

# Optimisation Methods

## Metaheuristics

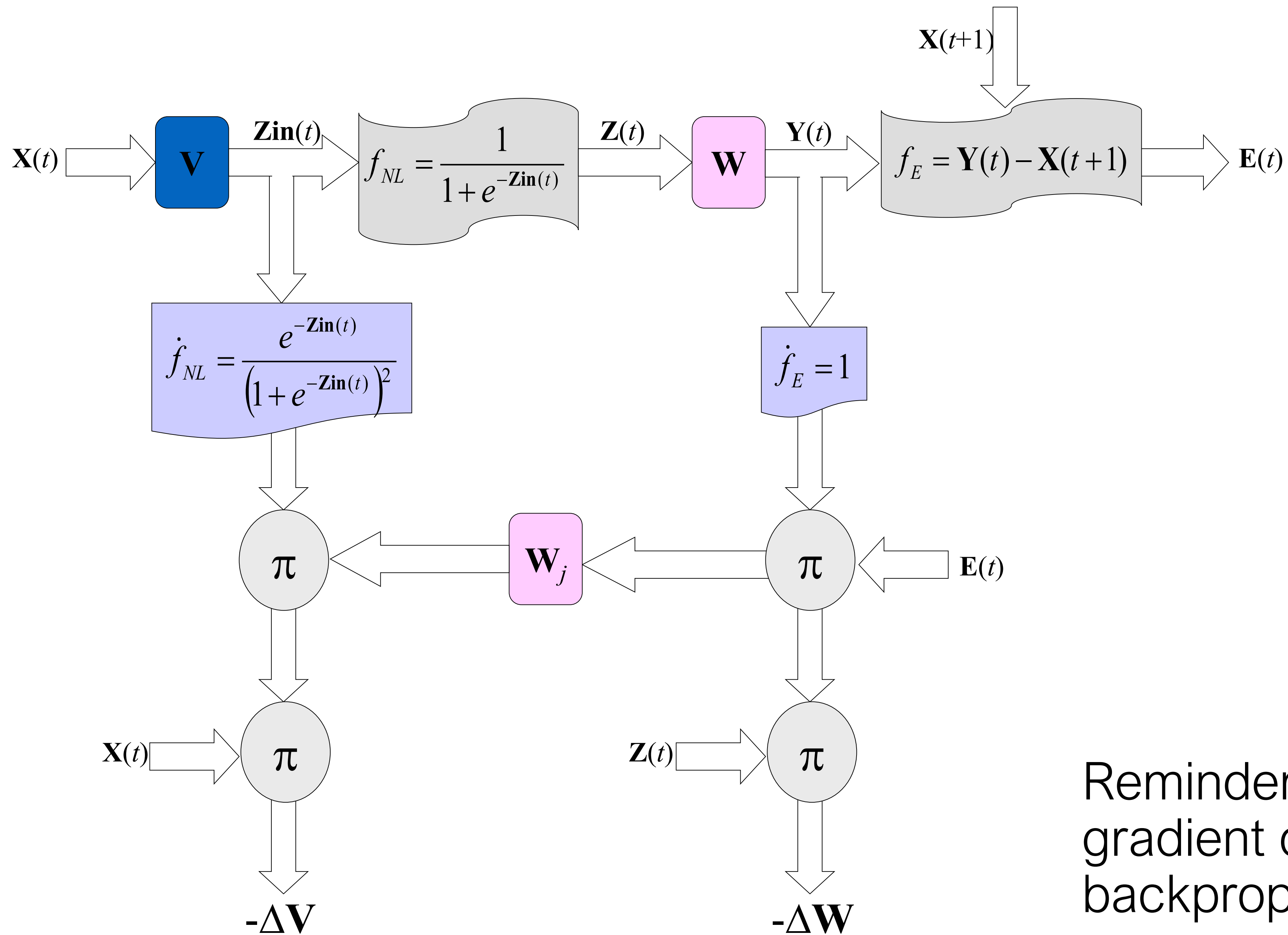
COMP9414: Artificial Intelligence

# Lecture Overview

- Optimisation methods
- Asymptotic complexity
- Classes of problems
- Search space
- Memoryless and memory-based metaheuristics
- Population-based methods

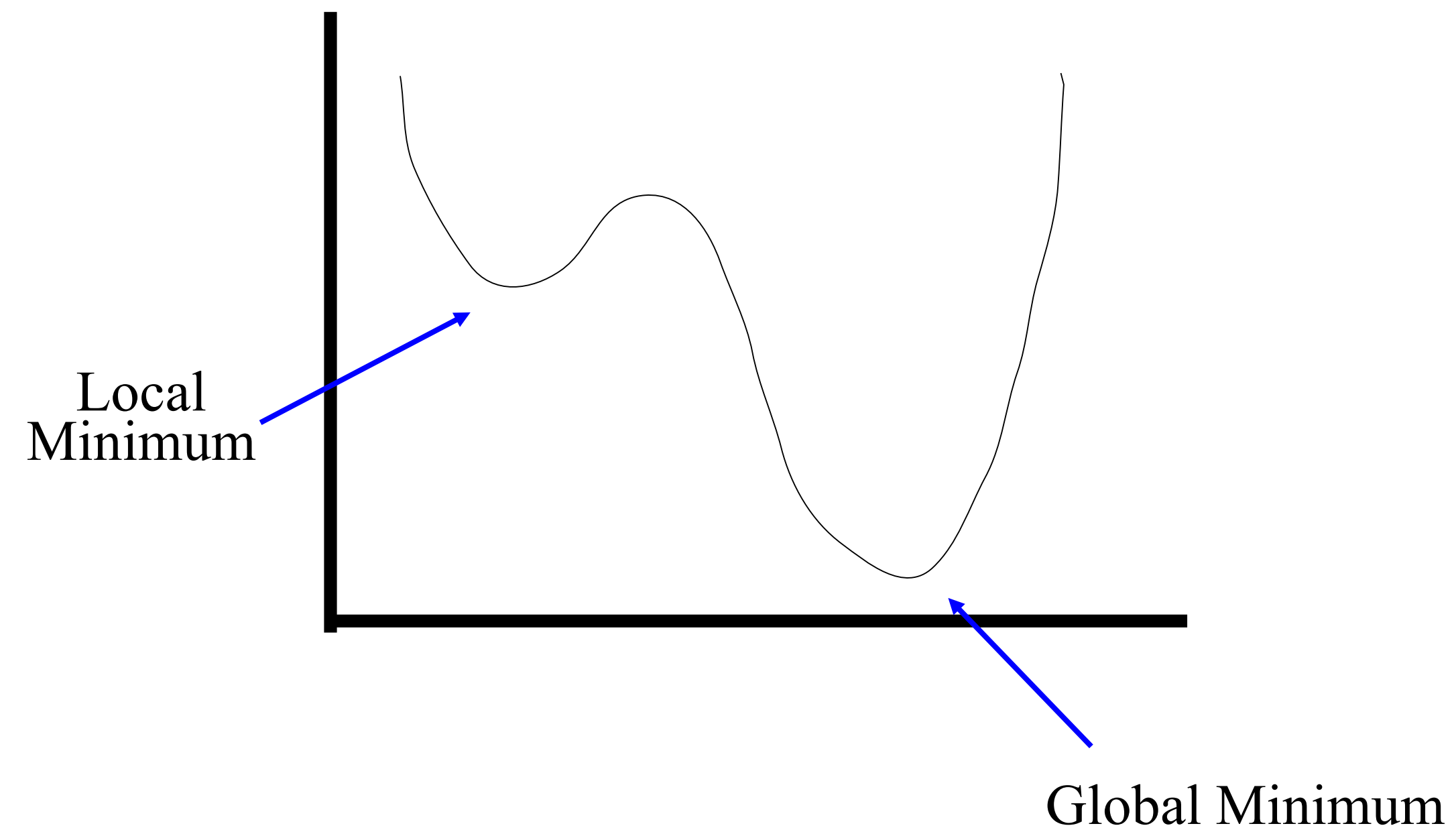
# Lecture Overview

- **Optimisation methods**
- Asymptotic complexity
- Classes of problems
- Search space
- Memoryless and memory-based metaheuristics
- Population-based methods



Reminder about the gradient descent and backpropagation.

# Optimisation Methods



# Optimisation Methods

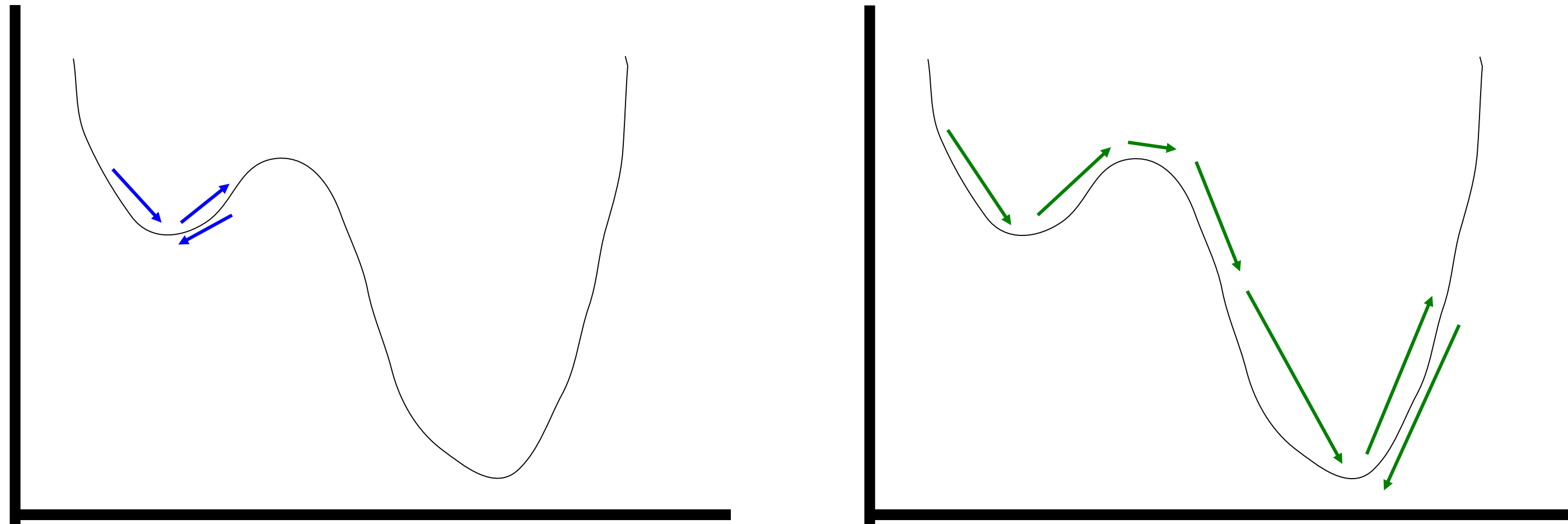
- Advantages of gradient descent:
  - Easy implementation.
  - Standard method that works generally well.
- Drawbacks of gradient descent:
  - Slow and inefficient.
  - It might get stuck on local minima leading to suboptimal results.

# Optimisation Methods

- Improvements to gradient descent:
  - Momentum: Add percentage of last movement to the actual one.
  - Stochastic batch (SGD): Estimate gradient using subsample set.
  - Adaptable estimation (Adam): Adapt the learning rate for each weight of the neural network.

# Optimisation Methods

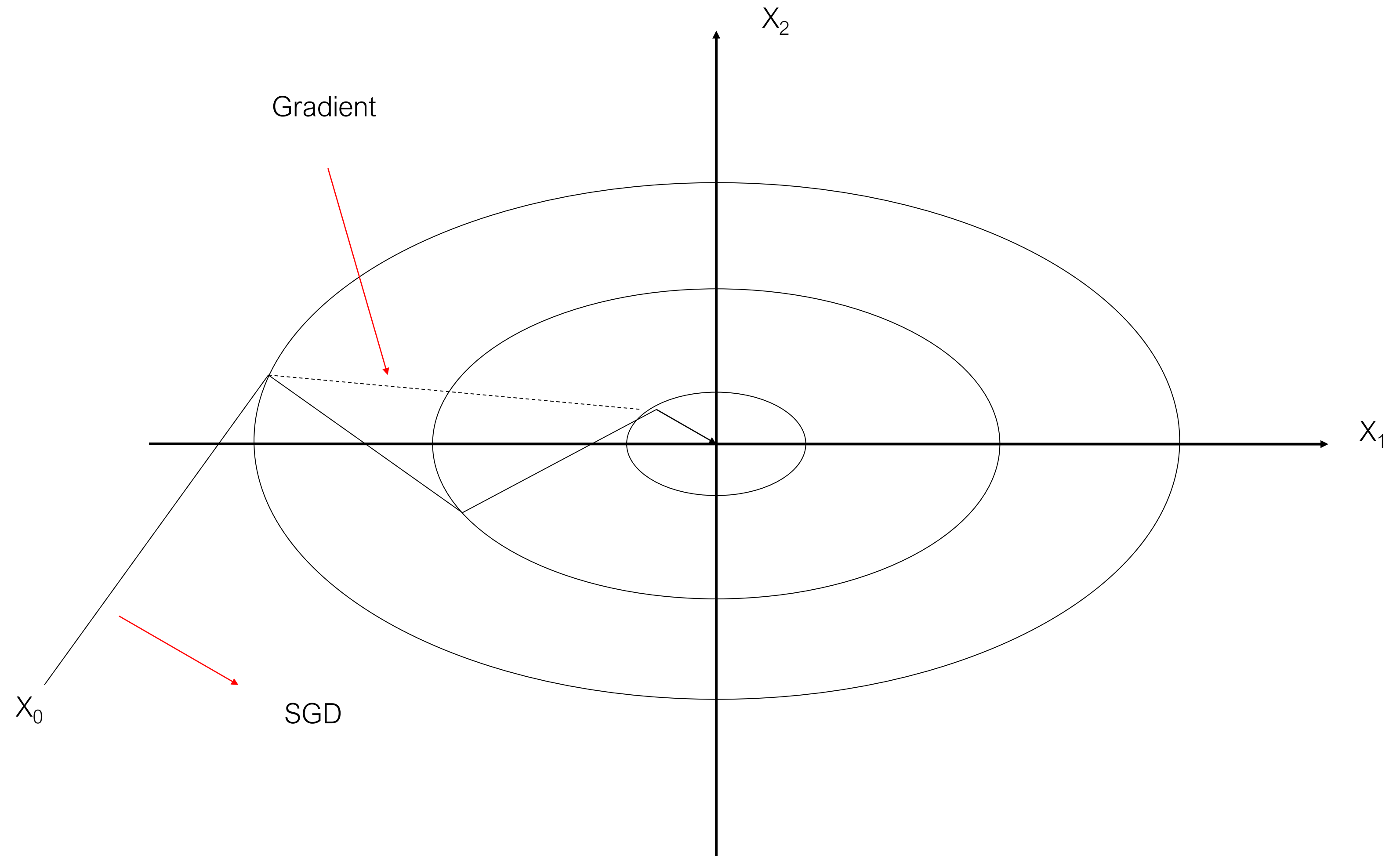
- Momentum: Add percentage of last movement to the actual one.





