# LOCAL SEARCH

ARTIFICIAL INTELLIGENCE | COMP 131

## TODAY ON AI

- Hill climbing
- Simulated annealing
- Genetic algorithms
- Questions?

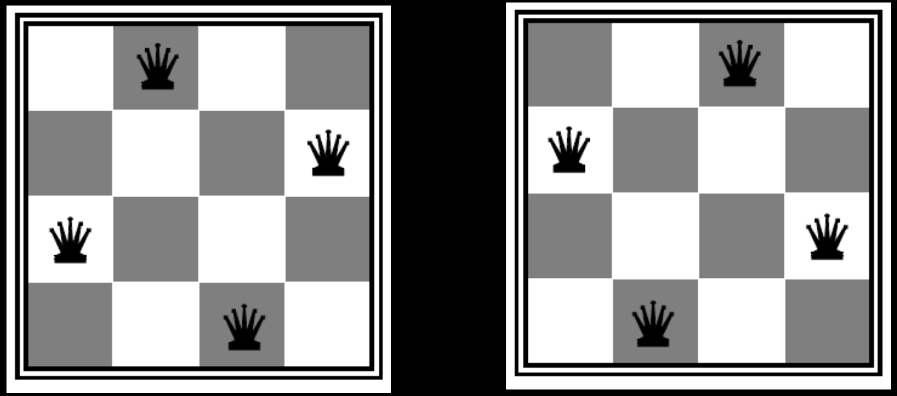
**Constraint satisfaction problems** (or **CSPs**) belong to a class of problems for which the goal itself is the most important part, not the path used to reach it.

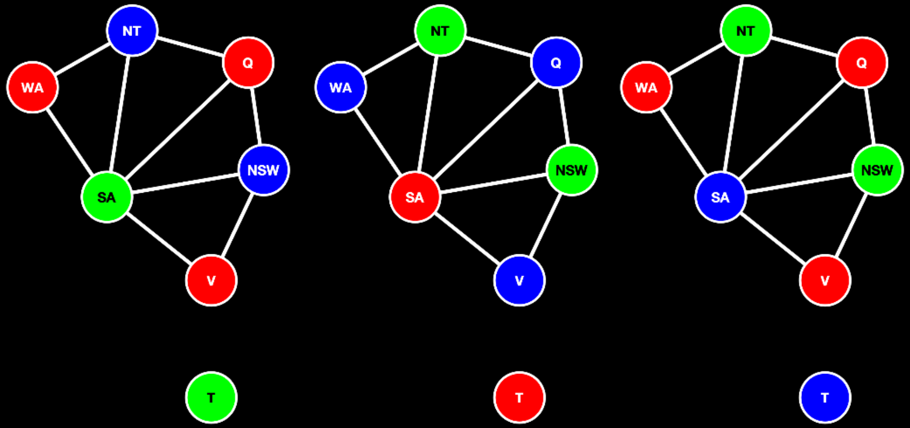
**EXAMPLES**

- Map coloring!
- Sudokus
- Crossword puzzles
- Job scheduling
- Cryptarithmetic puzzles
- N-Queens problems
- Hardware configuration
- Assignment problems
- Transportation scheduling
- Fault diagnosis
- More…

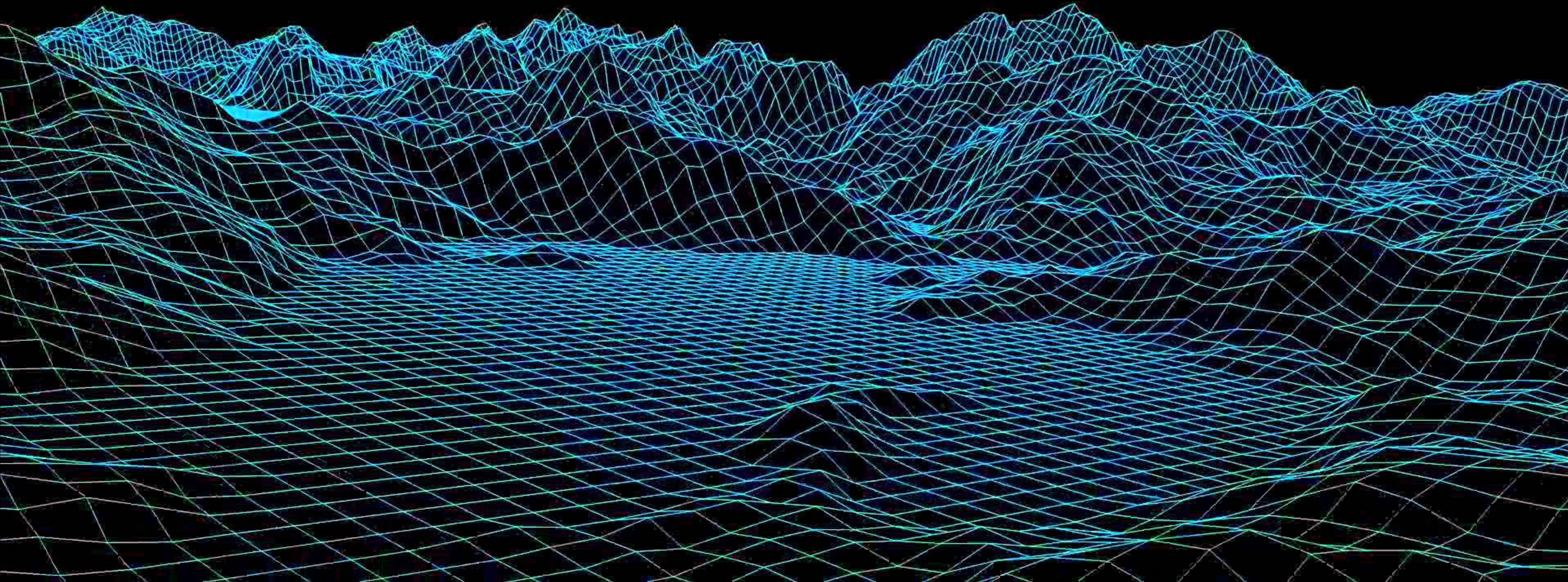
**Local search algorithms** aim to solve some Constraint Satisfaction Problems more efficiently. Specifically, they are tailored to find a solution to problems whose **search space is very big** or **infinite** without returning the actual path to the solution

They can **always** provide an answer to the problem, even if it is both not definitive nor correct.





