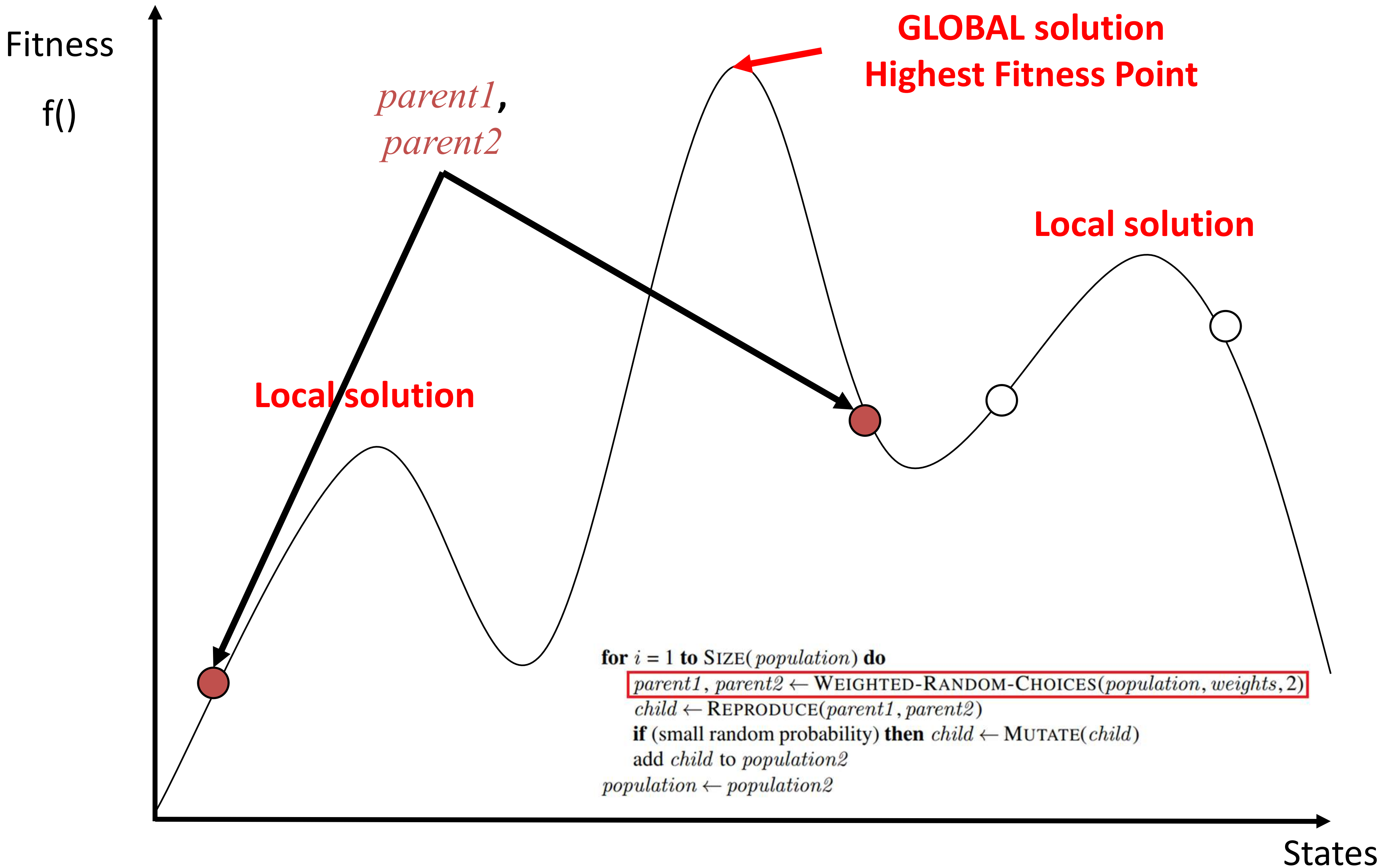


Genetic Algorithm: Progress



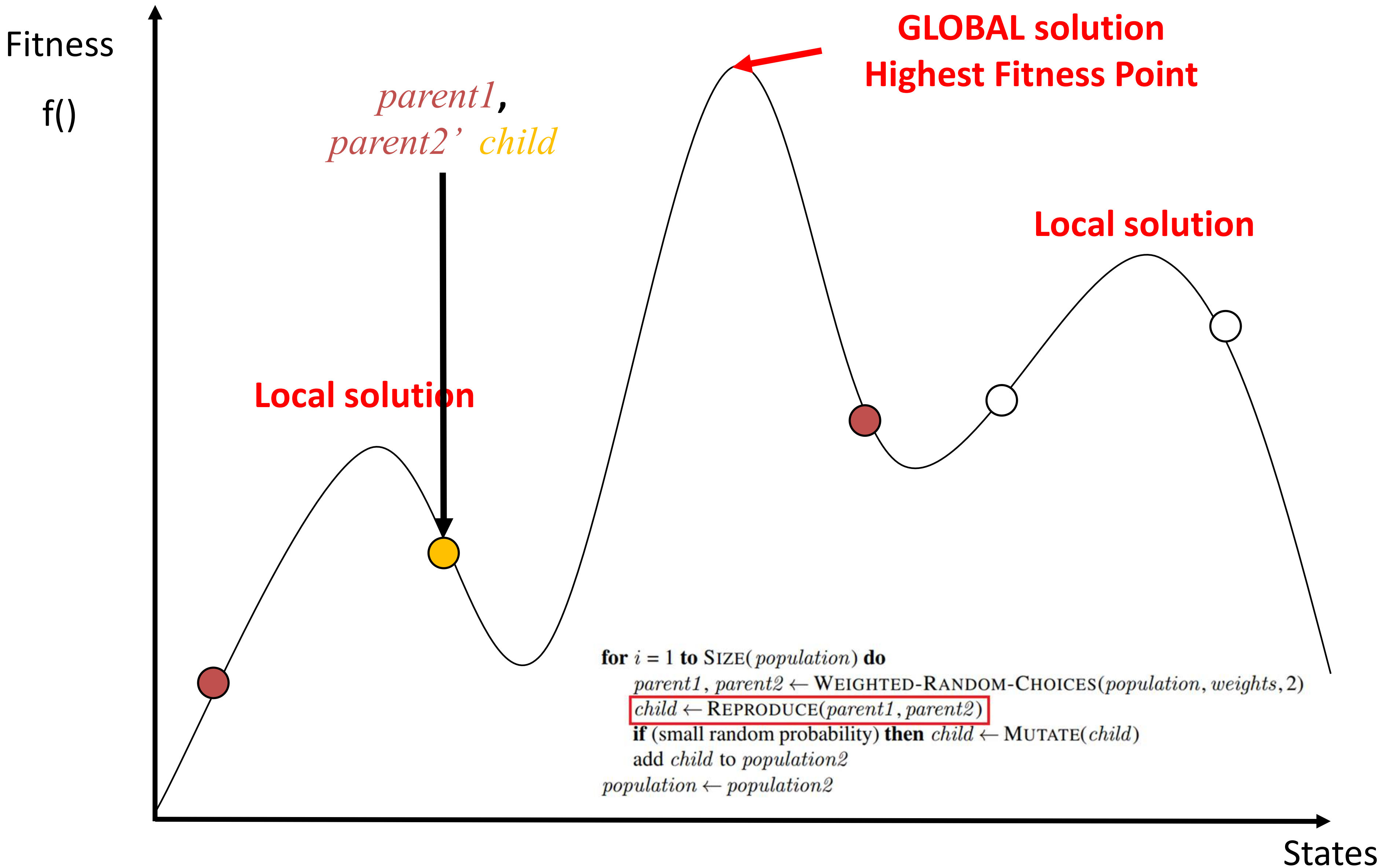

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress



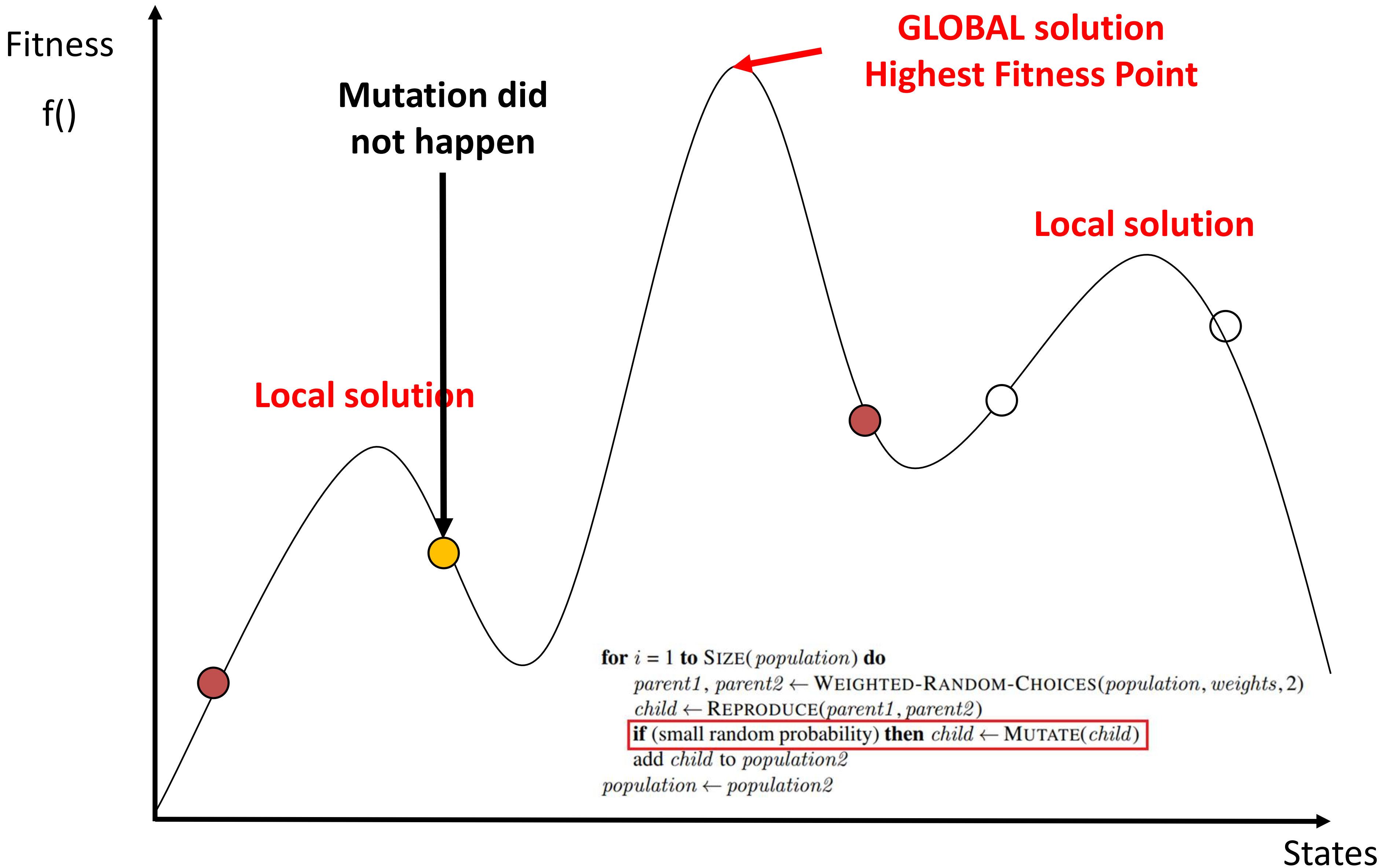

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress



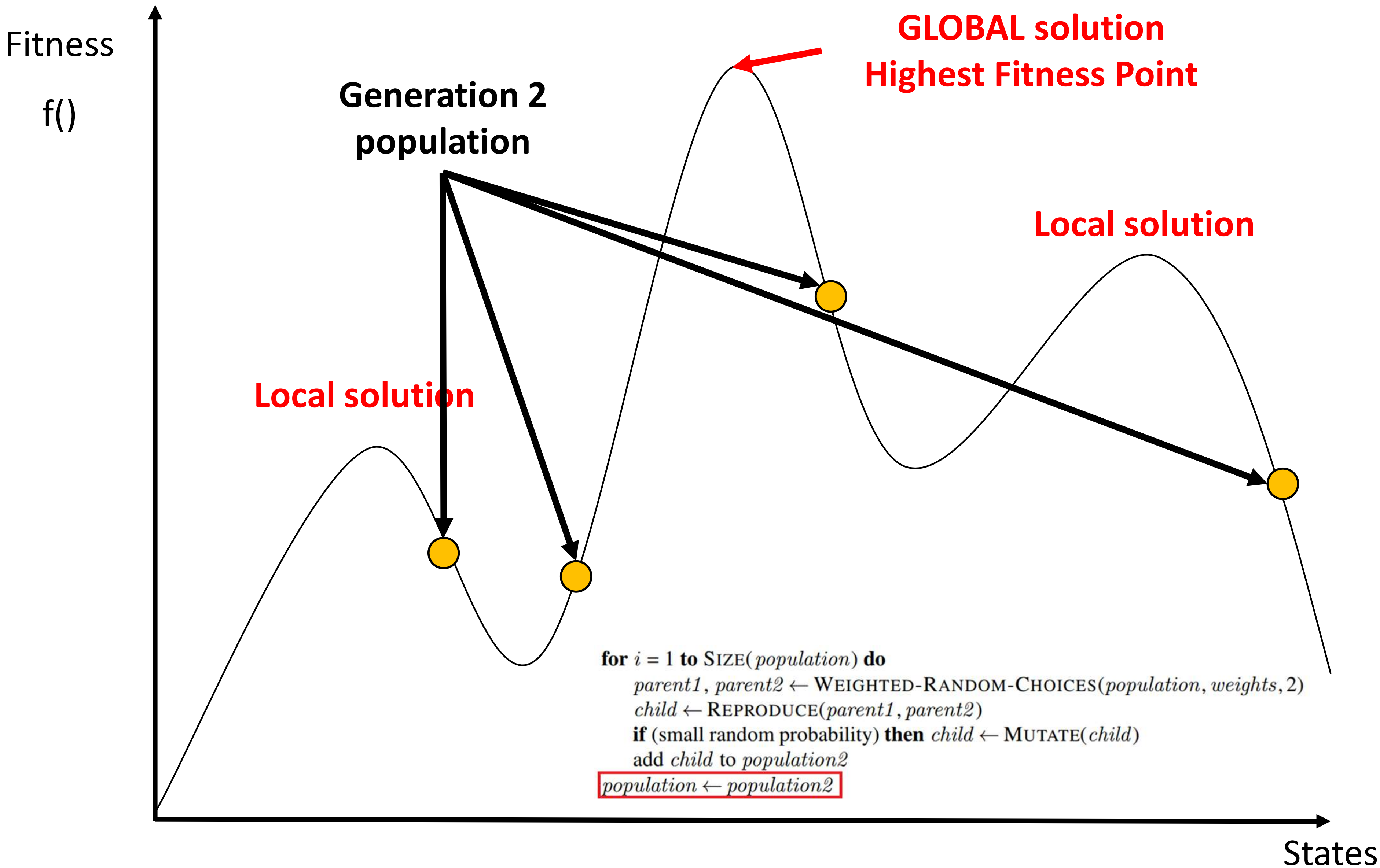

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress