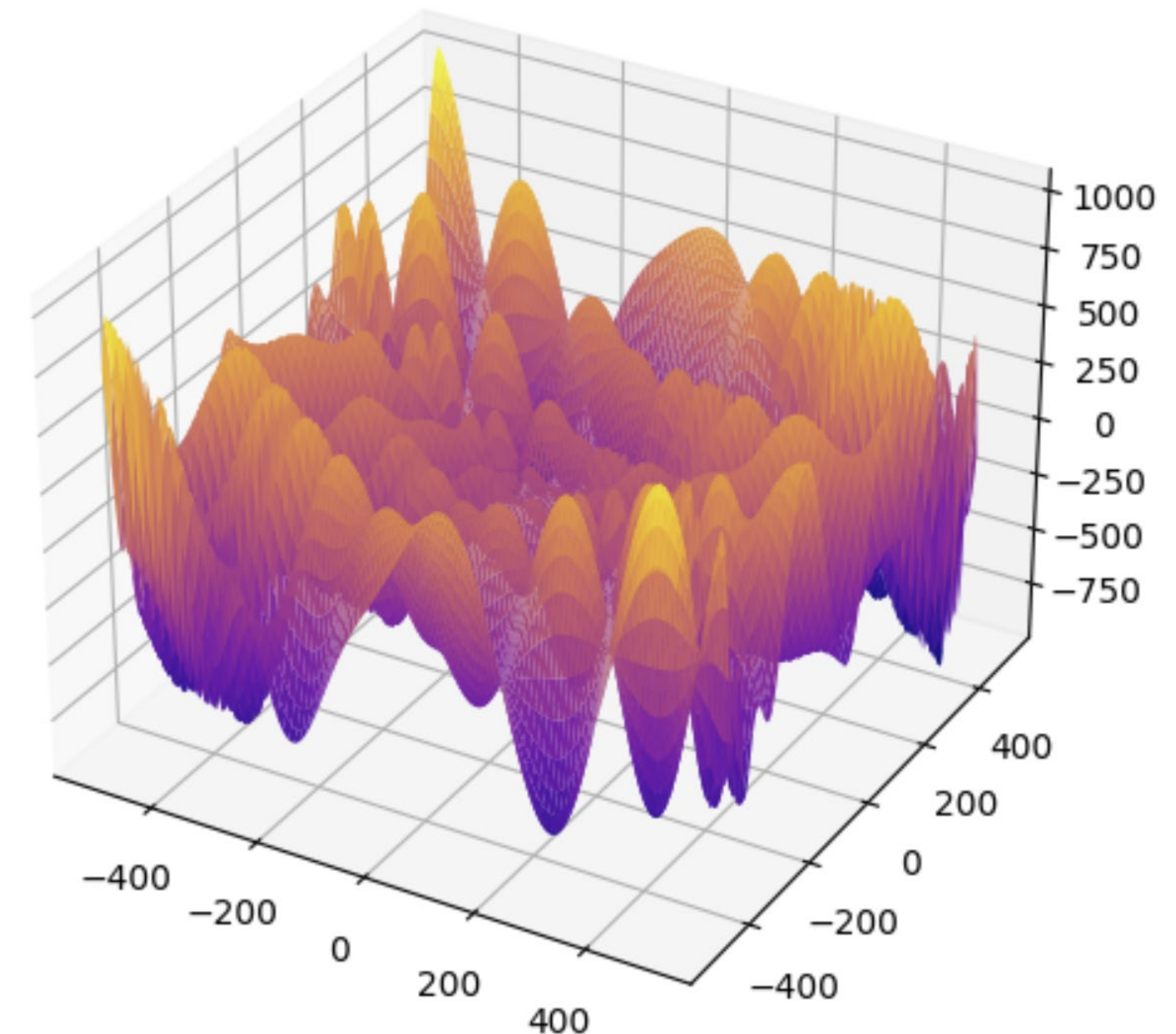
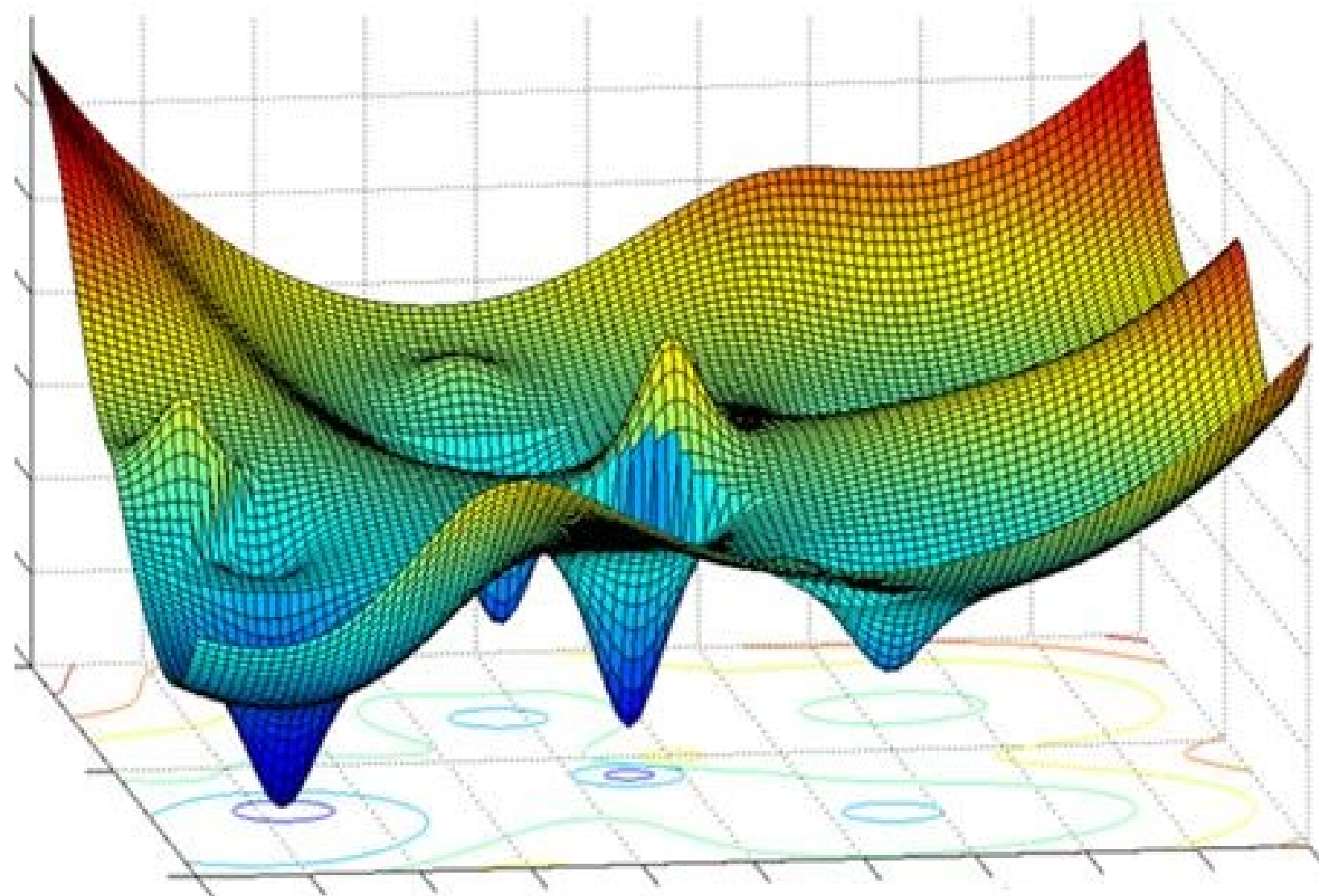
# Optimisation Methods

- Stochastic gradient descent.



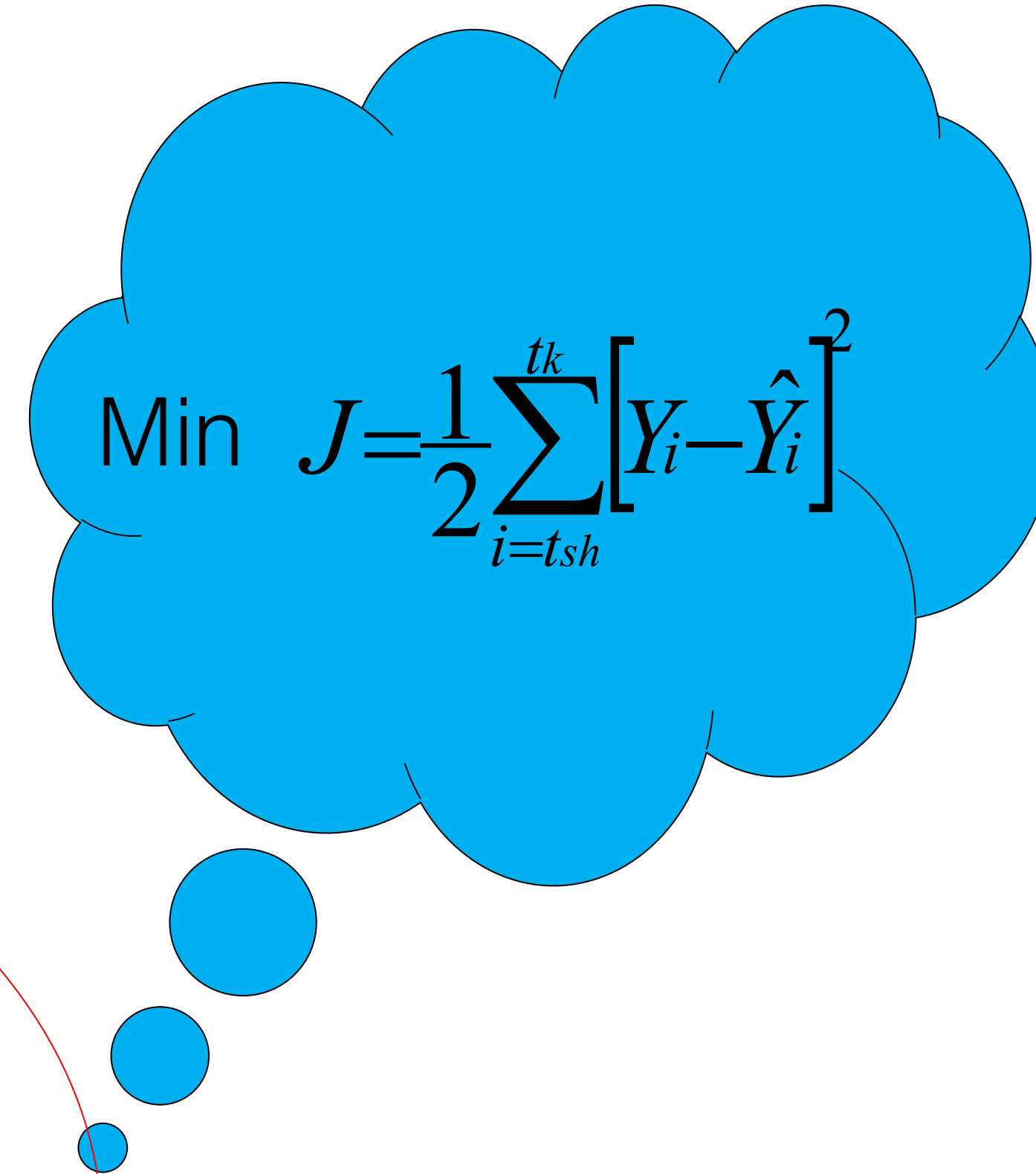
# Optimisation Methods

- The option to elegantly and efficiently calculate the gradient allows us to handle the optimization problem using the full range of optimization methods provided by nonlinear optimization systems.