Local search algorithms work best when a function that measure the **fitness** of the solution can be defined and the **fitness landscape** over the search space is continuous.



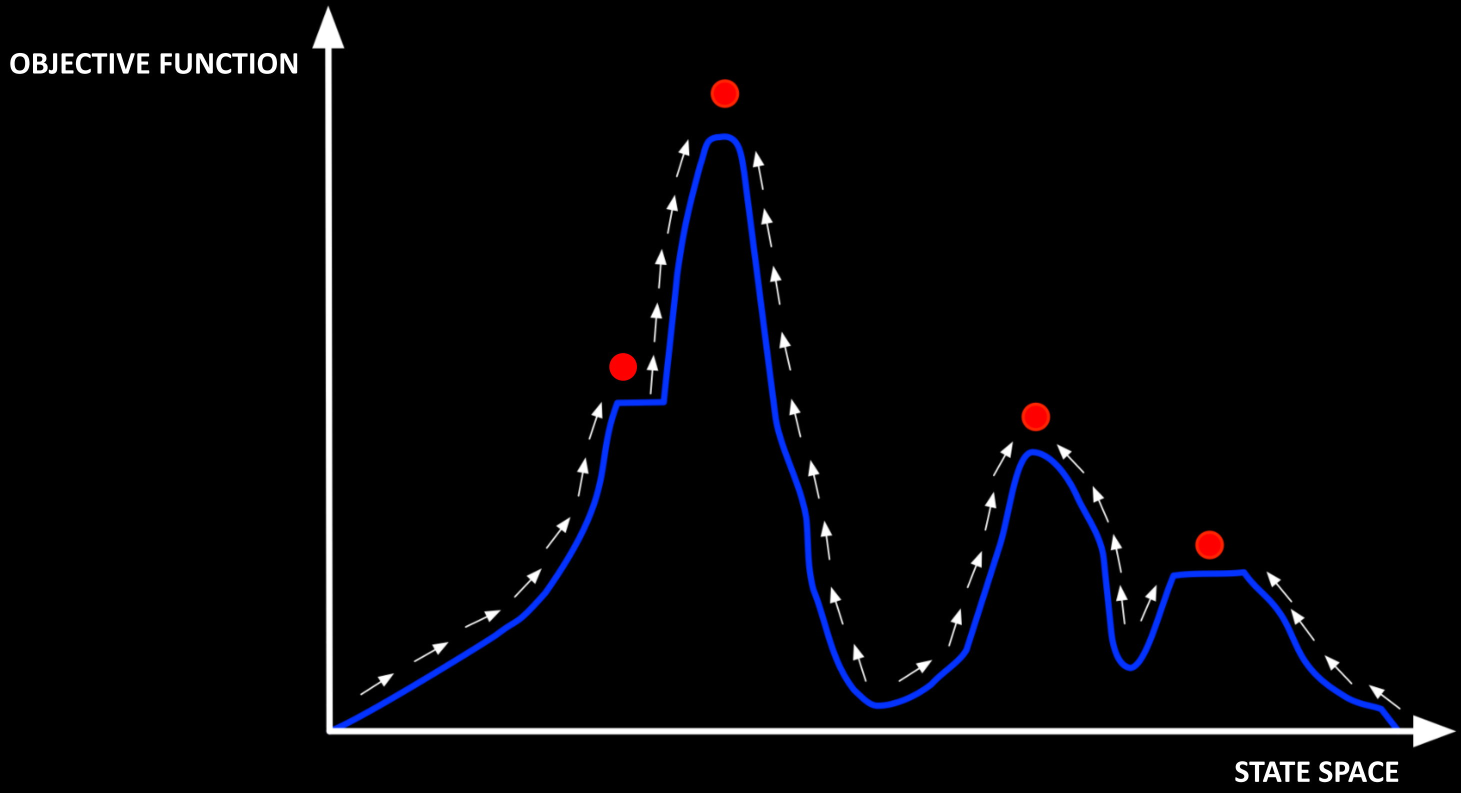
- This class of algorithms operate on **single current nodes** that represent the complete state of the search
- The **current state** is the only thing that matter
- The state is evaluated with a **fitness function**
- They **generally tend to move** through neighborhoods

| | |
|---|---|
| **GOOD** | Generally much faster for large or infinite state space. More memory efficient |
| **BAD** | Incomplete and suboptimal. Not systematic search |

SECTION 02

## Hill climbing algorithms

Hill climbing algorithm is the **most basic local search** technique. It is **greedy** in nature. At each step, the current node is **replaced** by the best neighbor.

