Lecture 9: Intro to Evolutionary Computation

Artificial Intelligence
CS-GY-6613

Julian Togelius
julian.togelius@nyu.edu

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms: keep a single "current" state, try to improve it

Hill-climbing

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  inputs: *problem*, a problem
  local variables: *current*, a node
           neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] ≤ VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*

Types of hill-climbing

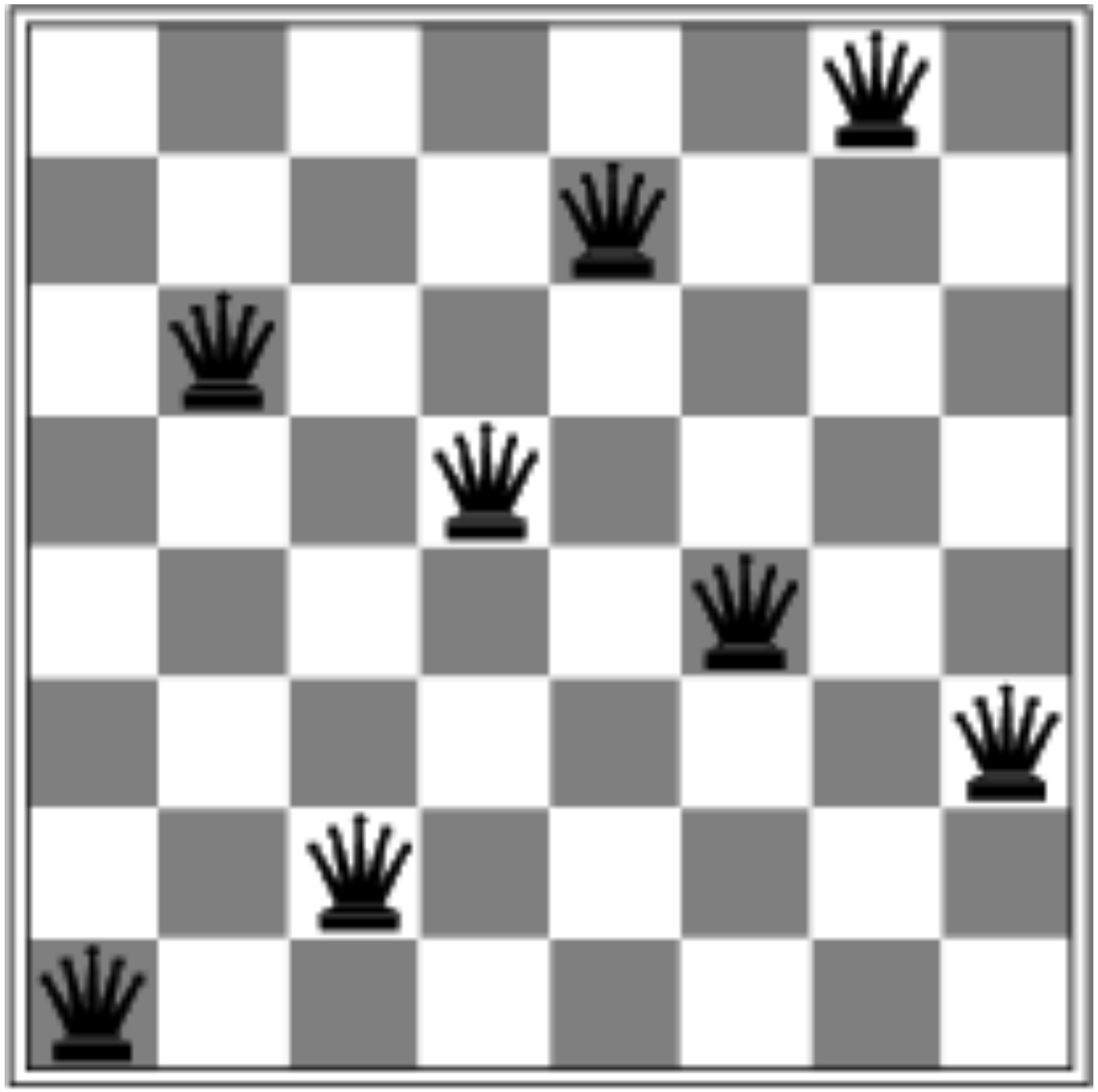
Hill-climbing is a somewhat generic concept, and there are several algorithms that fit the bill, such as:

- Stochastic hill-climbing: choose a random neighbor
- Steepest ascent hill-climbing: choose the best neighbor

n-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Local minimum



Other ideas…

- Exhaustive search: try all configurations, one after another
- Random search: generate random configurations

Random restarts

- When you stop making progress, start another hillclimber somewhere else
- Keep the best solution you found so far