# Optimisation Methods

- First-order gradient methods
- Second-order Newton's methods
- Single-solution metaheuristics
- Population-based algorithms
- ...etc.


$$\text{Min } J = \frac{1}{2} \sum_{i=tsh}^{tk} [Y_i - \hat{Y}_i]^2$$

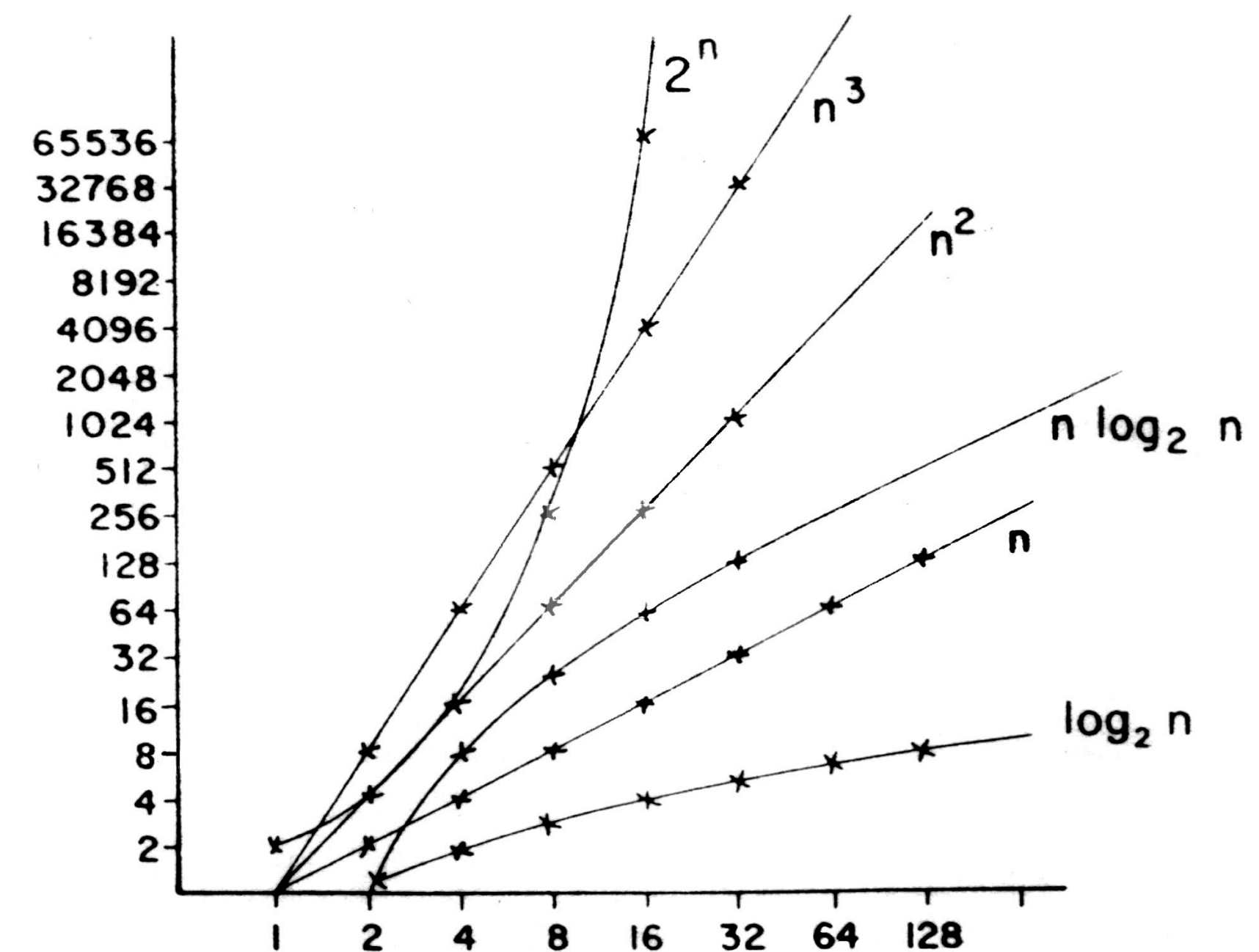
**Deterministic or Stochastics?**

# Lecture Overview

- Optimisation methods
- **Asymptotic complexity**
- Classes of problems
- Search space
- Memoryless and memory-based metaheuristics
- Population-based methods

# Asymptotic Complexity

- **Complexity:** The complexity of an algorithm is a measure of the amount of resources it consumes.
- **Resource:**
  - Time
  - Space
    - Memory
    - Drive



# Asymptotic Complexity

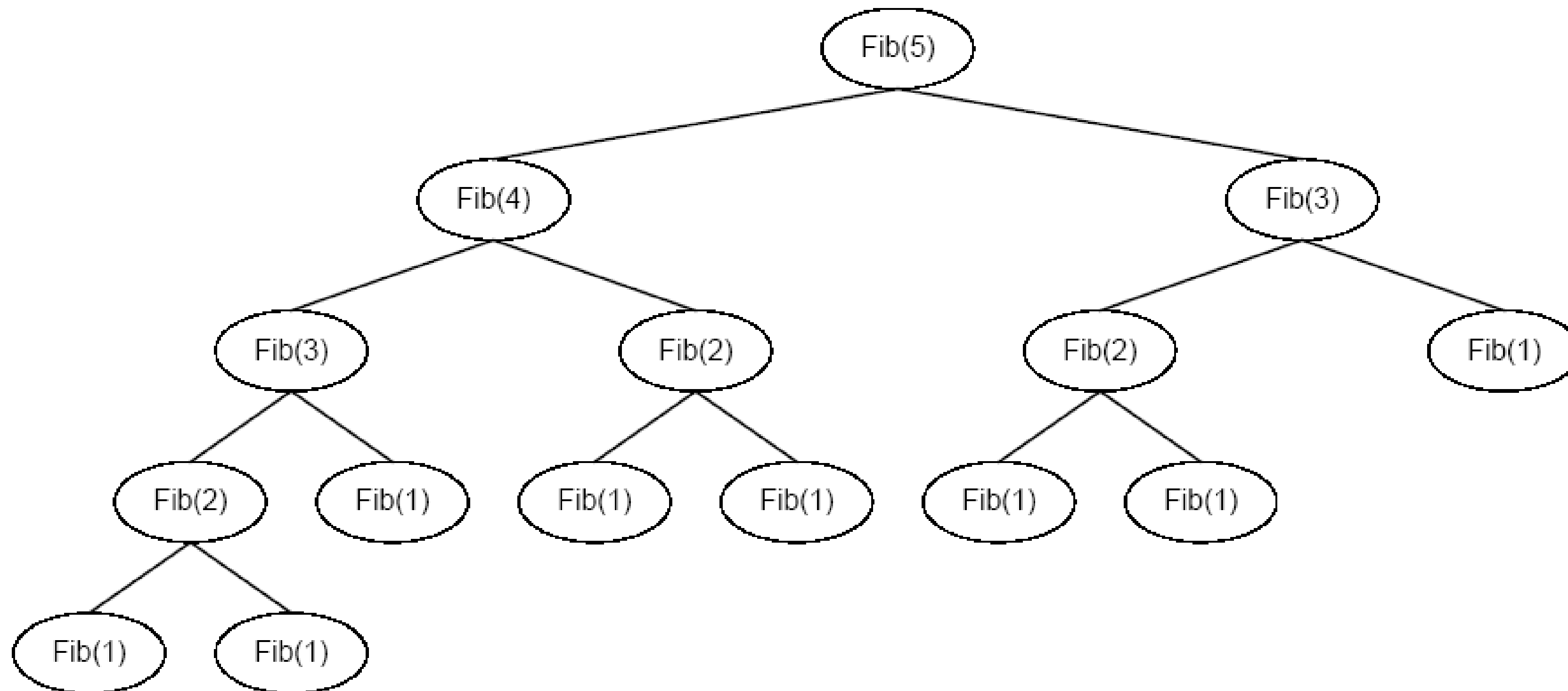
- **Example:** the complexity of a recursive algorithm to obtain the Fibonacci's series.

```
fibonacci(long N) {  
    if (N <= 1)  
        return 1;  
    else  
        return fibonacci(N-1)+fibonacci(N-2) ;  
}
```



# Asymptotic Complexity

- **Example:** the complexity of a recursive algorithm to obtain the Fibonacci's series.



# Asymptotic Complexity

- **Example:** the complexity of a recursive algorithm to obtain the Fibonacci's series.
  - $f(n) = 1 + f(n-1) + f(n-2) \quad \forall n \geq 2$
  - $f(n)$  is a growing function from  $n \geq 2$ , then:
  - $f(n) = 1 + f(n-1) + f(n-2) > 2 f(n-2)$
  - Using the same principle for  $f(n-2)$  y following  $f(n-i)$ , we obtain:
  - $f(n) > 2 \cdot 2 \cdot f(n-4) > 2 \cdot 2 \cdot 2 \cdot f(n-6) > \dots > 2^i f(n-2i) > 2^n$
  - $f(n) > 2^n$



# Asymptotic Complexity

- Assuming each term can be computed in 1 nano second ( $10^{-9}$  second)

N	Computed terms	Time
25	33,554,432	33 $\mu$ seconds
30	$1,1 \times 10^9$	1 second
35	$3,4 \times 10^{10}$	34 seconds
40	$1,1 \times 10^{12}$	18 minutes
45	$3,5 \times 10^{13}$	9 hours
50	$1,1 \times 10^{15}$	13 days
55	$1,4 \times 10^{17}$	4 years
60	$1,2 \times 10^{18}$	36 years
65	$3,6 \times 10^{19}$	11 centuries

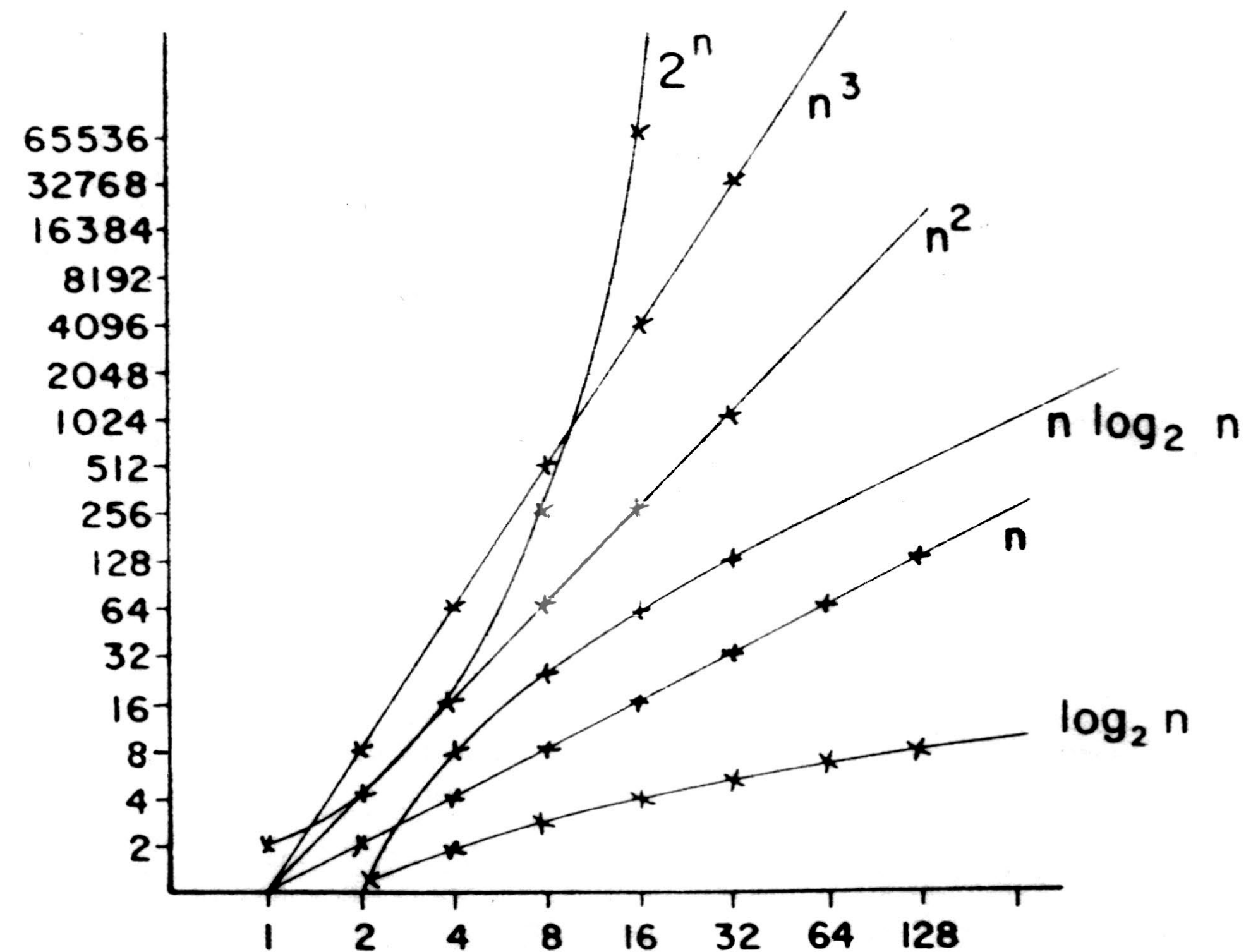
# Asymptotic Complexity

- Other examples

	$\log n$	$\sqrt{n}$	$n$	$n \log n$	$n^2$	$n^3$	$2^n$
	1	1,4	2	2	4	8	4
	2	2,0	4	8	16	64	16
	3	2,8	8	24	64	512	256
	4	4,0	16	64	256	4.096	65.536
	5	5,7	32	160	1.024	32.768	4.294.967.296
	6	8,0	64	384	4.096	262.144	$1,8 * 10^{19}$
	7	11,0	128	896	16.536	2.097.152	$3,4 * 10^{38}$

# Asymptotic Complexity

- Most common times for algorithms:
- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$



# Lecture Overview

- Optimisation methods
- Asymptotic complexity
- **Classes of problems**
- Search space
- Memoryless and memory-based metaheuristics
- Population-based methods

# Classes of Problems

- Class P:
  - A P problem can be solved in **polynomial time** in a deterministic computer.
  - The class P comprises problems that can be solved **quickly**.
  - **Examples:** Quicksort, binary search, matrix multiplication.
- Class NP:
  - An NP problem cannot be solved in a polynomial time using a deterministic computer (**intractable problem**).
  - The class NP comprises problems whose **solutions can be verified quickly** ( $P \subseteq NP$ ).
  - **Examples:** subsets addition, Sudoku, TSP.