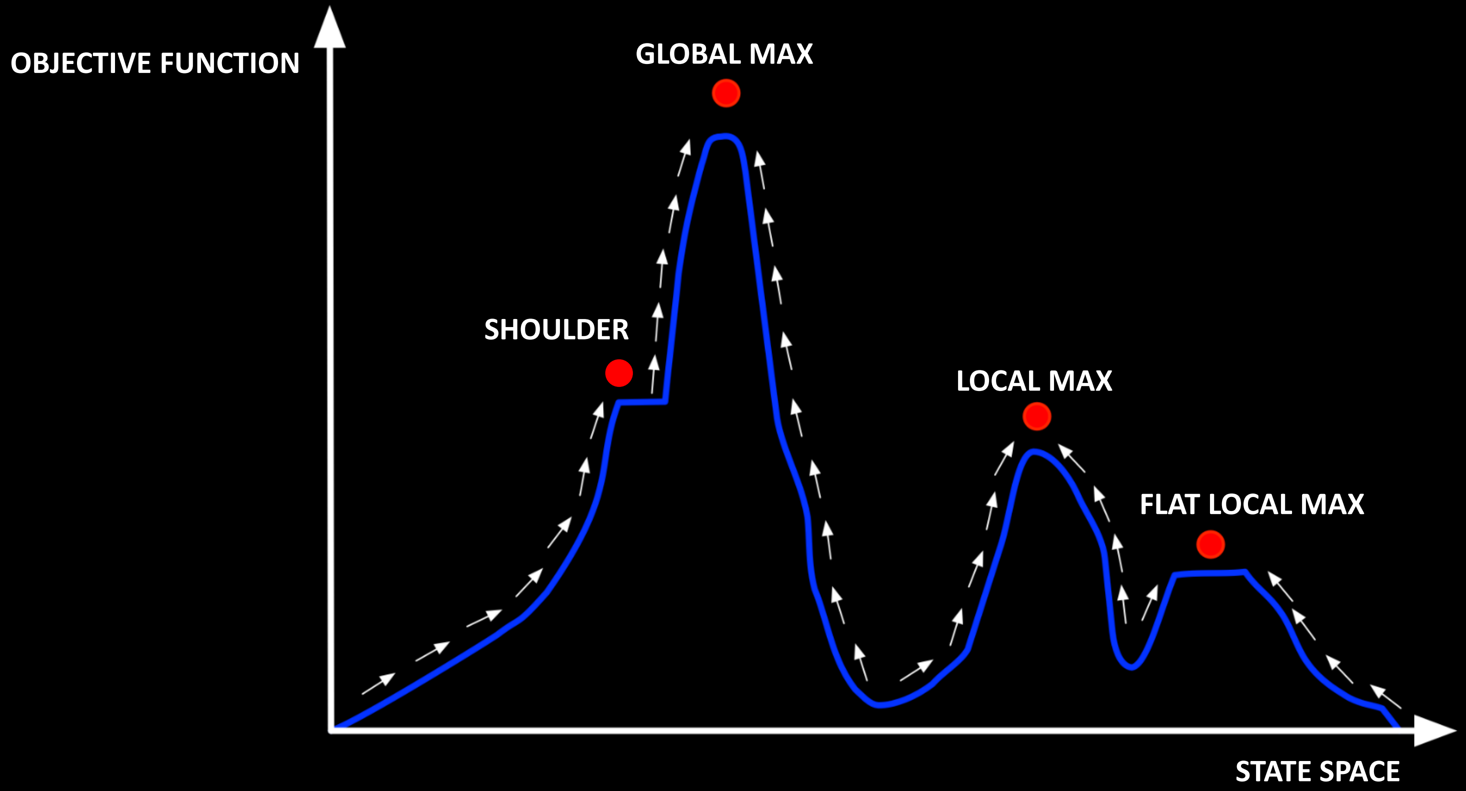


```
function Hill-climbing(PROBLEM) return SOLUTION, or FAILURE

current = Make-node(PROBLEM initial state)

loop do
  neighbor = a highest-valued successor of current
  if neighbor.value ≤ current value then
    return current state
  current = neighbor
```

**GOOD** Very simple to implement

**BAD** It can easily get stack in local maxima, ridges and plateau

- **Randomly choose** to initialize several times
- Implement hill climbing for **each initialization** and find the optimal
- If each hill climbing search has a **probability $p$ of success**, then the expected number of restarts required is $1/p$.