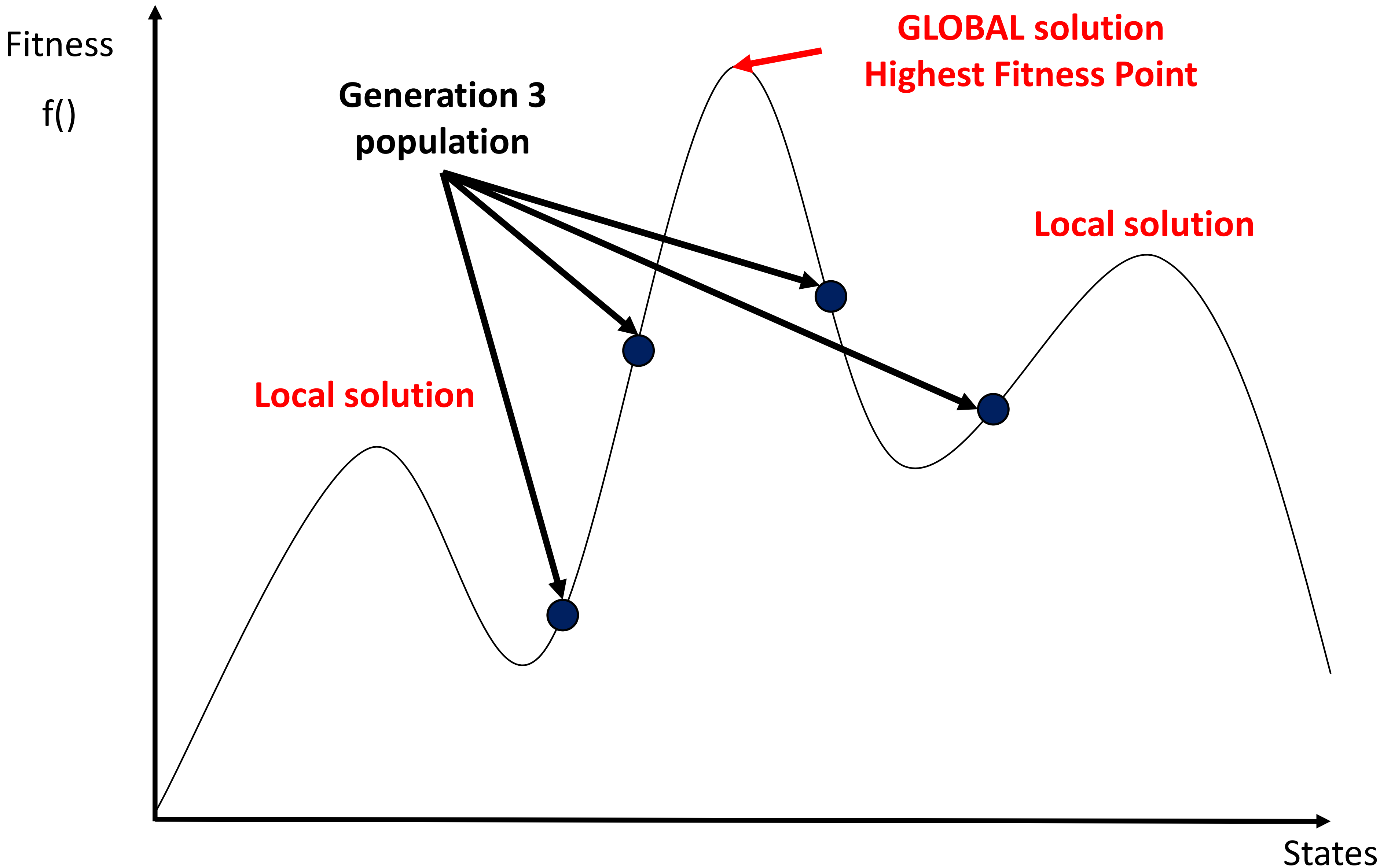

```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

Genetic Algorithm: Progress

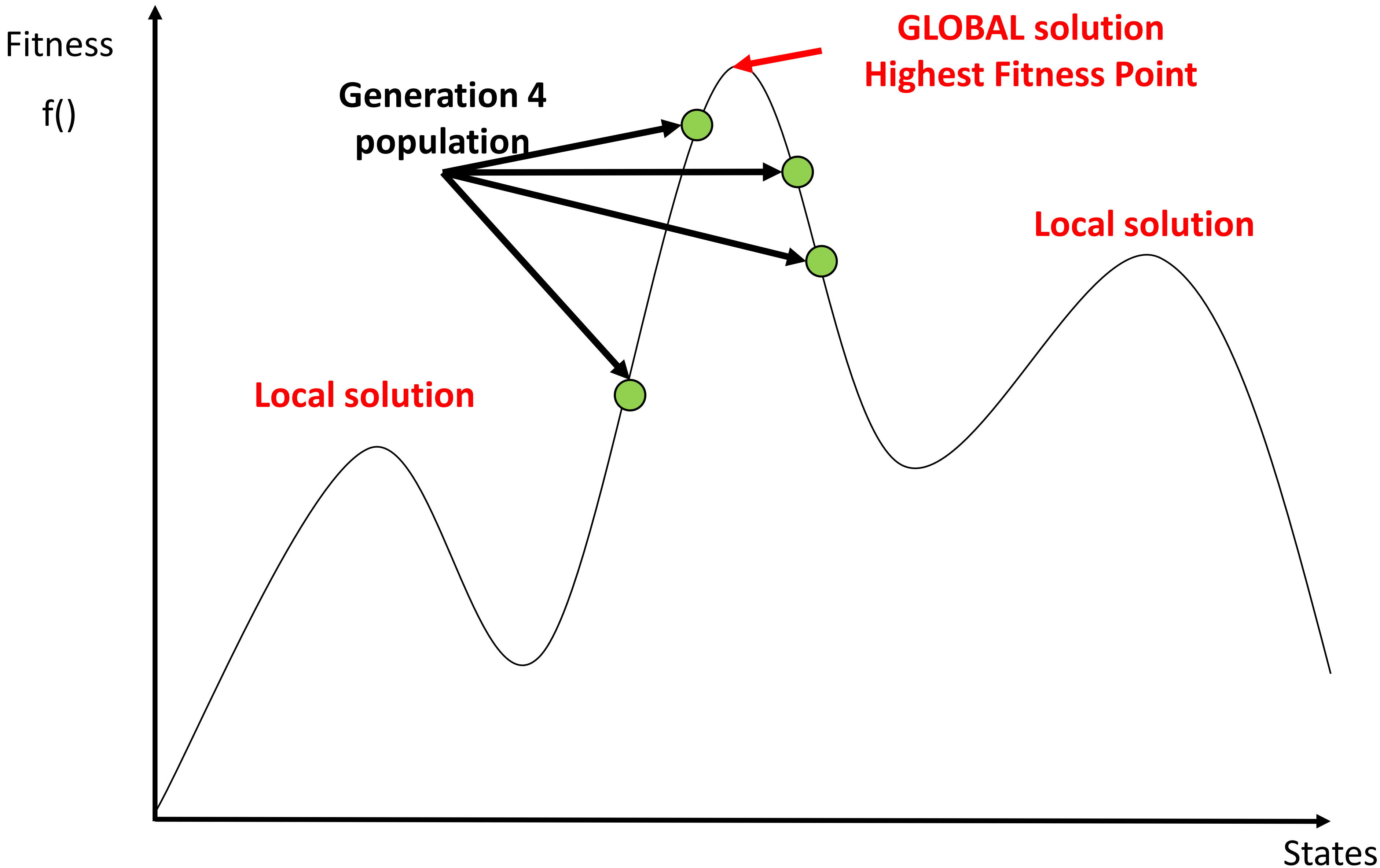



```
for i = 1 to SIZE(population) do
    parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
    child ← REPRODUCE(parent1, parent2)