Simulated annealing

- Do bad moves with decreasing probability

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(Δ E/T)
```

Limitations of local search

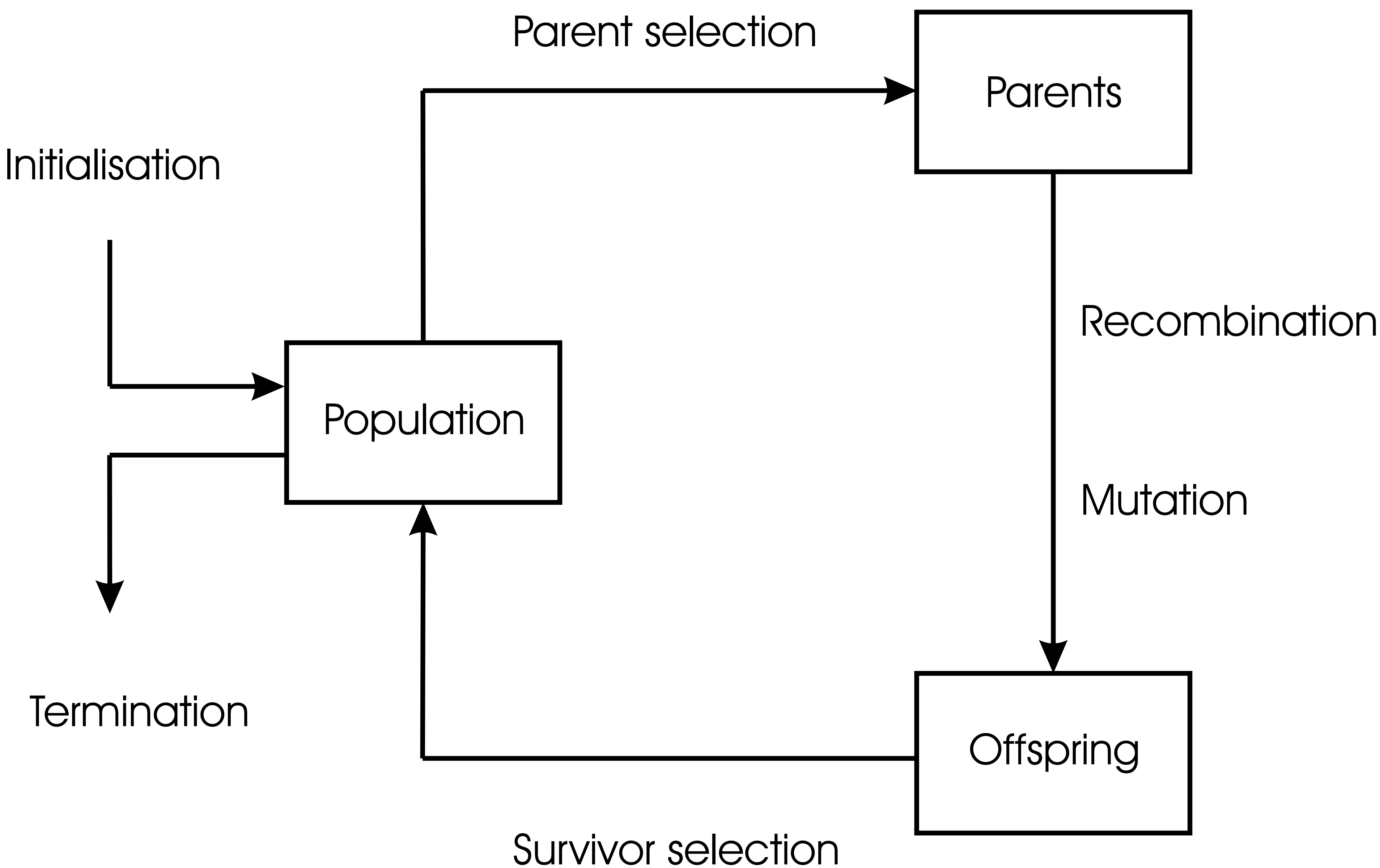
- Susceptible to local optima
- No sharing of information between parallel hill-climbing runs
- No possibility to learn from previous search experience

Evolutionary algorithms

- Stochastic global optimisation algorithms
- Inspired by Darwinian natural evolution
- Extremely domain-general, widely used in practice


© Reuters

Generic EA


Parent selection
Parents
Initialisation
Recombination
Population
Mutation
Termination
Offspring
Survivor selection

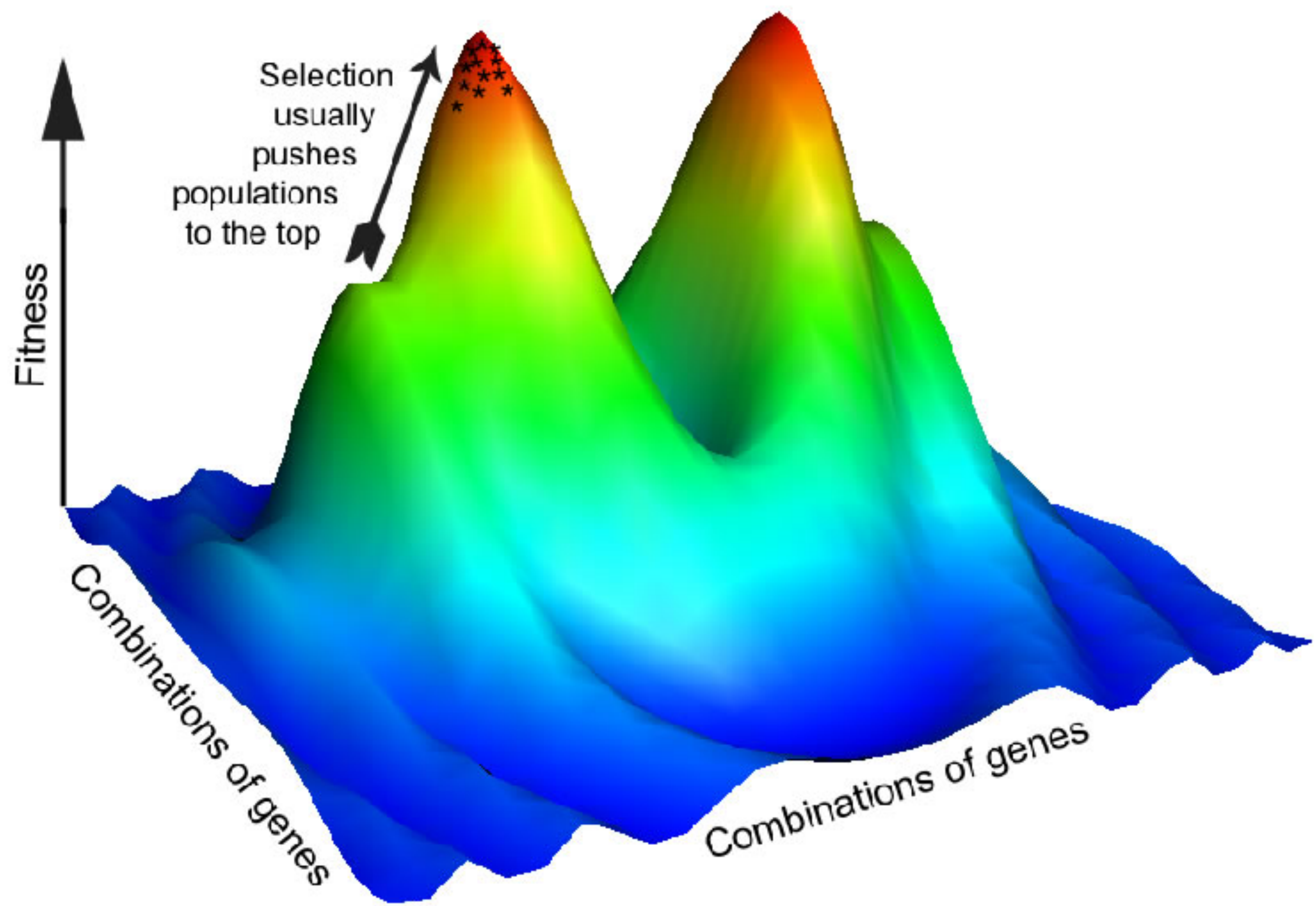
What's essential?

- Fitness evaluation
- Selection
  - Fitness influences how many offspring an individual has
- Variation
  - Could be mutation *and/or* crossover

Generic EA