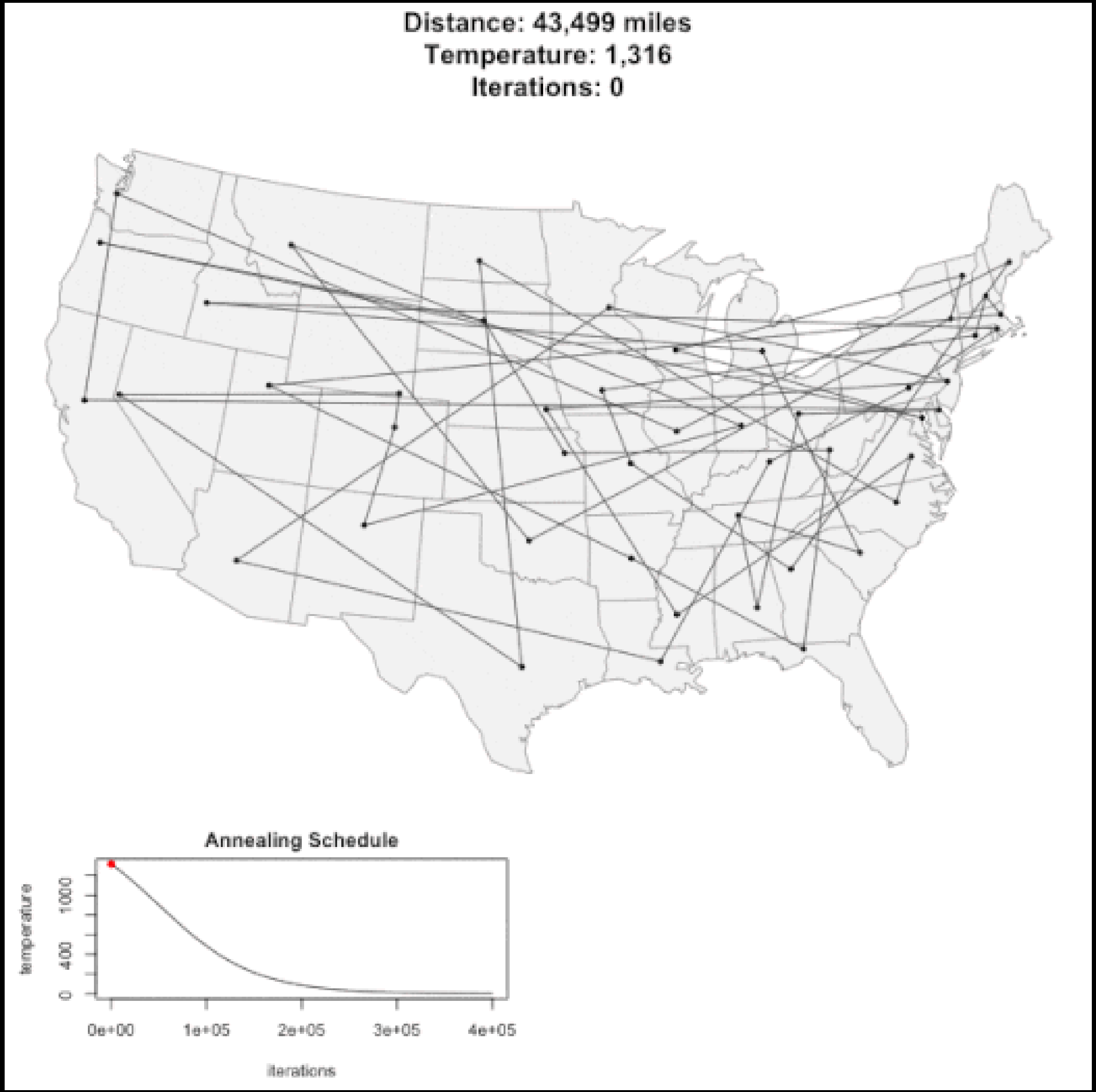


SECTION 02

## Simulated annealing

**Simulated annealing** is a class of algorithms that is inspired by statistical physics:

- **Annealing** is used in metal forging and glass making to aid the formation of crystal structures in the material
- The process **slowly reduces the temperature** the material to allow initial more random arrangements of atoms. At **lower temperatures,** the crystallin structure is more stable

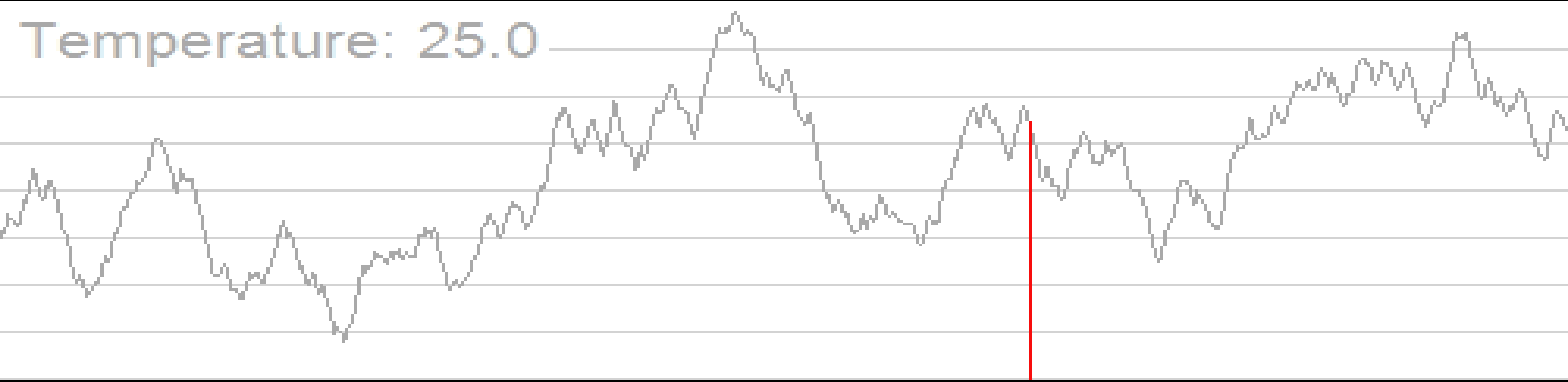


The **Traveling Salesman Problem** is a mathematical problem, formulated by W. R. Hamilton in the 1800s, in which one must find which is the **shortest route** which passes through each of a set of points once and only once.

## The Traveling Salesman Problem

Animation by shiny.rstudio.com

- The basic idea follows the **annealing physical metaphor**: select random successors with decreasing probability, also known as **temperature**.
- A gradient $\Delta E$ is calculated:
  - If $\Delta E > 0$ the new state is **accepted immediately** as an improvement
  - If $\Delta E < 0$ the new state is **accepted only with a probability** that depends on $\Delta E$ and $T$
- If $T$ decreases slowly enough, the algorithm will converge



```
function Simulated-annealing(PROBLEM, SCHEDULE) return SOLUTION, or FAILURE
current = Make-node(PROBLEM initial state)
loop do
  $T$ = SCHEDULE($t$)
  if $T$ = 0 then
    return current state
  next = a randomly selected successor of current
  $\Delta E$ = next value - current value
  if $\Delta E$ > 0 then
    current = next
  else
    current = next only with probability $e^{\frac{\Delta E}{T}}$
```

## Simulated annealing

Animation by Kingpin13, Wikimedia Commons.

SECTION 02

## Genetic algorithm

**Artificial evolution** includes a wide set of algorithms that take inspiration from the principles of natural evolution and molecular genetics in order to automatically find solutions to hard optimization problems.