# Classes of Problems

- Class NP-complete:
  - A class of problems for which it is **unknown if they are tractable**.
  - No one has found polynomial algorithms for any of them.
  - Some problems closely related to tractable problems.
  - All known algorithms for NP-complete problems **require exponential time** in relation to the size of the input.
  - In other words, they are **extremely difficult to solve**.
  - Examples include the Traveling Salesman Problem, which has a time complexity of  $O(n^2 2^n)$ .

# Classes of Problems

- Class NP-complete:

- This definition was proposed by Cook in 1971. He proved (Cook's theorem) that the **Boolean satisfiability problem** is NP-complete.
- Since then, it has been shown that **thousands of other problems** belong to this class, mostly by reduction from other problems that had already been proven to be NP-complete.
- Example: The Subset Sum problem.
  - Given a set  $S$  of integers, is there a non-empty subset of  $S$  whose elements sum to zero?
  - It is easy to verify if an answer is correct, but no better solution is known than exploring all  $2^n - 1$  possible subsets until finding one that satisfies the condition.

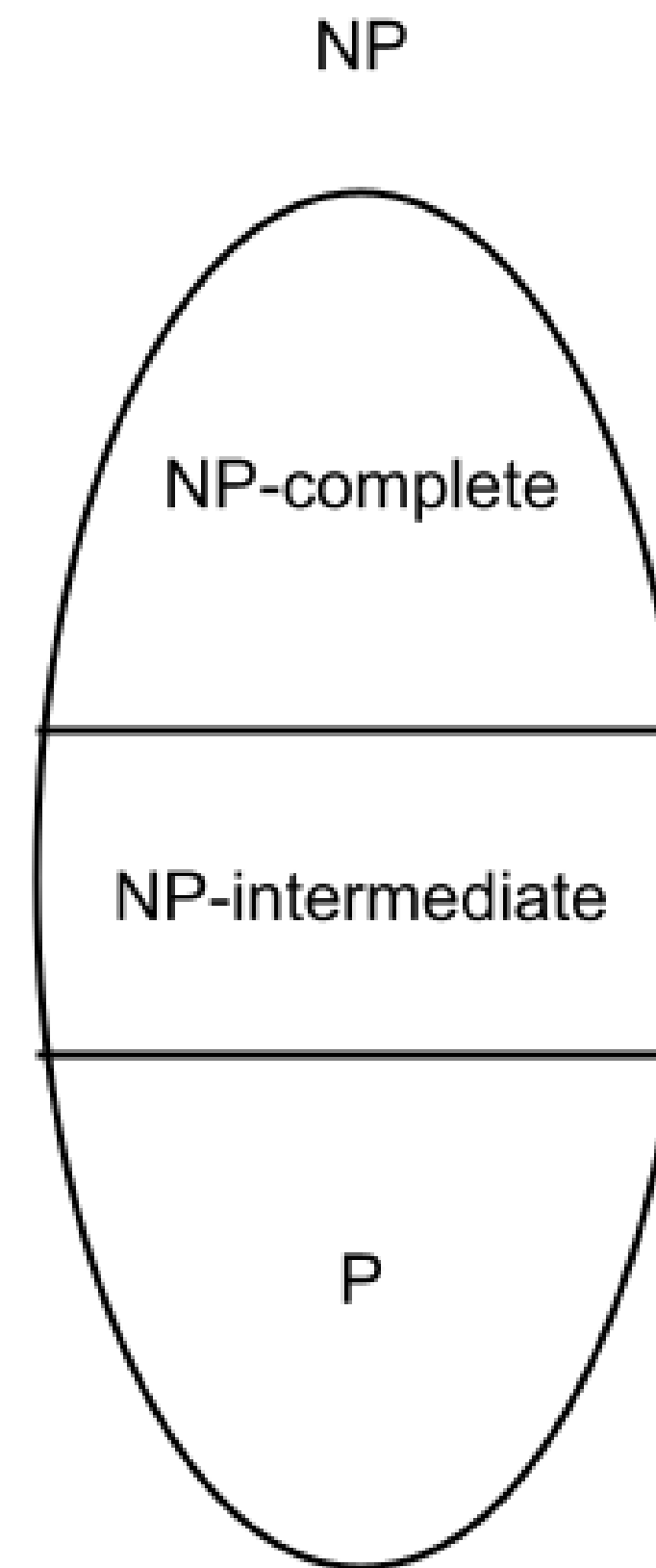
# Classes of Problems

- In 1971, the question "**Is  $P = NP$ ?**" was posed. Since then, it has remained an open question for theorists.
- The most accepted hypothesis is that  **$P \neq NP$** .
- The reason is that if a polynomial solution existed for an NP-complete problem, then all NP problems would also have a polynomial time solution, then  **$P = NP$** .
- If we want to tackle an NP-complete problem, it is better to seek **alternatives** such as simplifications, approximations, etc.

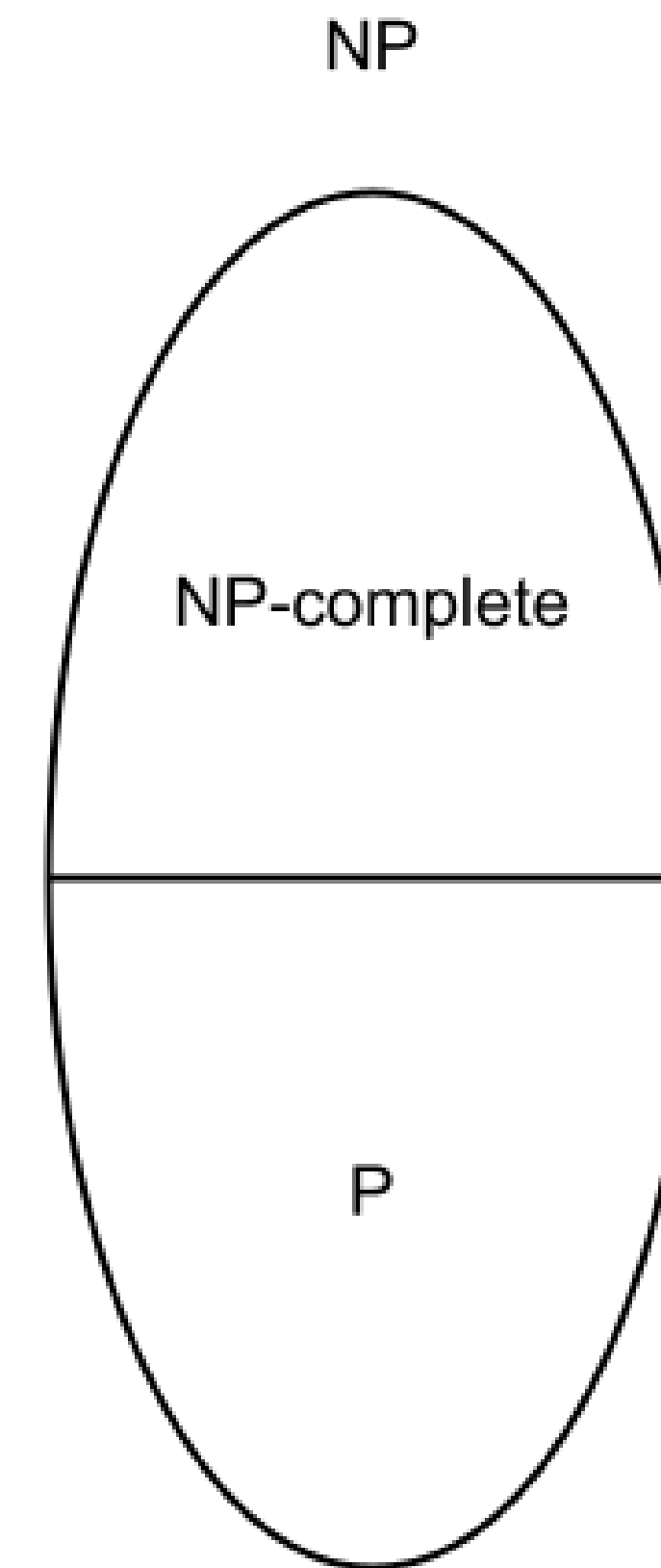


# Classes of Problems

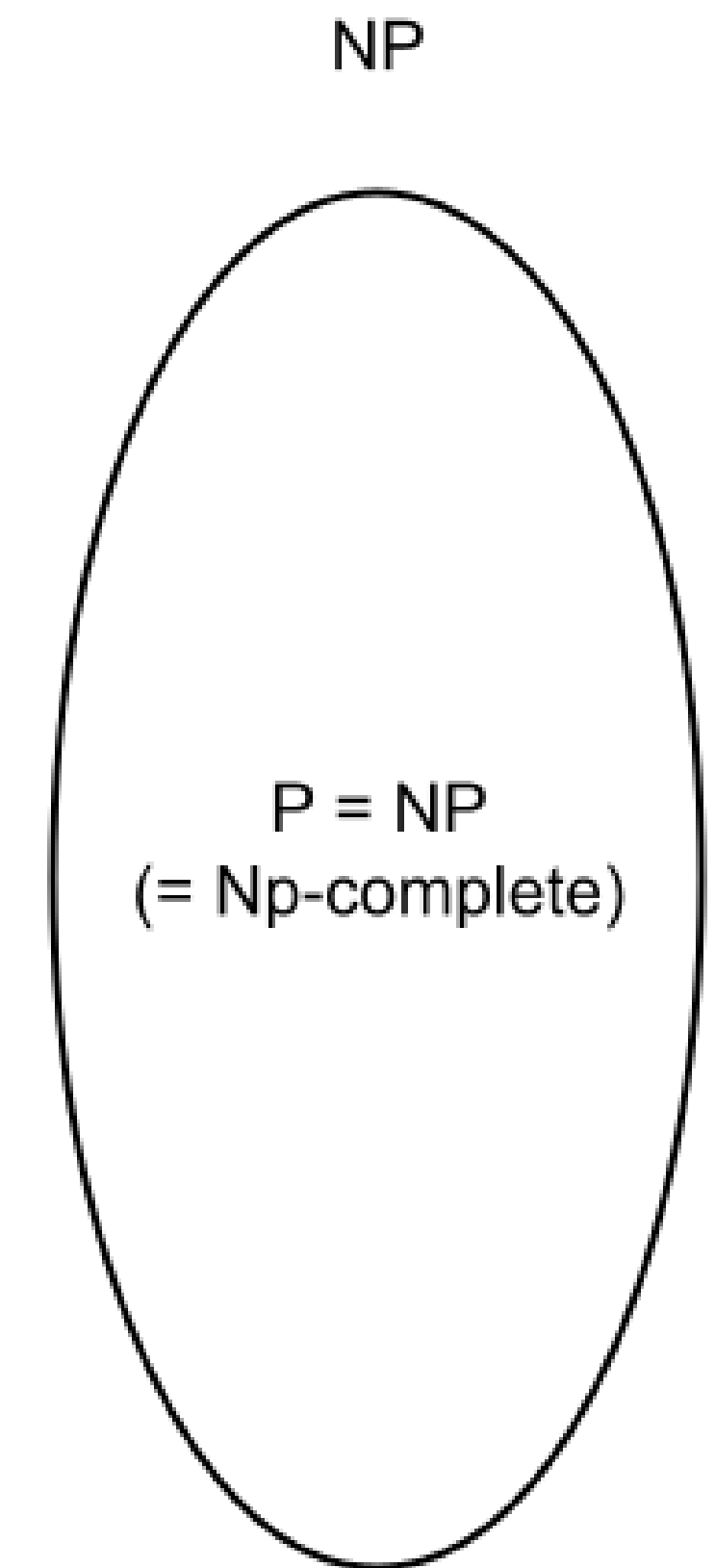
- Three possible alternatives of NP problems.



(a)



(b)



(c)

# Classes of Problems

- It is unknown if there are better algorithms to solve an NP-complete problem of arbitrary size, therefore, some of the following approaches might be used:
  - **Approximation:** An algorithm that **quickly finds a solution** that may not be optimal but falls within a certain range of error. In some cases, finding **a good approximation** is sufficient to solve the problem, but not all NP-complete problems have good approximation algorithms.
  - **Heuristics and Metaheuristics:** An algorithm that performs **reasonably well** in many cases. They are generally fast, but there is no measure of the quality of the answer.
  - **Genetic Algorithms:** Algorithms that improve possible solutions until finding one that is possibly close to the optimum. There is also no guarantee of the quality of the answer.