    if (small random probability) then child ← MUTATE(child)
    add child to population2
population ← population2
```

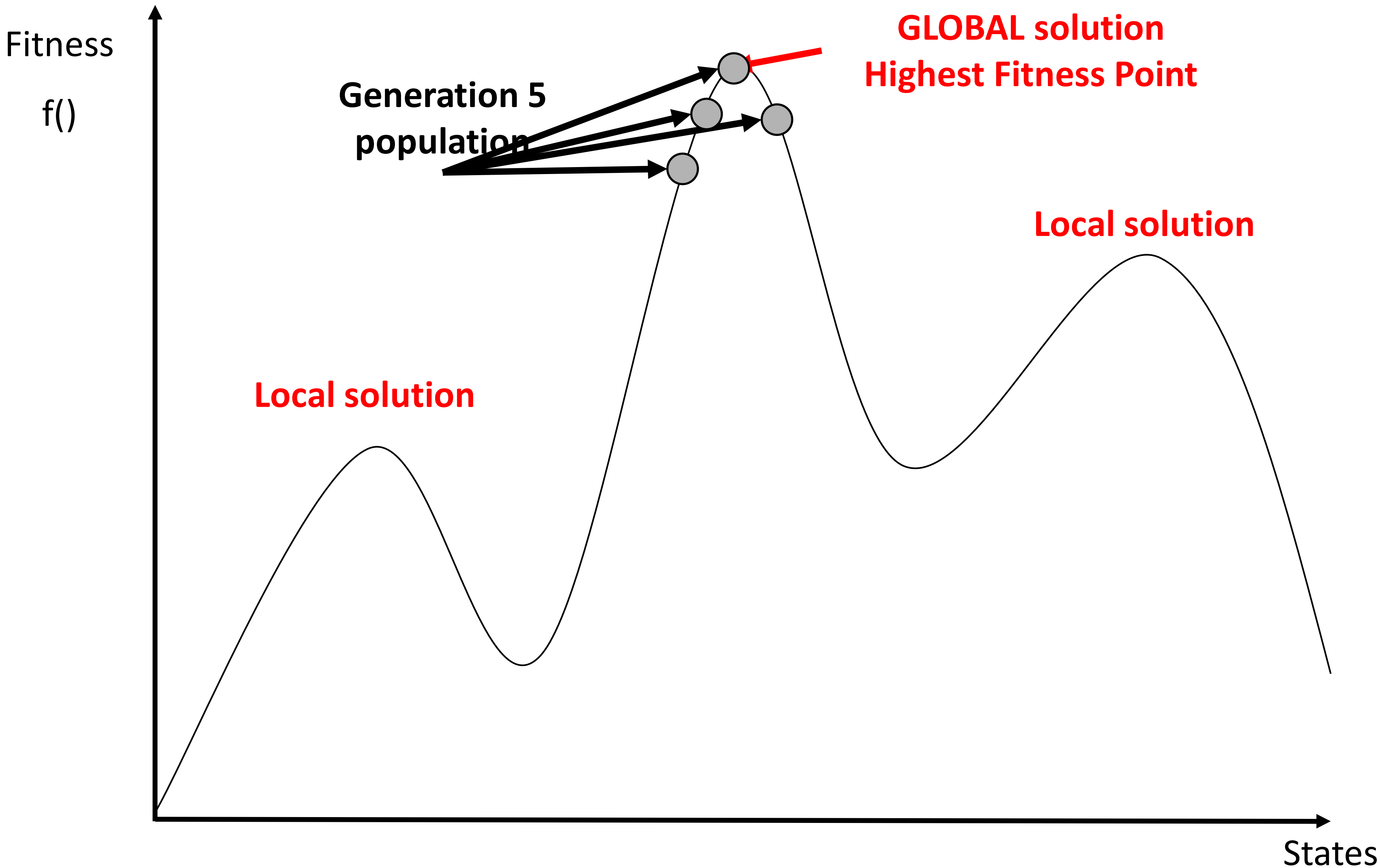
Genetic Algorithm: Progress



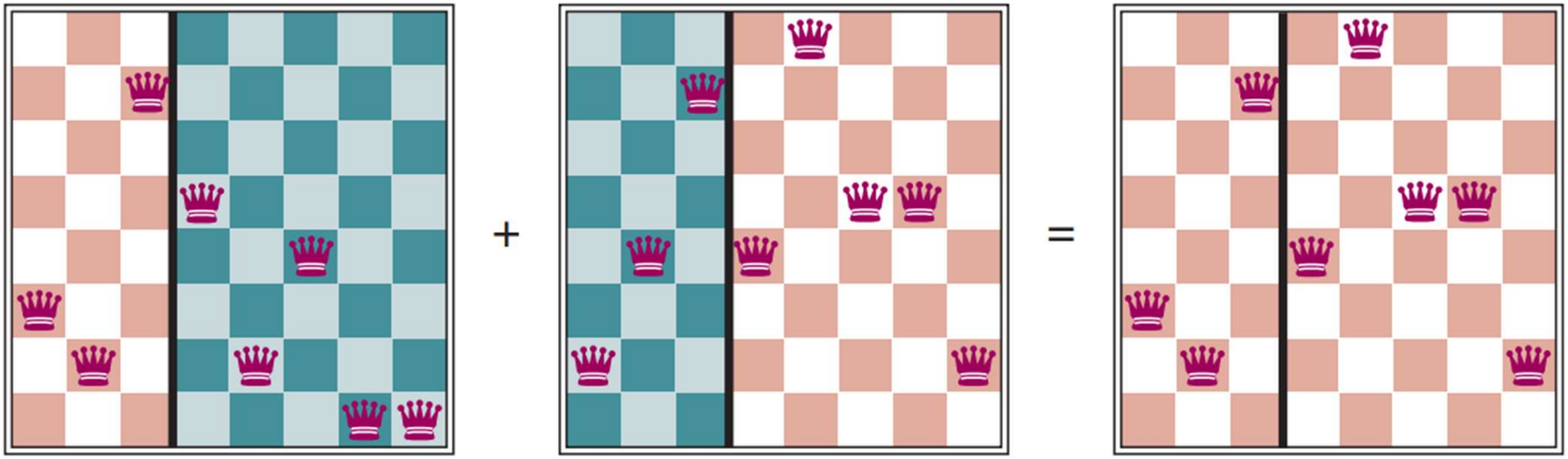
Genetic Algorithm: Progress

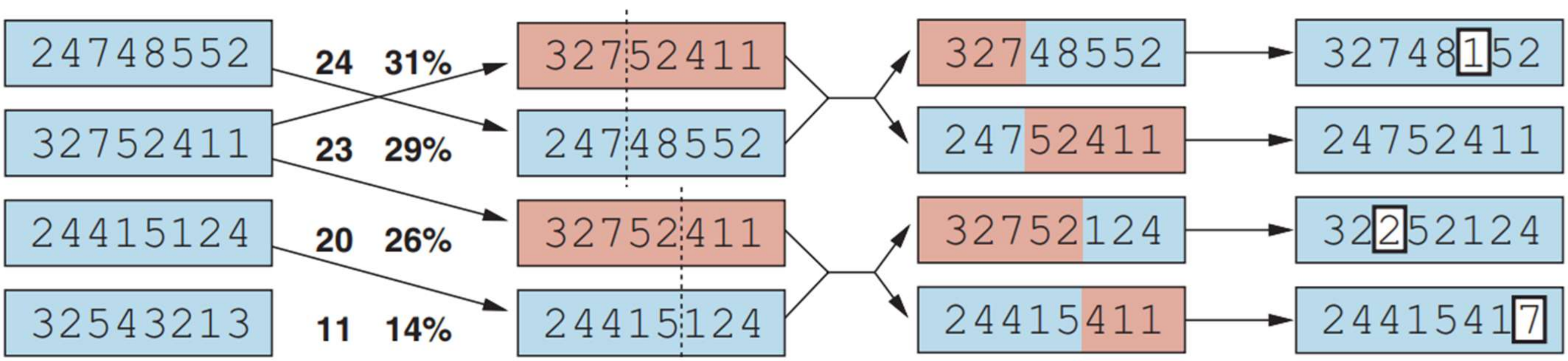


Genetic Algorithm: Progress



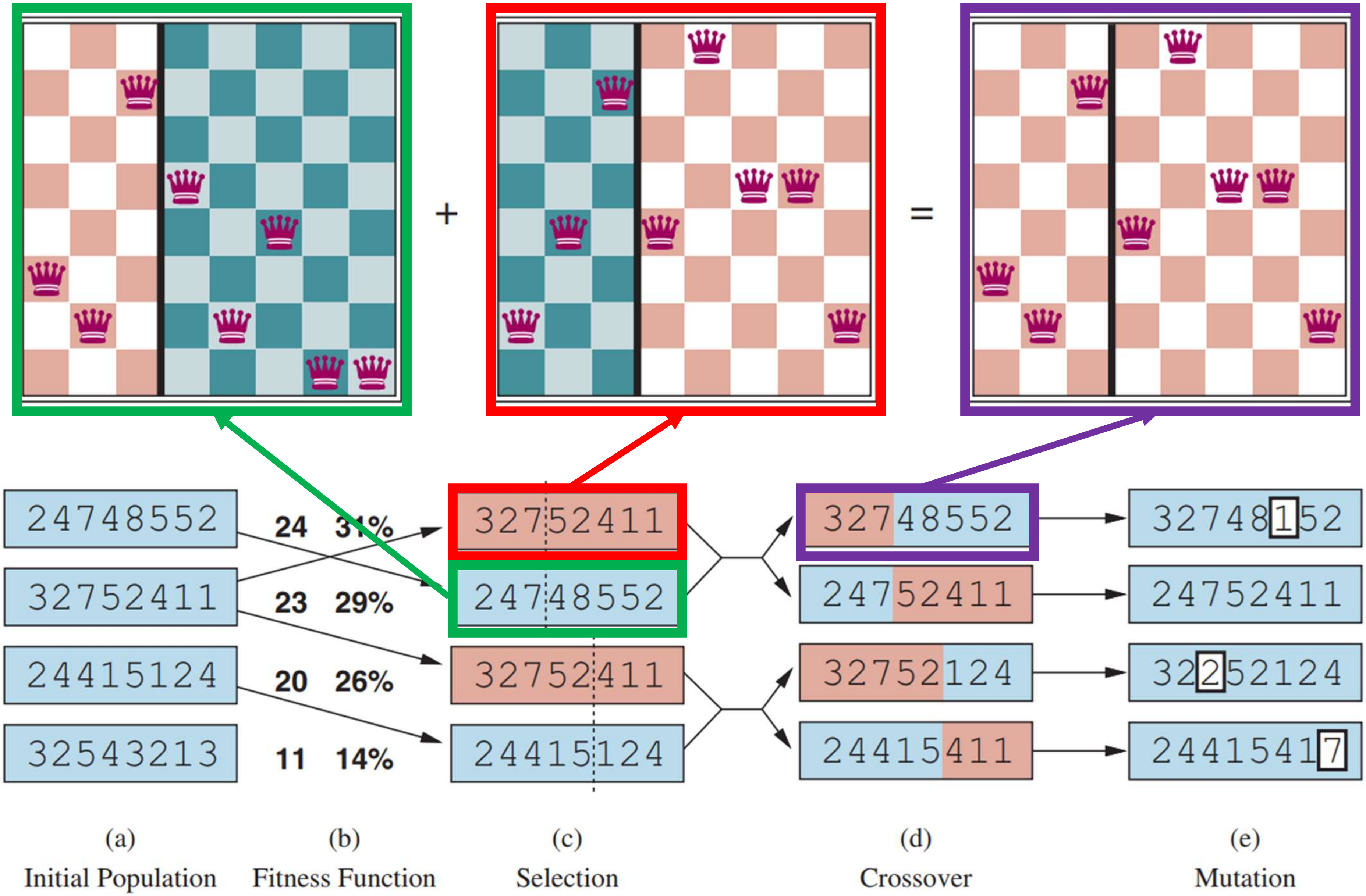