Most artificial evolution is based on the very same pillars of **natural evolution**:

1. **Maintenance** of a population
2. **Creation** of diversity
3. **Selection** mechanisms
4. **Inheritance** processes

**Genetic algorithms** are a randomized heuristic search strategy that use a **natural selection metaphor** to find the best solution.

The genetic material of an individual is known as the **genotype**, whereas its manifestation as an organism is known as the **phenotype**.

For example, having brown hair is a **phenotype**; lacking the gene for hair color is a **genotype**.

The selection process is applied to a population that is composed of **candidate solutions** (or **phenotype**) of the problem.

- Possible solutions or hypothesis are described by a **phenotype** or **chromosome**
- Selection of an **initial population**
- A **fitness function** is used to simulate the natural selection process
- **Selection** and **reproduction** processes must be established
- **Genetic operators** must be selected
- A **solution test** is required if different from the fitness function
- **Evolutionary measures** can visually indicate the system evolution

A **genetic representation** or **genetic encoding** describes the elements of the genotype and how these elements are mapped into the phenotype:

A **suitable genetic representation** should be devised:

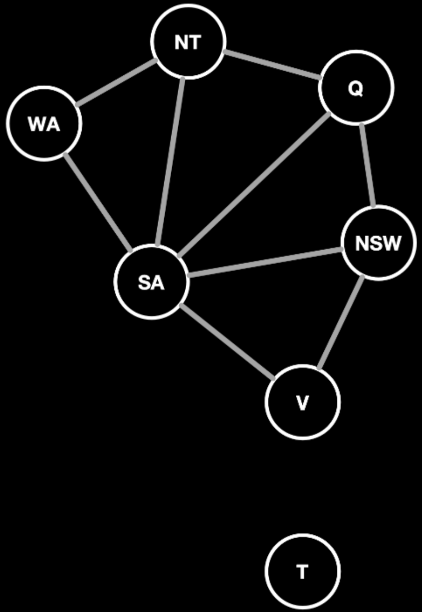
- The **recombination** and **mutation operators** have a high likelihood of generating increasingly better individuals
- The **set of all possible phenotypes** have a high likelihood of covering the space of optimal solutions

Genetic representations can be **discrete representations**; the genotype is described by a sequence of $l$ discrete values drawn from an alphabet with cardinality $k$:

| JOB | A.M. | P.M. |
|---|---|---|
| 1 | X | |
| 2 | | X |
| 3 | X | |
| 4 | | X |
| 5 | X | |
| 6 | | X |
| 7 | X | |
| 8 | X | |

| JOB 1 | JOB 2 | JOB 3 | JOB 4 | JOB 5 | JOB 6 | JOB 7 | JOB 8 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

## Discrete representations can also be used to describe **hypergraphs** or **sequences**:



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| GENOTYPE | WA | NT | Q | NSW | V | SA | T |
| PHENOTYPES | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| | R | G | R | B | R | B | B |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| GENOTYPE | S1 | S2 | S3 | S4 | S5 | S6 | S7 |
| PHENOTYPES | WA | NT | Q | NSW | V | SA | T |
| | NT | Q | WA | NSW | T | V | SA |

Genetic representations can be **real-valued representations**; the genotype consists of a set of n **numbers** belonging to the domain of **real numbers** (floating-point or integers):

| BIT 8 | BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

= 84 or 0.328125

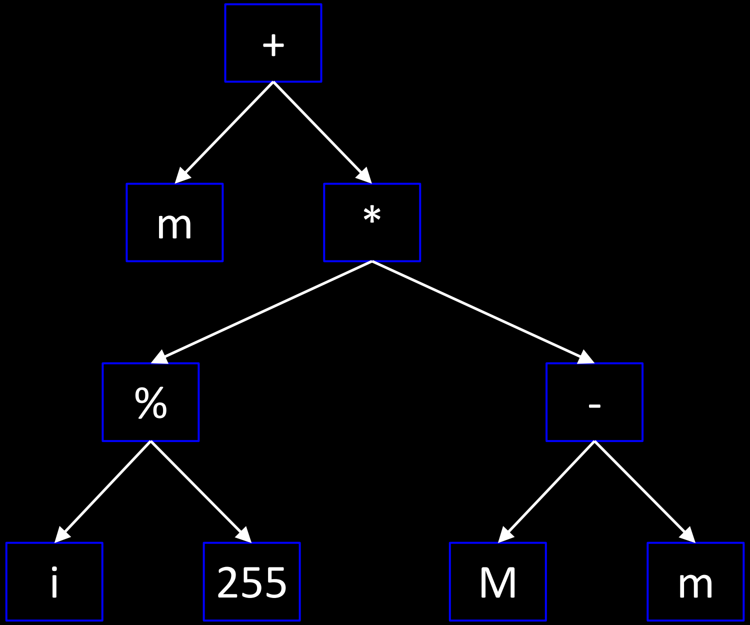
Floating-point numbers between a minimum $m$ and a maximum $M$ can be decoded from a sequence of $n$ bits as:

$$r = m + \frac{i}{2^n}(M - m)$$

A genetic representation can be a **tree-based representation**; the genotype is composed of a finite set of functions and of a finite set of terminals:

(**+**, **m**, (*****, (**/**, **i**, **255**), (**-**, **M**, **m**)))

$$= m + \frac{i}{255}(M - m)$$



The random **initial population** of individuals should be sufficiently large and diverse to ensure that individuals display different fitness values.

**How large** a population should be dependent on:

- The properties of the search space at hand
- The computational cost of evaluating all the individuals for several generations

In most cases the initial population size is determined by **rule of thumb** or **computational cost**.