# Lecture Overview

- Optimisation methods
- Asymptotic complexity
- Classes of problems
- **Search space**
- Memoryless and memory-based metaheuristics
- Population-based methods

# Search Space

- **Feasible solution:** is in the feasible region of the problem.
- For instance, in linear programming, an optimisation problem can be represented as:

$$\max f(x,y) = 3X + 8Y$$

$$\text{subject to } 2X + 4Y \leq 1600 \quad (\text{C1})$$

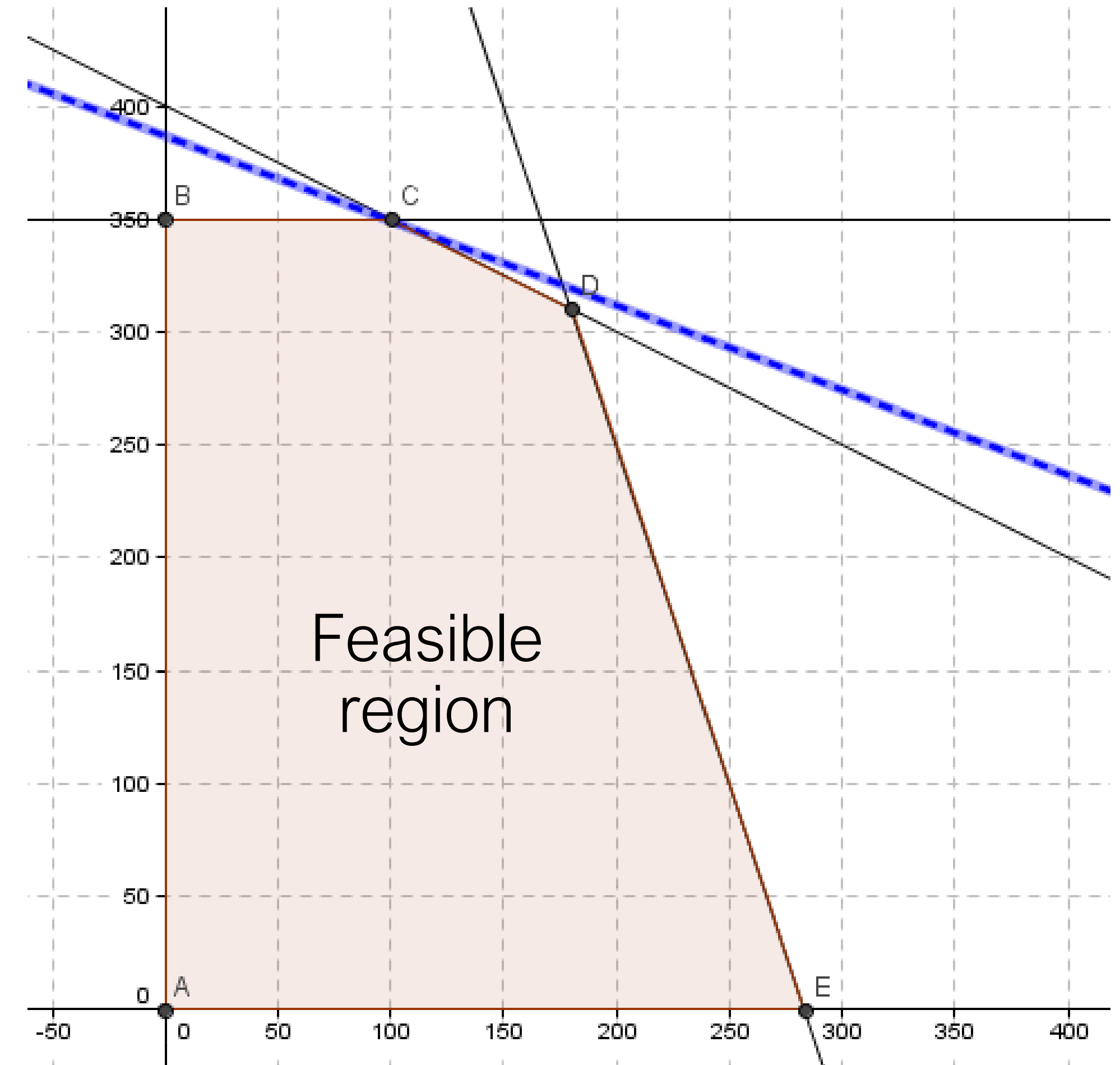
$$6X + 2Y \leq 1700 \quad (\text{C2})$$

$$Y \leq 350 \quad (\text{C3})$$

$$X \geq 0$$

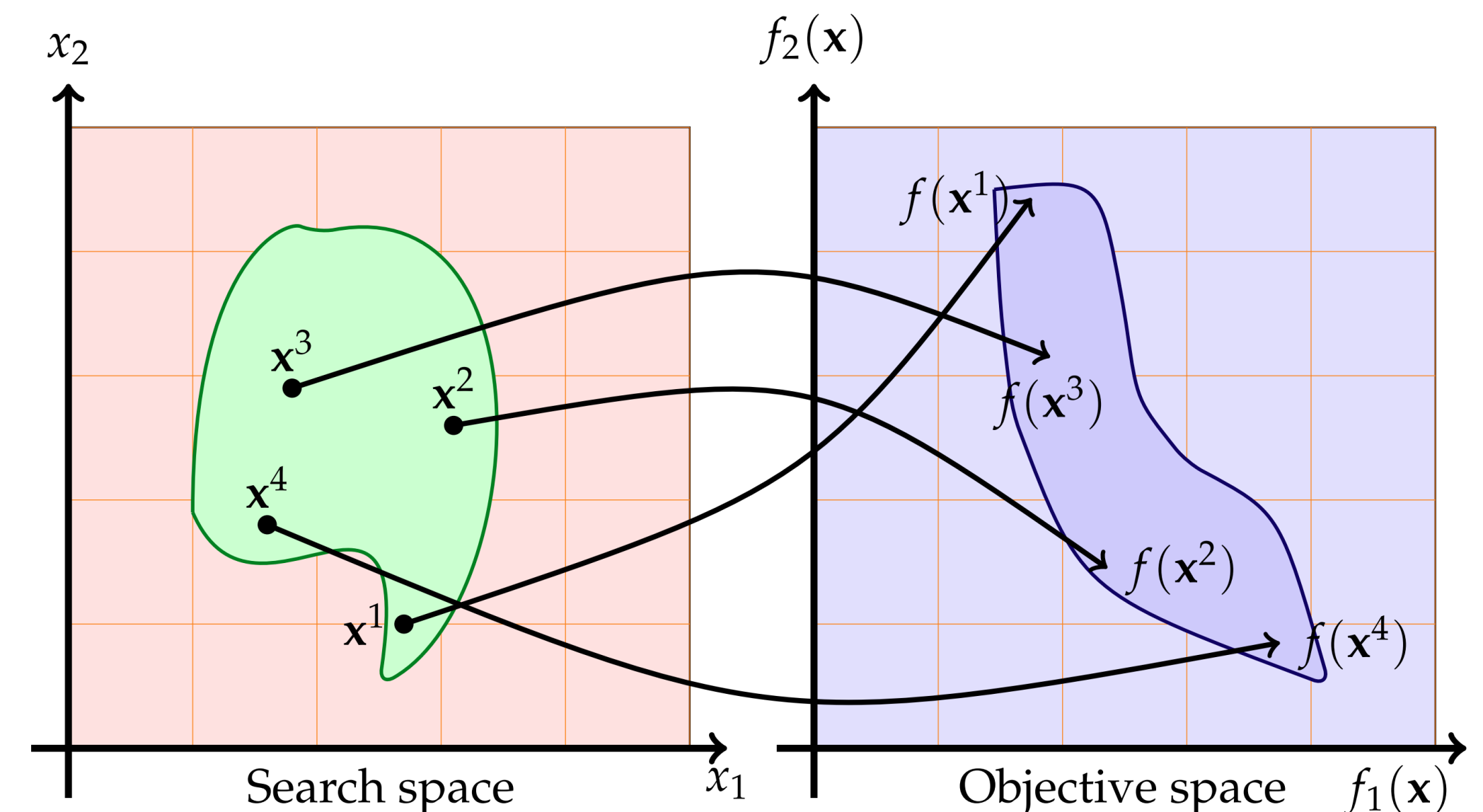
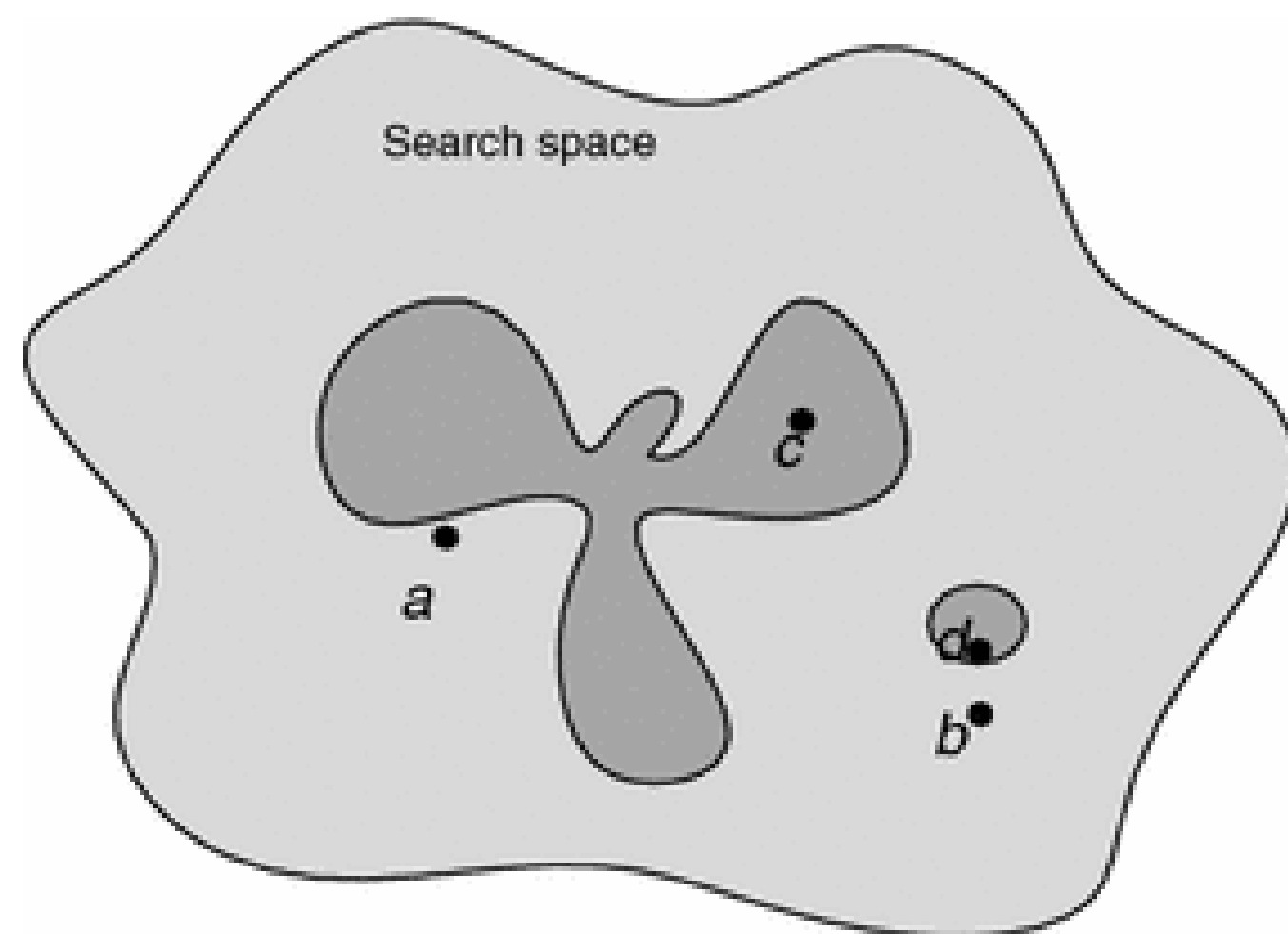
$$Y \geq 0$$

- Optimal solution in  $c = (100, 350) \rightarrow f(c) = 3100$



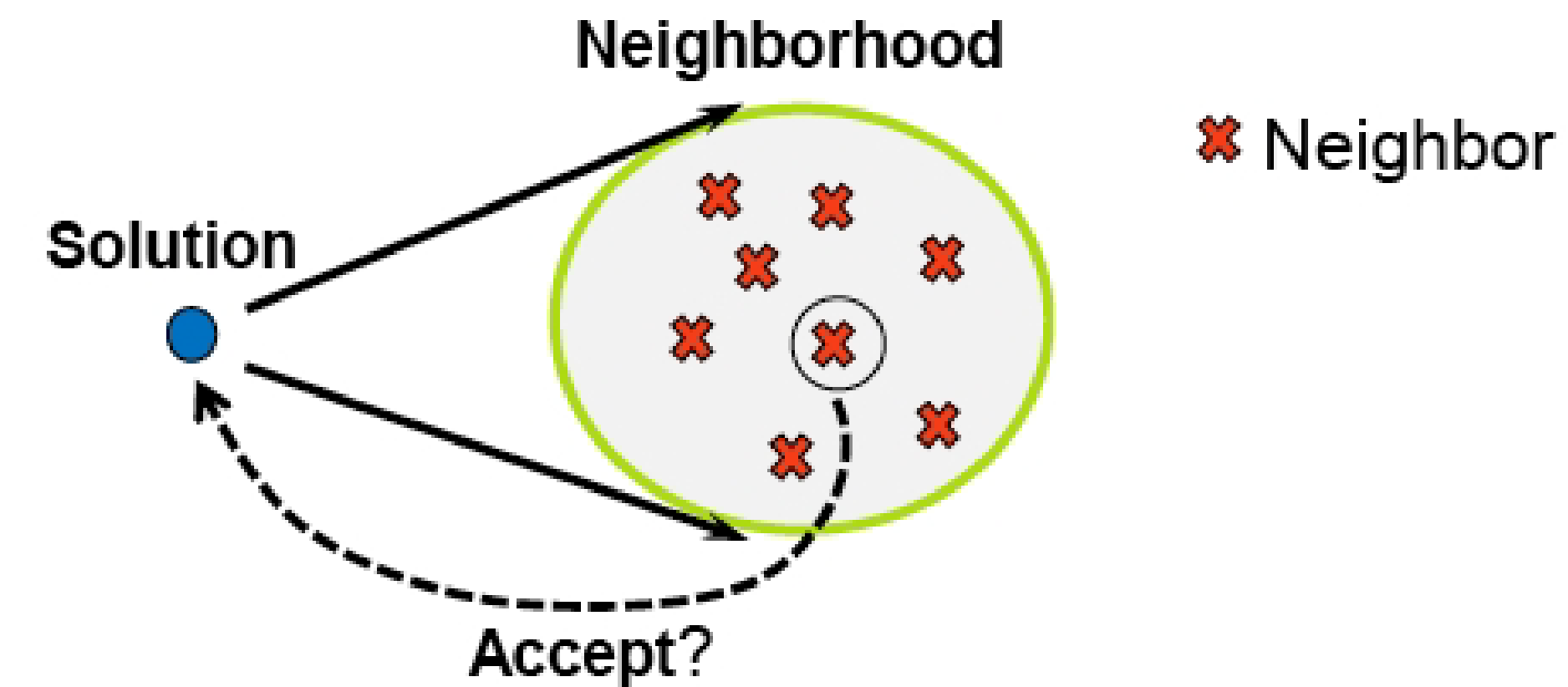
# Search Space

- Solution space  $S$  and objective function  $f$ .
- The optimisation problem  $O(S, f)$  is solved by determining an optimal solution, i.e., a feasible solution  $x_0 \in S / f(x) \leq f(x_0) \forall x \in S$ .
- Constraints of the problem reduce the universe of solutions  $U$ , then  $X \subseteq U$ , also called feasible region.