```
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
```

The fitness landscape



Simple μ+λ Evolution Strategy

- Create a population of μ+λ individuals
- Each generation
    - Evaluate all individuals in the population
    - Sort by fitness
    - Remove the worst λ individuals
    - Replace with mutated copies of the μ best

Simple μ+λ ES (concrete)

- Create a population of μ+λ individuals (e.g. 50+50) represented as e.g. vectors of real numbers in [0..1]
- Each generation (until 100 generations)
    - Evaluate all individuals in the population
    - Sort by fitness (e.g. win rate or score for the game; higher is better)
    - Remove the worst λ individuals
    - Replace with mutated copies of the μ best (mutate through Gaussian mutation with mean 0 and s.d. 0.1)

Stochastic hillclimber = evolution strategy with $\mu=\lambda=1$

Example: OneMax

- Representation: binary strings of length 5
- Examples: 00000, 01001, 00100, 11101, 11111
- Fitness function: count the number of ones
- Mutation: flip a random bit

Example: OneMax

- Initial population: 00100, 11001, 10010, 01010
- Gen 1: 11001 : 3, 10010 : 2, 01010 : 2, 00100 : 1
  Selection: 11001 : 3, 10010 : 2
- Gen 2: 11101 : 4, 11001 : 3, 10010 : 2, 00010 : 1
  Selection: 11101 : 4, 11001 : 3
- Gen 3: 11111 : 5, 11101 : 4, 11101 : 4, 11001 : 3

Generally, you need

- A solution representation
- Variation operators (mutation and/or crossover)
- A fitness (evaluation) function