The population size **must** always be kept constant.

A **fitness function** associates each phenotype with a numerical score.

Two important aspects in the design of a fitness function:

- The choice and combination of fitness components
- The way in which the function is evaluated

Fitness function often attempt to optimize **multiple objectives**.

The evaluation of a fitness function can be **time consuming** if it requires the **physical synthesis or construction** of the individual.

The role of selection is to **allocate** a larger number of **offspring** to the **best individuals** of the population.

**Selection pressure** indicates the percentage of individuals that will generate offspring.

- **Proportionate selection**: The probability of an individual to make $n$ copies of its own genome is $Np(i)$ where, $N$ is the population, $f(i)$ is the fitness function (assumed strictly positive), and $p(i)$ is:

$$p(i) = \frac{f(i)}{\sum_j^N f(j)}$$

- **Rank-based selection**: It consists of ranking all individuals from best to worst and allocating reproduction probabilities proportional to the rank of the individual.

- **Truncated rank-based selection** (or **Culling**): It is a variation that consists of taking only the top n individuals in the ranked list and making the same number of offspring for each individual selection.

- **Tournament selection**: It consists of organizing a tournament among a small subset of individuals in the population for every offspring to be generated:
  1. Picks randomly k (tournament size) individuals
  2. The individual with the best fitness wins the tournament
  3. All k individuals are put back into the population and available for more tournaments

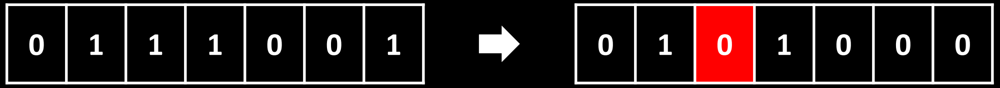
**Generational replacement** is a technique for which the newly produced offspring replace the entire population of individuals.

**Elitism** consists of maintaining the n best individuals from the previous generation.

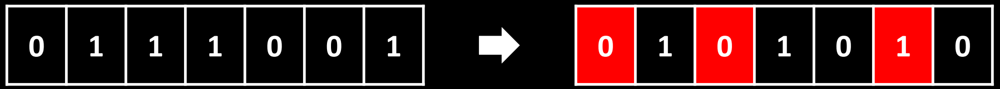
**Genetic operators** (also known as **fringe operations**) modify the genotype to obtain a new individual.

A **mutation**: Given an individual, randomly change the phenotype:

- **Single-point mutation**: a single gene is modified.

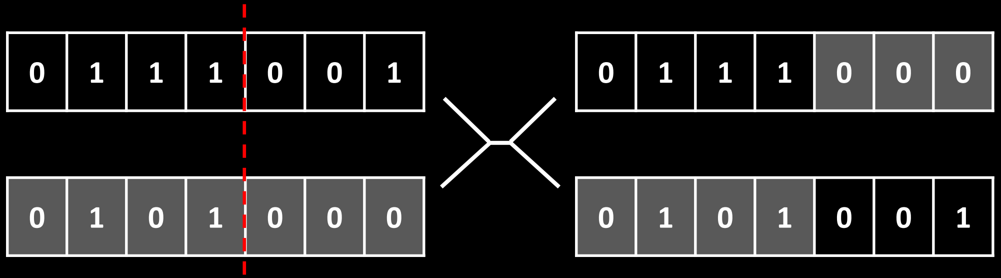


- **Multi-point mutation**: multiple genes are altered.

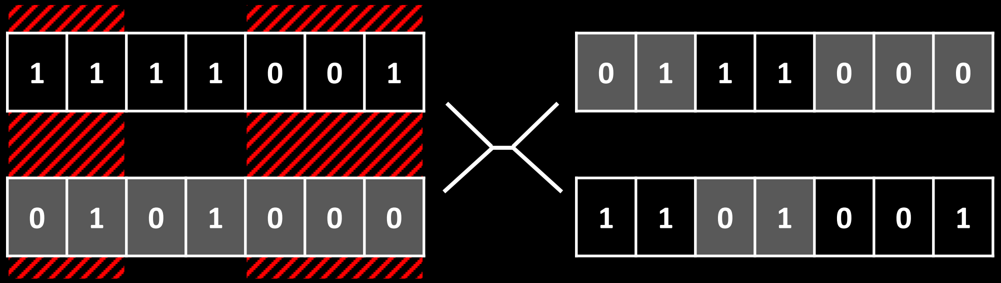


A **recombination** or **crossover**: Given two individuals, produce a new one that has elements of each:

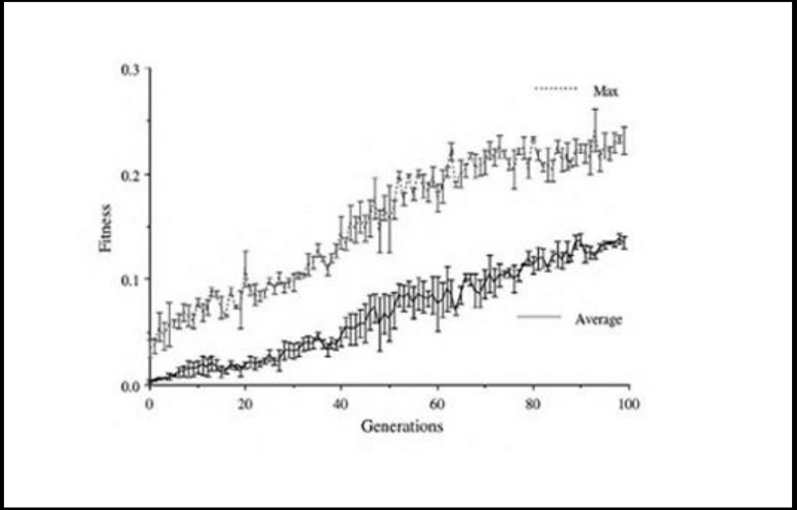
- **One-point crossover**: It consists of randomly selecting a point on each of the two individuals and swapping the genetic material around this point.