# Search Space

- Neighbourhood search procedures:
  - Transformations or **movements** from the current solution.
  - Generate an **initial solution**.
  - **Iteratively** modify it until a stopping criterion.
  - Solutions are **evaluated** while traversing.
- Possible movements create a **neighbourhood**.
- **Feasible movements** are those that provide a feasible solution.



# Search Space

- Constraints:
  - Can be **strong** (must be satisfied) or **weak** (recommended to be satisfied).
  - **Example:** In course scheduling, a strong constraint is that classes should not overlap, while a weak constraint is that there should be no classes after 4pm.
- Restricted exploration of the feasible region within the search space:
  - **Advantages:** Infeasible solutions are not evaluated. The algorithms ensure obtaining a feasible solution.
  - **Disadvantages:** The search can be inefficient if restricted only to the feasible region. Optimal solutions may be located near the boundary and difficult to reach.
- Complete exploration of the solution space:
  - **Advantages:** The exploration of the search space is more effective.
  - **Disadvantages:** Time is spent evaluating infeasible solutions. There is a possibility of returning an infeasible solution as the final output of the algorithm.



# Search Space

- Three strategies for **restricted exploration** of the feasible region within the search space:
  - **Rejection strategies:** Any infeasible solution generated during the search is directly ignored.
  - **Repair strategies:** A repair operator is applied to each infeasible solution generated to transform it into a feasible solution. This strategy is often based on heuristics.
  - **Preservation strategies:** Both the representation scheme and the operators are specifically designed for the problem in a way that ensures the feasibility of generated solutions. It requires more design effort and are problem-specific.



# Search Space

- Complete exploration of the solution space:
- The most common scheme for complete exploration of the solution space is penalty-based strategies:
  - A **penalty function** is added to the original unconstrained objective function:

$$\text{Min } f'(x) = f(x) + w \cdot P(x)$$

- where  $P(x)$  is a penalty function and  $w$  is a weighting coefficient (intensify/diversify).
- $P(x)$  takes a value of 0 when the solution  $x$  is feasible. Otherwise, the greater the degree of constraint violation, the larger the value of  $P$ .

# Lecture Overview

- Optimisation methods
- Asymptotic complexity
- Classes of problems
- Search space
- **Memoryless and memory-based metaheuristics**
- Population-based methods

# Metaheuristics

- Metaheuristics provide **strategies** for solving a problem by **conducting a search** over the space of possible solutions.
- The solution representation must include all the necessary information for their **identification and evaluation**.
- A search over a space involves generating a **sequence of points in the space**, where each point is obtained from the previous one through a series of transformations or movements.
- The goal of search-based metaheuristics is to provide guidelines for **obtaining paths** that yield high-quality solutions while also ensuring adequate efficiency.

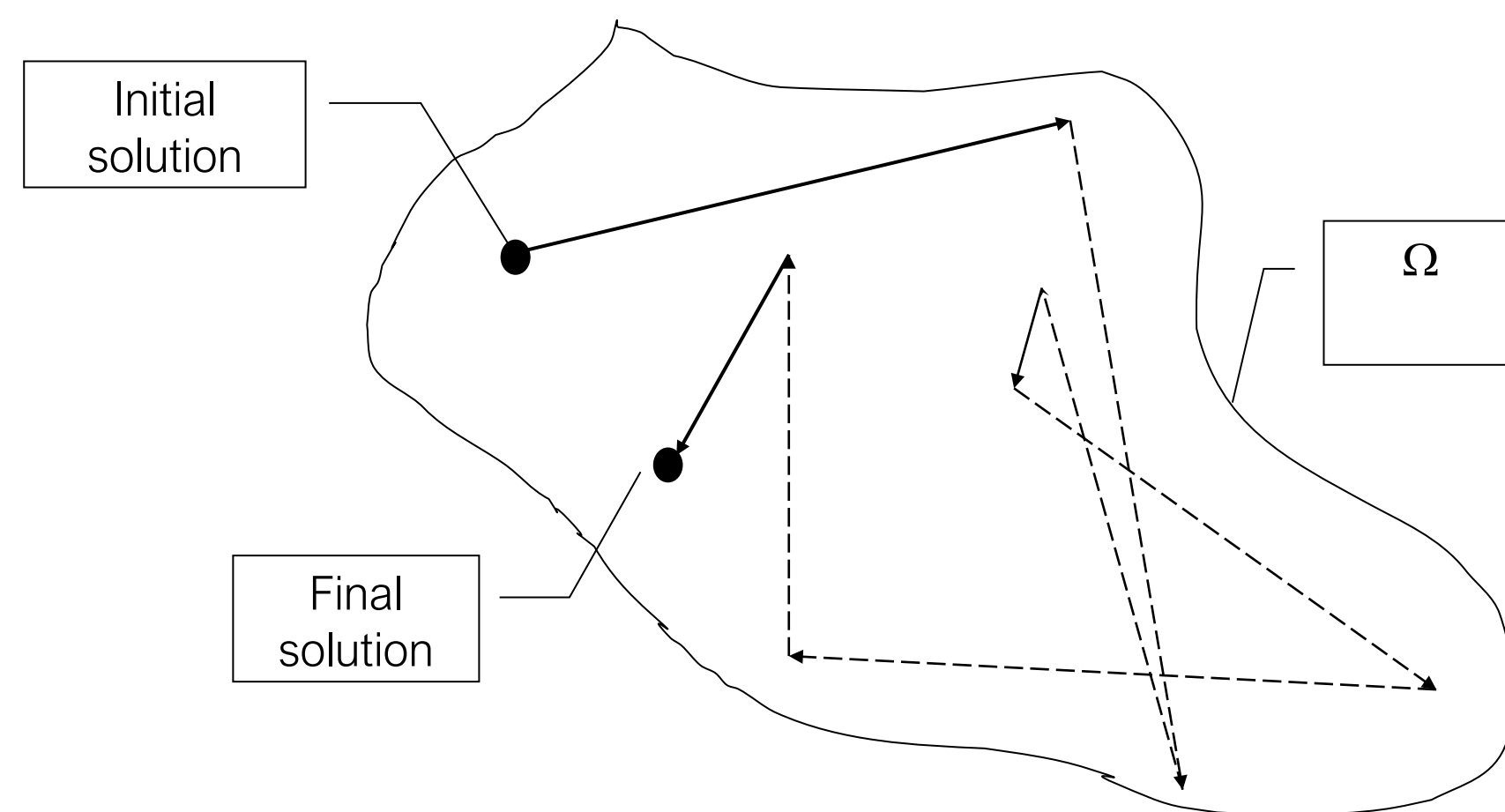
# Metaheuristics

- **Memoryless metaheuristics:** do not use or maintain any explicit memory of past search information. They **rely solely on the current solution and its neighborhood** to make decisions about the next search move. These metaheuristics typically focus on exploration by using randomized or stochastic search techniques.
- **Memory-based metaheuristics:** are algorithms that **use past information or historical data to guide the search process**. They remember and store certain aspects of the search, such as the best solutions found so far or promising regions in the solution space. This memory allows them to make informed decisions and adapt their search strategy based on past experiences.

# Metaheuristics

- **Simulated annealing** is a probabilistic optimization algorithm inspired by the annealing process in metallurgy. It is used to find near-optimal solutions for combinatorial optimization problems.

[Note: Algorithm 1 is minimising]



---

**Algorithm 1** Simulated annealing optimisation method.

---

**Require:** Input( $T_0, \alpha, N, T_f$ )

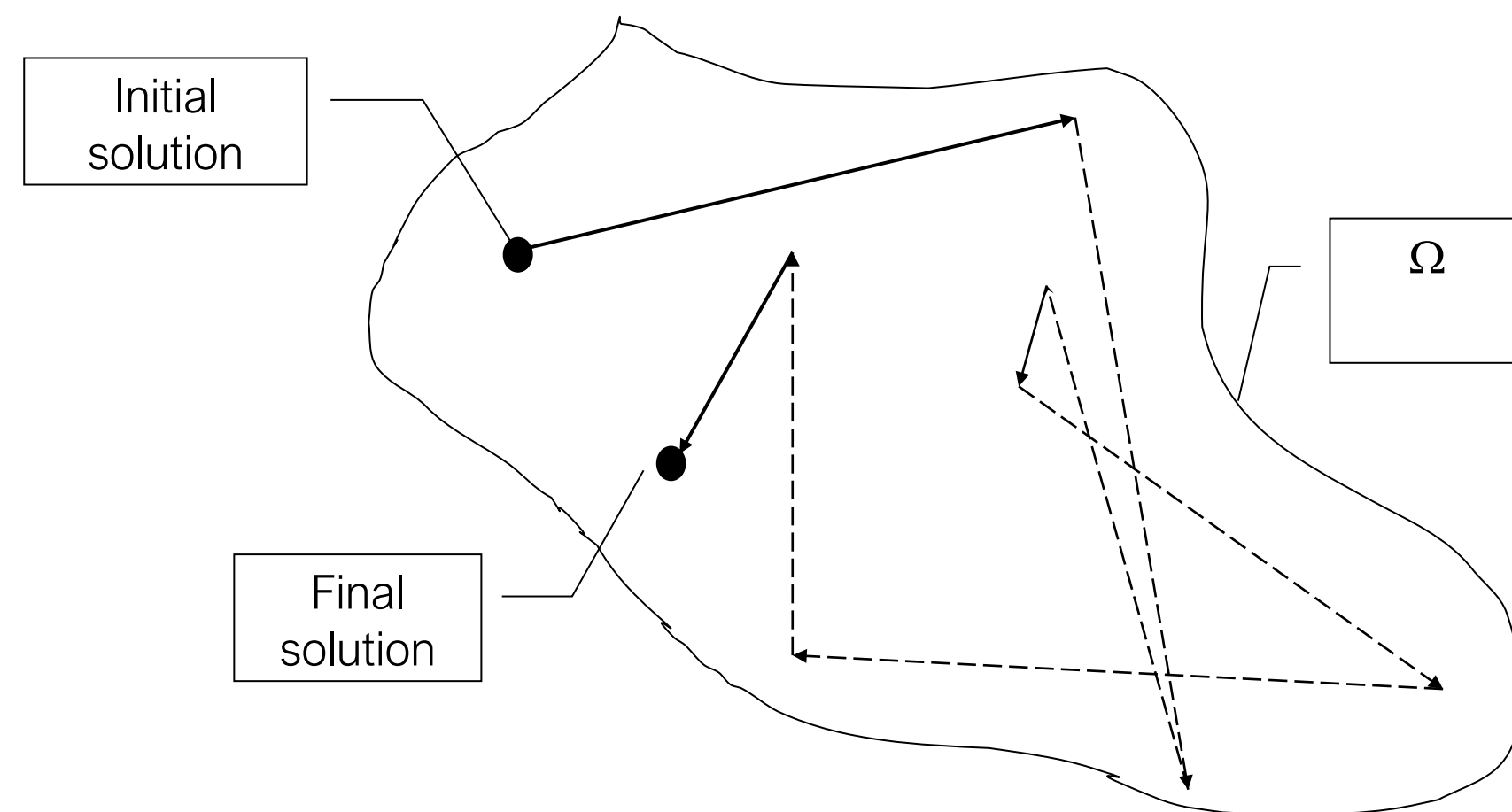
```
1:  $T \leftarrow T_0$ 
2:  $S_{act} \leftarrow$  generate initial solution
3: while  $T \geq T_f$  do
4:   for cont  $\leftarrow 1$  TO  $N(T)$  do
5:      $S_{cond} \leftarrow$  Neighbour solution [from ( $S_{act}$ )]
6:      $\delta \leftarrow f(S_{cond}) - f(S_{act})$ 
7:     if rand(0,1)  $< e^{-\delta/T}$  or  $\delta < 0$  then
8:        $S_{act} \leftarrow S_{cond}$ 
9:     end if
10:  end for
11:   $T \leftarrow \alpha(T)$ 
12: end while
13: return Best  $S_{act}$  visited
```

---

# Metaheuristics

- **Tabu search** is an algorithm used for solving optimization problems. It is based on the concept of maintaining a tabu list, which keeps track of recently visited solutions to prevent cycling and encourage exploration.