Genetic Algorithm: 8-Queens Problem





(a) Initial Population
(b) Fitness Function
(c) Selection
(d) Crossover
(e) Mutation

Genetic Algorithm: 8-Queens Problem



Genetic Algorithms: Design Issues

Choosing basic implementation issues:

- **representation**
- **population size, mutation rate, ...**
- **selection, deletion policies**
- **crossover, mutation operators**

- **Termination criteria**
- **Performance, scalability**
- **Solution is only as good as the evaluation function (often the hardest part)**

Genetic Algorithms: Benefits

- Easy to understand and implement
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always has an answer
  - answers gets better with time
- Inherently parallel → easily distributed

Genetic Algorithms: Benefits

- **Variety of ways to improve performance as knowledge about the problem domain is gained**
- **Can exploit historical / alternative solutions**
- **Can be easily integrated into hybrid applications**
- **Numerous problems solved using this approach**

When To Use Genetic Algorithms

- **Other solutions too slow or overly complicated (intractable mathematically)**
- **As an exploratory tool to examine new approaches / hypotheses**
- **Similar problem to others solved with GA**
- **Want to hybridize with an existing solution**
- **GA benefits match new problem requirements**

Selected GA Applications

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

Example: Genetic Algorithm

http://ostap0207.github.io/web-ga-tsp/

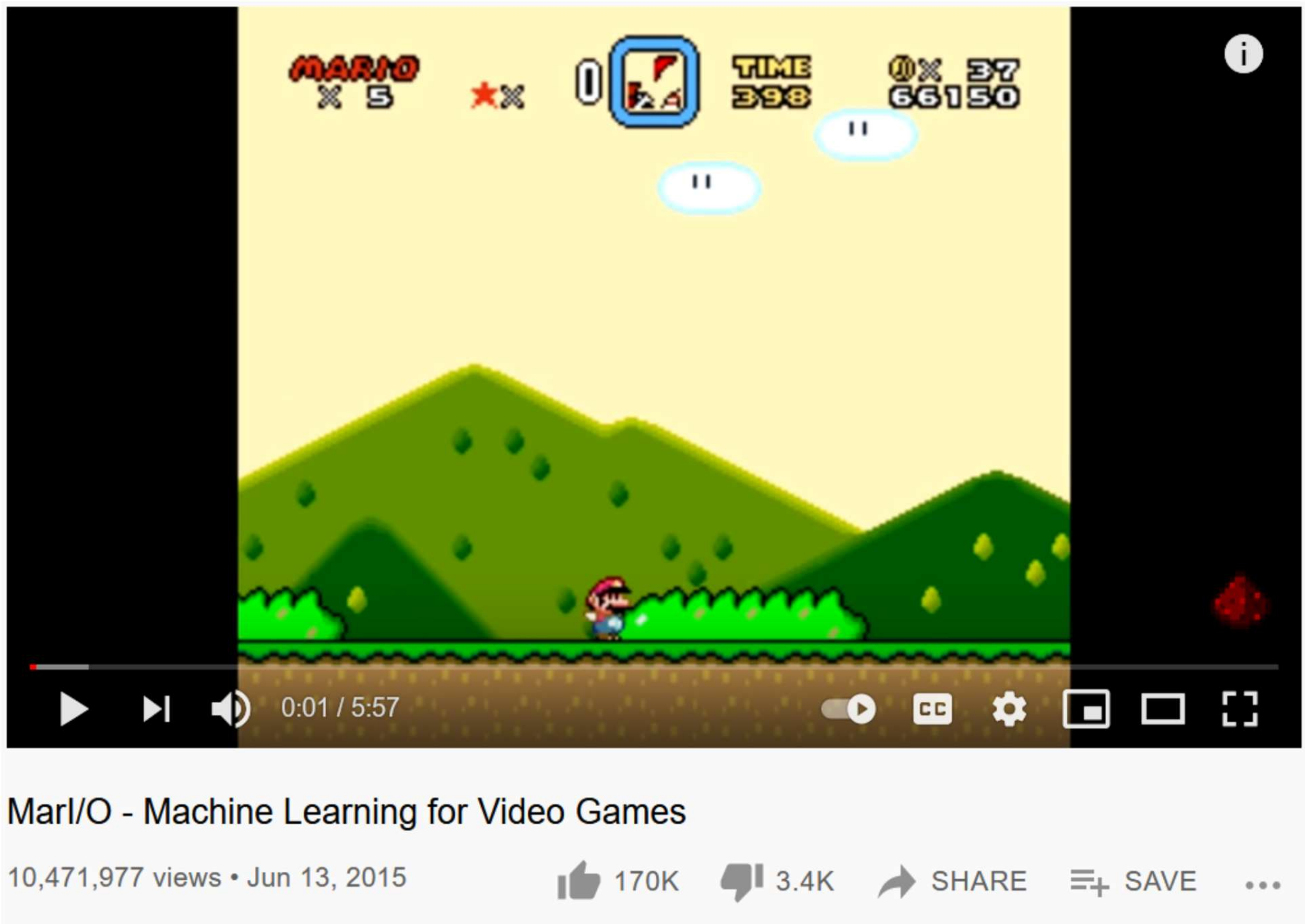
Example: Genetic Algorithm

https://chriscummins.cc/s/genetics/

Example: Genetic Algorithm

https://tarunbisht.com/evolution-visualizer/

Genetic Algorithm in Action



Source: https://www.youtube.com/watch?v=qv6UVOQ0F44