- **Multi-point mutation**: It consists of randomly selecting $n$ points on each of the two individuals and swapping the sections of genetic material between these points.



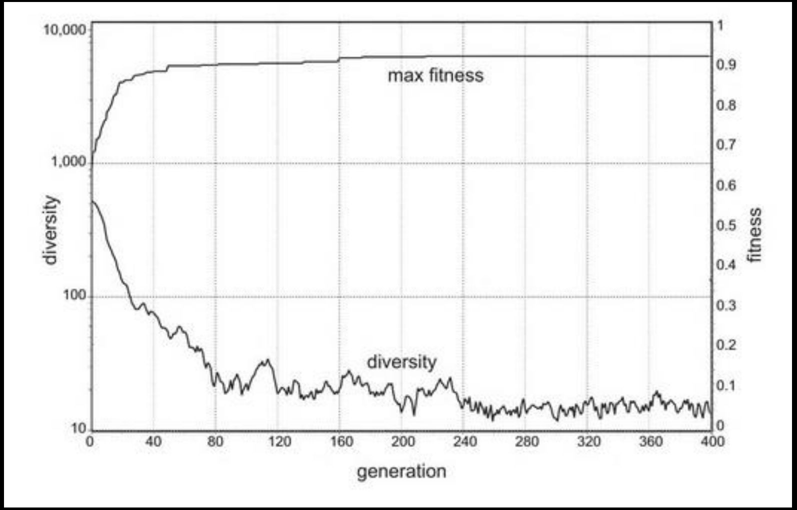
- **Fitness graph**: It shows  the average and best fitness of the population across generations. Each data point can show the average and best fitness over multiple runs from different initial conditions.



- **All-possible-pairs diversity**: It shows the all-possible pairs diversity measure across generations:

$$D_a(P) = \sum_{i,j \in P} d(g_i, g_j)$$

where $d(g_i, g_j)$ is a **Euclidean** or **Hamming** distance between two different genotypes.