[Note: Algorithm 2 is minimising]



---

**Algorithm 2** Tabu search optimisation method.

---

```
1:  $s_0 \leftarrow$  generate initial solution
2:  $s_{best} \leftarrow s_0$ 
3:  $\text{tabuList} \leftarrow \{s_0\}$ 
4: repeat
5:    $\{s_1, s_2, \dots, s_n\} \leftarrow$  generate neighbourhood from  $(s_0)$ 
6:    $s_{candidate} \leftarrow s_1$ 
7:   for  $i \leftarrow 2$  TO  $n$  do
8:      $\delta \leftarrow f(s_i) - f(s_{candidate})$ 
9:     if  $s_i$  is not in  $\text{tabuList}$  and  $\delta < 0$  then
10:       $s_{candidate} \leftarrow s_i$ 
11:     end if
12:   end for
13:    $s_{best} \leftarrow s_{candidate}$ 
14:   Add  $s_{candidate}$  to  $\text{tabuList}$ 
15: until a termination criterion is satisfied
16: return  $s_{best}$ 
```

---



# Metaheuristics



- (Additional) Example: ROADEF Challenge 2005 – Car Sequencing Problem



- Assign a production day to each ordered vehicle and schedule the order of cars to be put on the line for each production day.
- Focus on paint (hard constraint) and assembly (a ratio soft constraint)
- <https://www.roadef.org/challenge/2005/en/sujet.php>

# Lecture Overview

- Optimisation methods
- Asymptotic complexity
- Classes of problems
- Search space
- Metaheuristics with and without memory
- **Population-based methods**



# Population-based Methods

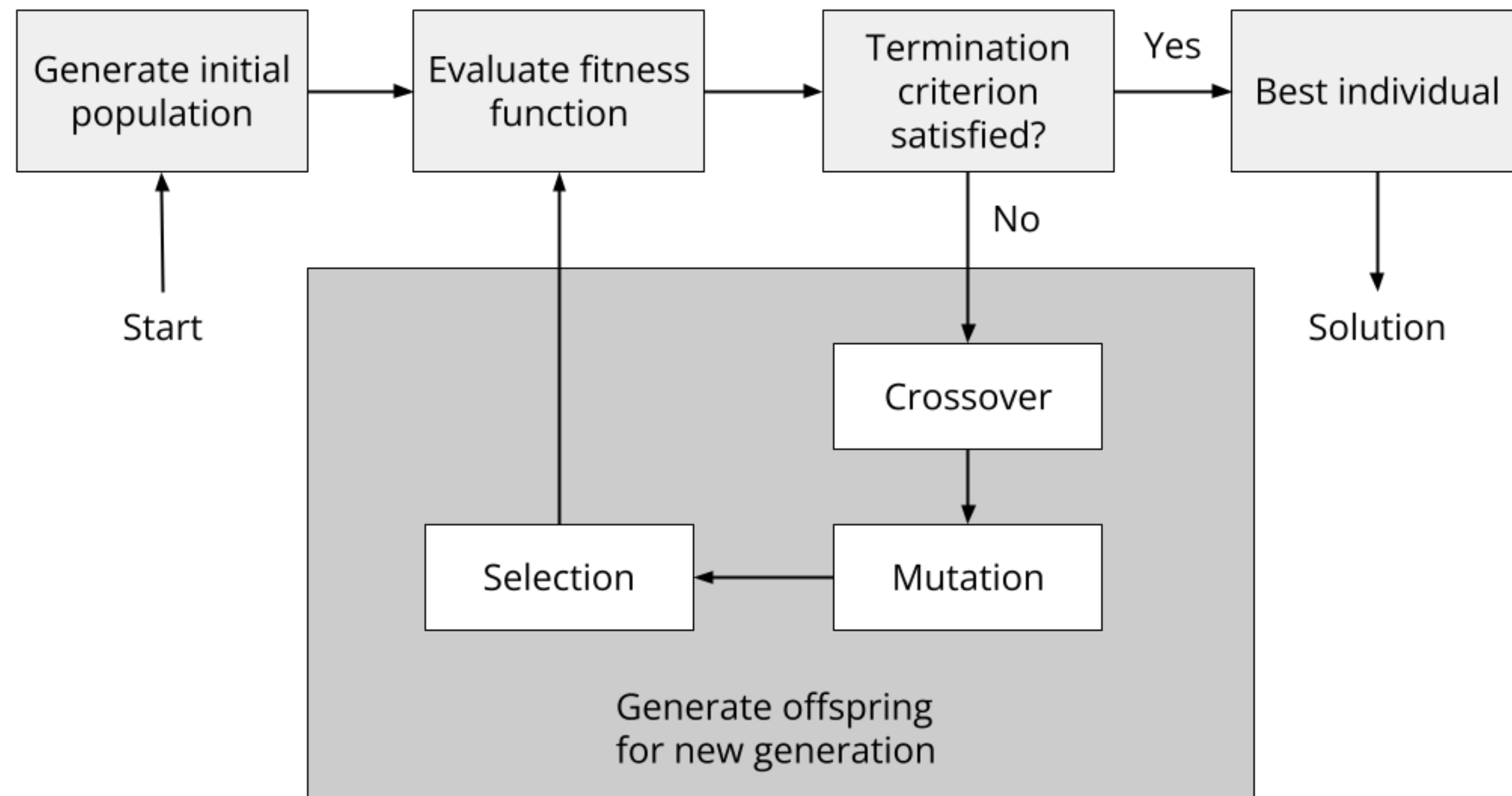
- Operate on a **population of candidate solutions** rather than a single solution. The population is typically initialized randomly or using heuristic techniques.
- Multiple solutions are evaluated simultaneously, allowing for the exploration of the search space more **efficiently**.
  - **Advantages:** ability to simultaneously explore multiple regions of the search space, promoting diversity and preventing premature convergence to suboptimal solutions.
  - **Disadvantages:** it may require careful parameter tuning and can be computationally demanding due to the population size and iterative nature of the algorithms.
- Examples of bio-inspired population-based methods include ant colony optimization, black hole algorithm, particle swarm optimization, and **genetic algorithms**.

# Population-based Methods

- **Genetic algorithms:** Based on Darwin's evolution theory.
- “One general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die.” Charles Darwin.
- **Stochastic search** technique based on the mechanisms of natural selection and natural genetics.
- Use **analogies** of natural selection to develop better solutions.
- Widely used in problems of nonlinear and high-dimensional optimization.

# Genetic Algorithms

- GA model the process of evolution as a **sequence** of changes in genes, with solutions analogous to chromosomes.
- The search space is explored by applying **transformations** to candidate solutions, just as observed in living organisms: crossover, mutation, and selection.



# Genetic Algorithms

- Terminology used in genetic algorithms:

Term	Meaning
Chromosome (string, individual)	Solution
Genes (bits)	Part of the solution.
Locus	Position of the gene
Alleles	Value of the gene
Phenotype	Decoded solution (external appearance)
Genotype	Encoded solution (internal structure)

# Genetic Algorithms

- Three main operators: crossover, mutation, and selection.
- **Genetic: crossover and mutation.** They emulate the process of gene inheritance to create new solutions. **Evolution: selection.** It emulates Darwinian evolution to create a population from one generation to another.
- **Crossover:** operates on 2 chromosomes, generating **two offspring** by combining characteristics. The performance of the algorithm highly depends on this operation.
  - Crossover rate ( $p_c$ ): the number of offspring produced each generation divided by the population size.

# Genetic Algorithms

- Three main operators: crossover, mutation, and selection.
- **Genetic: crossover and mutation.** They emulate the process of gene inheritance to create new solutions. **Evolution: selection.** It emulates Darwinian evolution to create a population from one generation to another.
- **Mutation:** operates on **1 chromosome**, producing random spontaneous changes in a gene, contributing to exploration of the search space.
  - Mutation rate ( $p_m$ ): percentage of the total number of genes in the population to mutate. It controls the rate at which new genes are introduced into the population.



# Genetic Algorithms

- Three main operators: crossover, mutation, and selection.
- **Genetic: crossover and mutation.** They emulate the process of gene inheritance to create new solutions. **Evolution: selection.** It emulates Darwinian evolution to create a population from one generation to another.
- **Selection:** the **selective pressure** is critical for the algorithm.
  - High pressure: the search may end prematurely (**intensification**).
  - Low pressure: progress is slower than necessary (**diversification**).
- The ideal approach is to maintain low pressure at the beginning for broad exploration, and high pressure towards the end to exploit more promising areas.

# Genetic Algorithms

- General structure:
- Chromosome commonly represented as strings of bits or binary representation.
- Parameters include population size and probability of applying the genetic operators

---

**Algorithm 3** Genetic algorithm optimisation method.

---

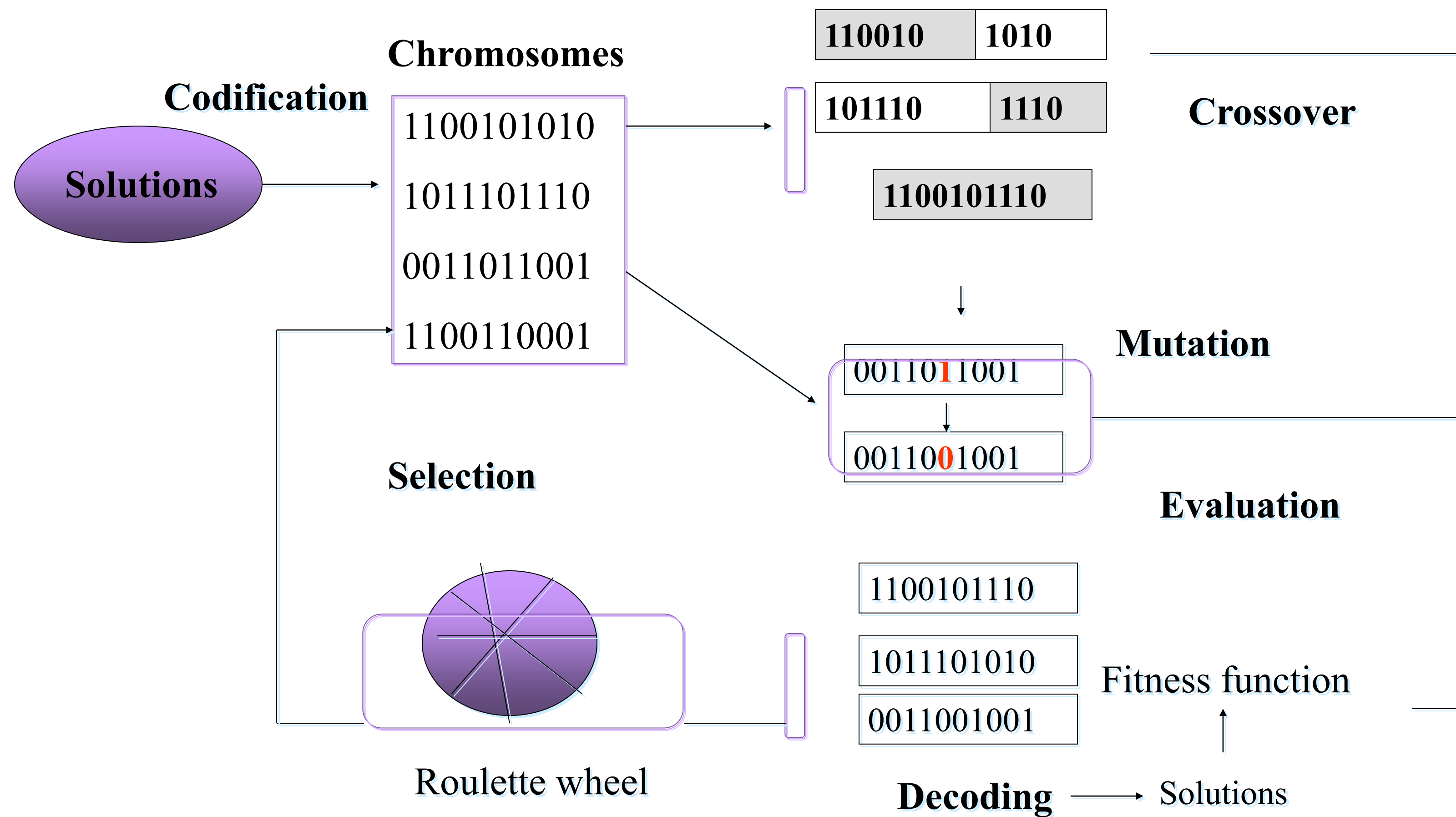
```
1:  $t \leftarrow 0$ 
2: Initialise  $P(t)$  ► initial population
3: Evaluate  $P(t)$ 
4: repeat
5:   Generate offspring  $C(t)$  from  $P(t)$  ► using crossover and mutation
6:   Evaluate  $C(t)$ 
7:   Select  $P(t + 1)$  from  $P(t) \cup C(t)$ 
8:    $t \leftarrow t + 1$ 
9: until a termination criterion is satisfied
10: return Best individual found from  $P$ 
```