Images from Bio-Inspired Artificial Intelligence

1. Start with a random population
2. Apply a **fitness function** to recognize the fittest individuals
3. $N$ hypotheses are selected for reproduction
4. Apply one or more **fringe operations** to generate a new population
5. Apply the **solution test** to the best candidate (if necessary)
6. Start over if the solution is "**good enough**"

```
function GENETIC-ALGORITHM(population, FITNESS-FN) return an individual
  repeat
    new population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
        add child to new population
        population ← new population
    until some individual is fit enough, or enough time has elapsed
    return the best individual in population, according to FITNESS-FN

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

**GOOD**

- Faster and with lower memory requirements
- It can explore a very large search space
- Easy to design

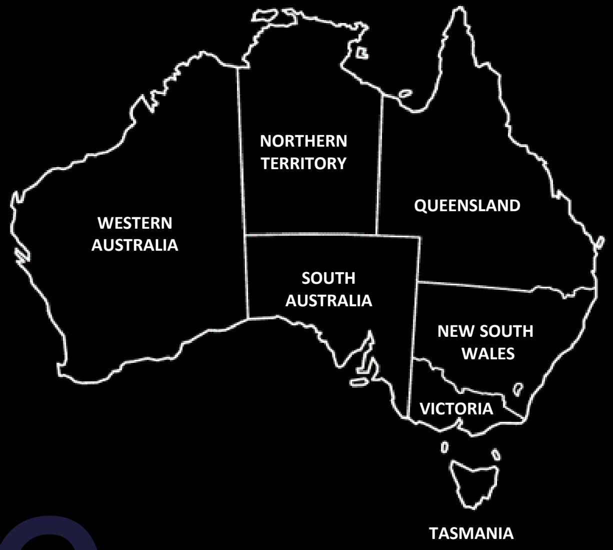
**BAD**

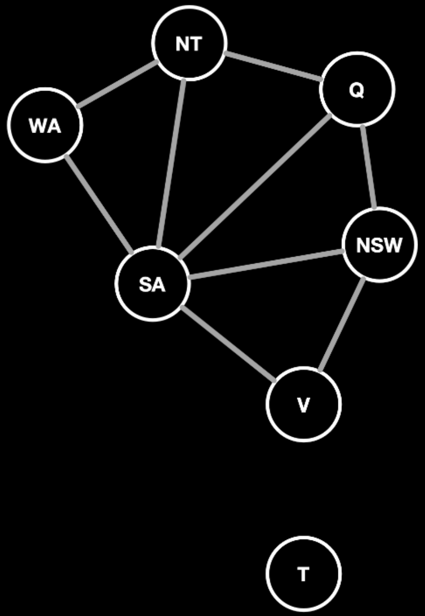
- Randomized – not optimal or even complete
- Can get stuck on local maxima, though mutation can help mitigate this
- It can be hard to design a chromosome

- **STATES**
  Color the Australia map so that neighboring regions do not match
- **CHROMOSOME**

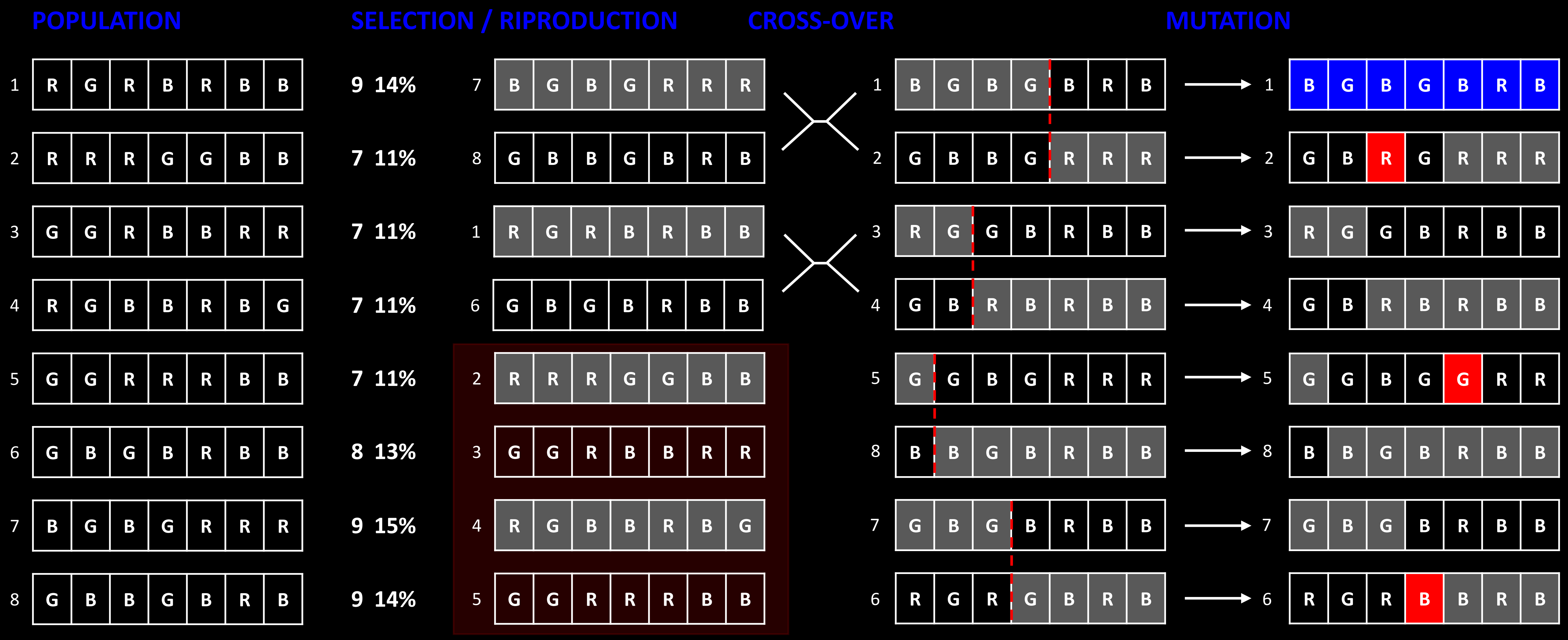
| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|

- **FITNESS FUNCTION**
  Number of pairs of regions that do not violate the constraint (max value 10)





| # | INDIVIDUAL | | | | | | | FITNES FUNCTION | NORMALIZED FITNES FUNCTION |
|---|---|---|---|---|---|---|---|---|---|
| 1 | R | G | R | B | R | B | B | 9 | 14% |
| 2 | R | R | R | G | G | B | B | 7 | 11% |
| 3 | G | G | R | B | B | R | R | 7 | 11% |
| 4 | R | G | B | B | R | B | G | 7 | 11% |
| 5 | G | G | R | R | R | B | B | 7 | 11% |
| 6 | G | B | G | B | R | B | B | 8 | 13% |
| 7 | B | G | B | G | R | R | R | 9 | 15% |
| 8 | G | B | B | G | B | R | B | 9 | 14% |
| | | | | | | | | 63 | 100% |



| POPULATION | | | | | | | | SELECTION / RIPRODUCTION | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | R | G | R | B | R | B | B | 9 | 14% |
| 2 | R | R | R | G | G | B | B | 7 | 11% |
| 3 | G | G | R | B | B | R | R | 7 | 11% |
| 4 | R | G | B | B | R | B | G | 7 | 11% |
| 5 | G | G | R | R | R | B | B | 7 | 11% |
| 6 | G | B | G | B | R | B | B | 8 | 13% |
| 7 | B | G | B | G | R | R | R | 9 | 15% |
| 8 | G | B | B | G | B | R | B | 9 | 14% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | B | G | B | G | R | R | R |
| 8 | G | B | B | G | B | R | B |
| 1 | R | G | R | B | R | B | B |
| 6 | G | B | G | B | R | B | B |
| 2 | R | R | R | G | G | B | B |
| 3 | G | G | R | B | B | R | R |
| 4 | R | G | B | B | R | B | G |
| 5 | G | G | R | R | R | B | B |

**Truncated rank-based selection** is applied, and the least fit individuals are eliminated.

| CROSS-OVER | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | B | G | B | G | B | R | B |
| 2 | G | B | B | G | R | R | R |
| 3 | R | G | G | B | R | B | B |
| 4 | G | B | R | B | R | B | B |
| 5 | G | G | B | G | R | R | R |
| 8 | B | B | G | B | R | B | B |
| 7 | G | B | G | B | R | B | B |
| 6 | R | G | R | G | B | R | B |

| MUTATION | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | B | G | B | G | B | R | B |
| 2 | G | B | R | G | R | R | R |
| 3 | R | G | G | B | R | B | B |
| 4 | G | B | R | B | R | B | B |
| 5 | G | G | B | G | G | R | R |
| 8 | B | B | G | B | R | B | B |
| 7 | G | B | G | B | R | B | B |
| 6 | R | G | R | B | B | R | B |

# READINGS

Chapters 4.1 – 4.6

Chapter 5

QUESTIONS ?

Tufts UNIVERSITY

# ARTIFICIAL INTELLIGENCE

COMP 131

FABRIZIO SANTINI

VERSION 4.1