---



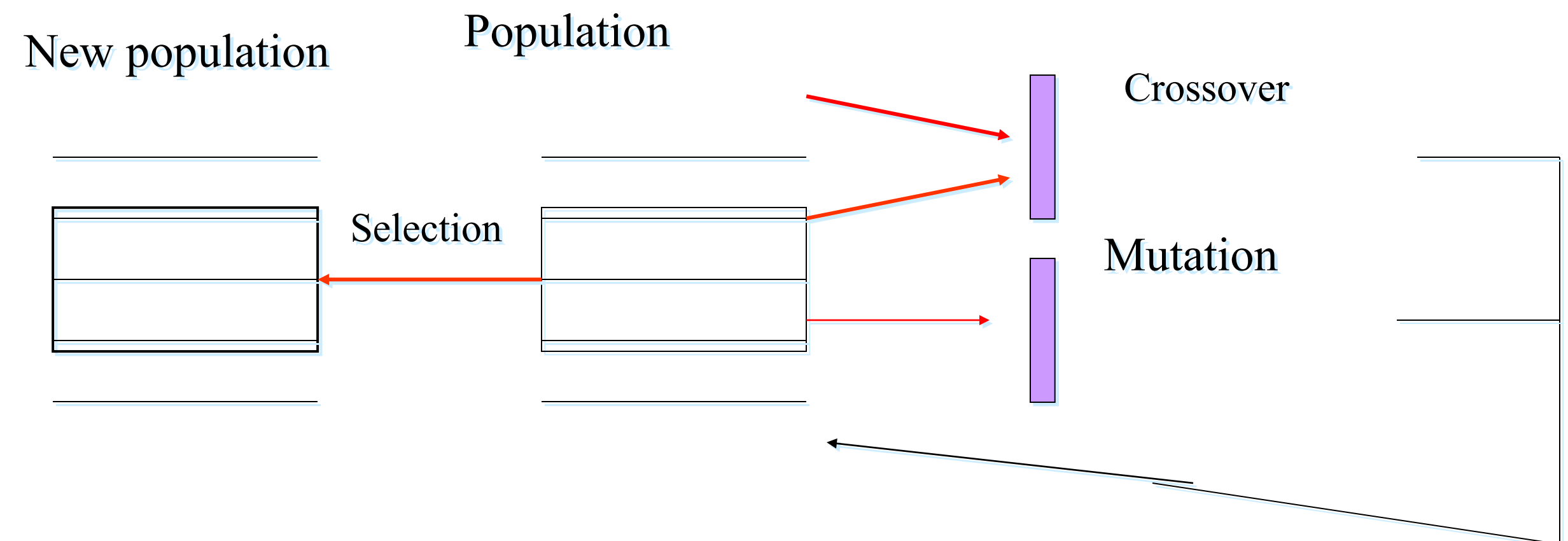
# Genetic Algorithms

- General structure example:



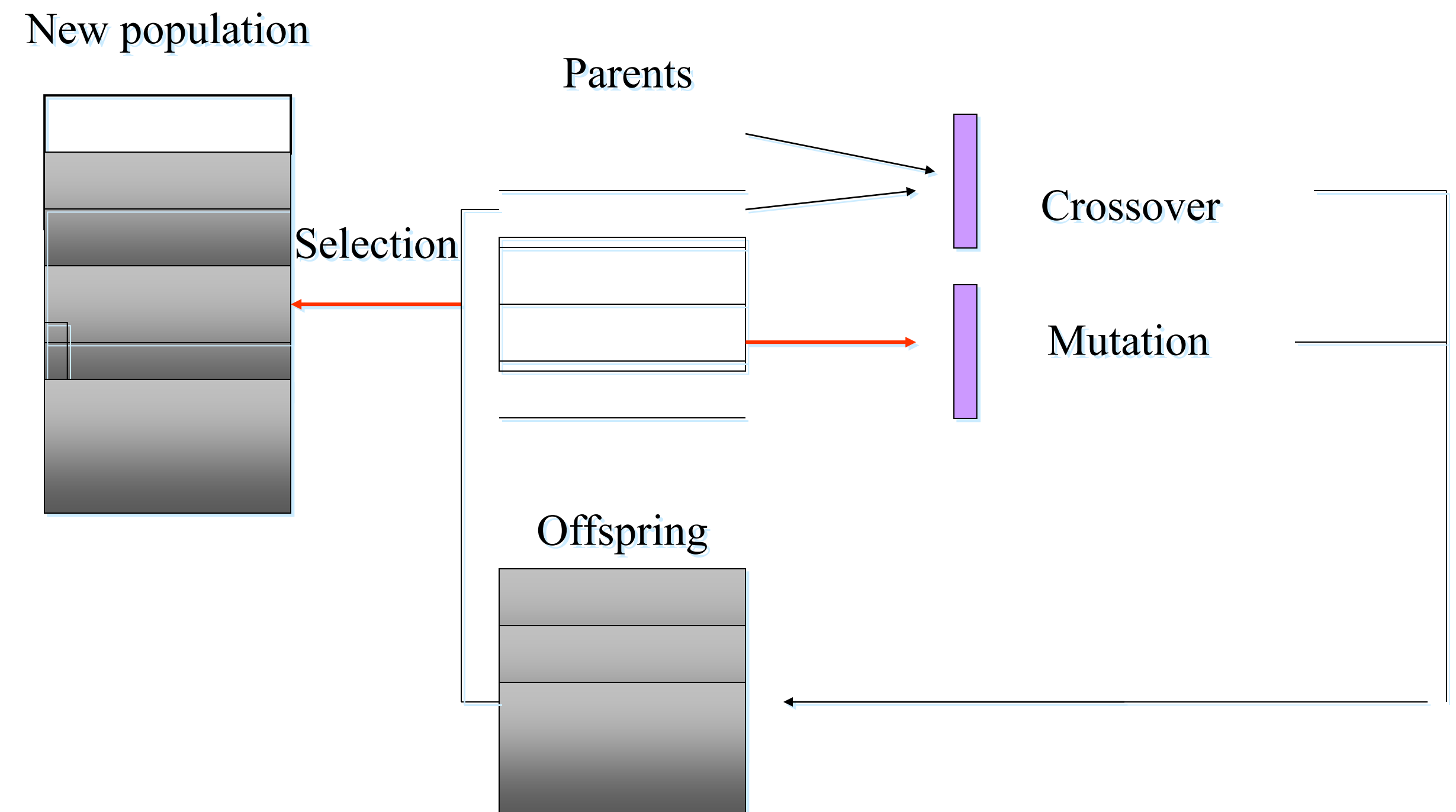
# Genetic Algorithms

- New generation size: Uniform
- New generation size = Same as previous generation
- All offspring and some parents. Originally, all offspring replaced all parents



# Genetic Algorithms

- New generation size: Expanded
- New generation size =  
Previous generation size +  
number of offspring
- All offspring and parents.

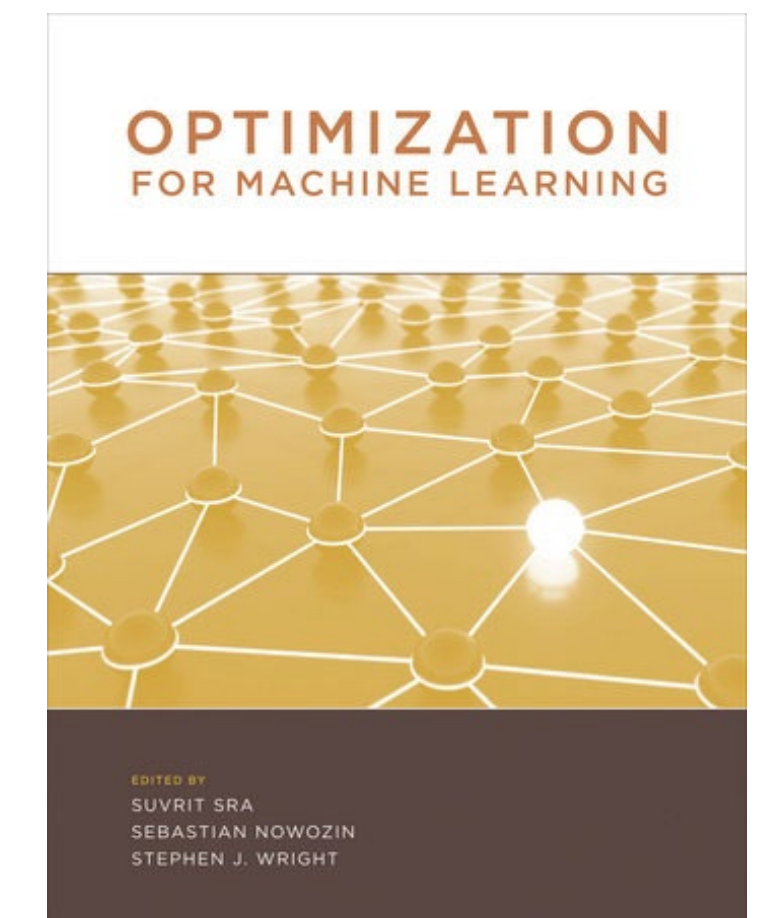
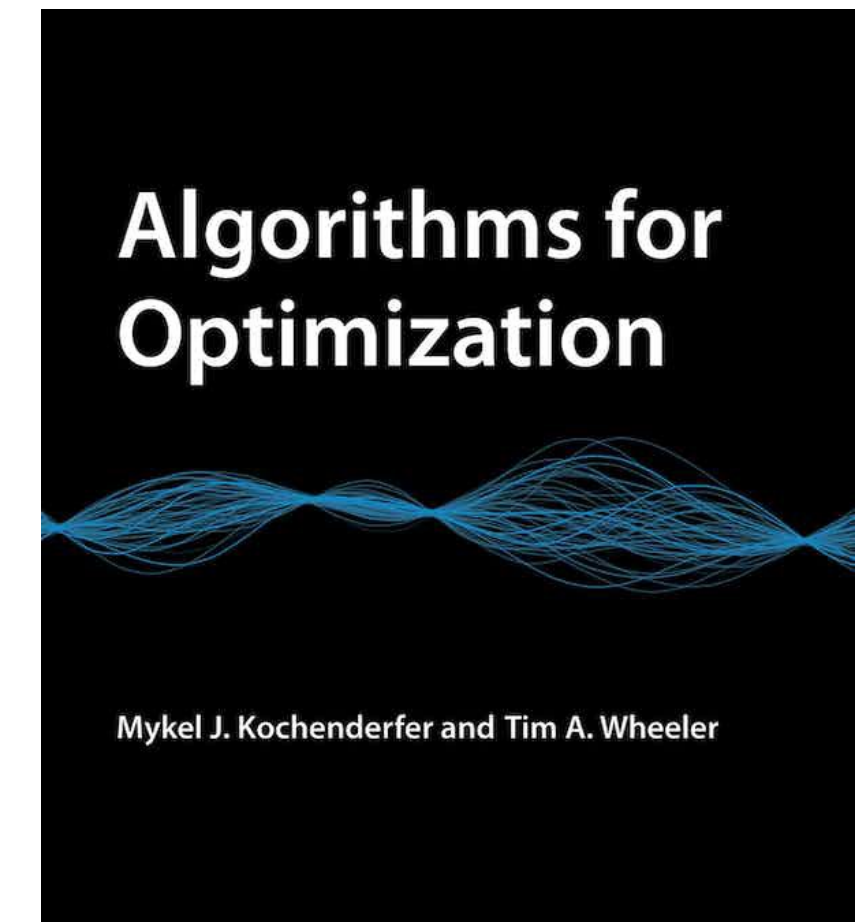
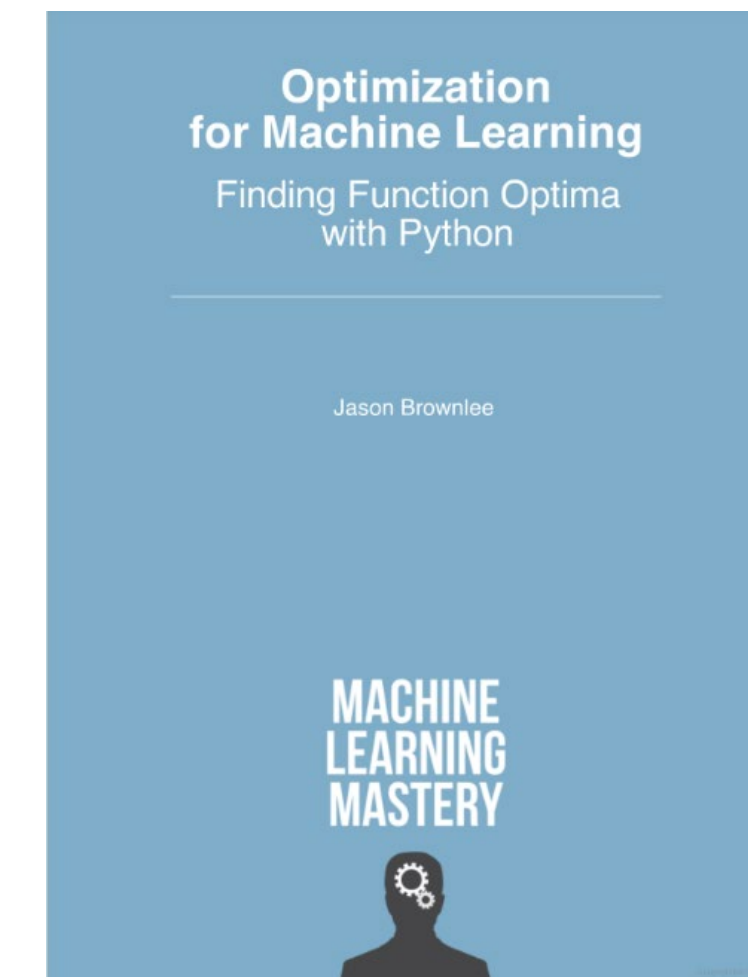


# Genetic algorithms

- **Stochastic sampling:** prevent super chromosomes. For instance, roulette wheel.
- **Deterministic sampling:** sort chromosomes according to their fitness and choose the best ones. Elitist selection.

# References

- Kochenderfer, M. J., & Wheeler, T. A. (2019). Algorithms for optimization. MIT Press.
- Brownlee, J. (2021). Optimization for machine learning. Machine Learning Mastery.
- Sra, S., Nowozin, S., & Wright, S. J. (Eds.). (2012). Optimization for machine learning. Mit Press.






# Feedback

- In case you want to provide anonymous feedback on these lectures, please visit:
- <https://forms.gle/KBkN744QuffuAZLF8>

Muchas gracias!



**AI Lecture Feedback**

This is a short form to provide early feedback for lectures

franciscocruzhh@gmail.com [Switch account](#)

Not shared

\* Indicates required question

In case you want a reply, provide your zID. Otherwise your answer is anonymous.

Your answer

how did you participate? \*

☐ In the classroom

☐ Watch the class from automatic recording

If you have any comments, feedback, or question about the lectures, this is the place. \*

Your answer

Submit Clear form