

---

---

# Minimum Weight Vertex Cover Problem

Artificial Intelligence Project

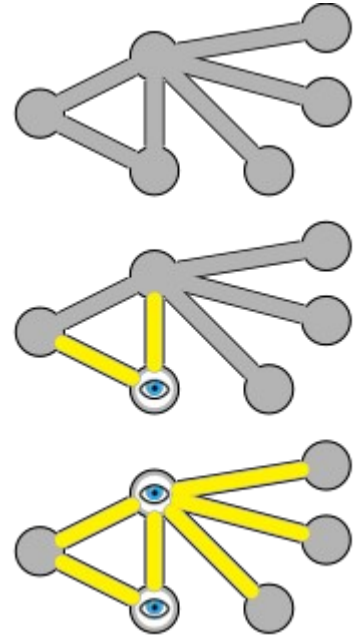
# The problem

Given a problem instance  $(G, \omega)$ , where  $G$  is an undirected graph  $G(V, E)$  and  $\omega : V \rightarrow \mathbb{R}^+$  a function that associates a positive weight value  $\omega(v)$  to each vertex  $v \in V$ , the Minimum Weight Vertex Cover can formally be defined as follows:

$$\text{minimize } \omega(S) = \sum_{v \in S} \omega(v), \quad S \in V$$

such that  $\forall (v_i, v_j) \in E, v_i \in S \vee v_j \in S$ .

Note that the MWVC is a NP-complete problem.

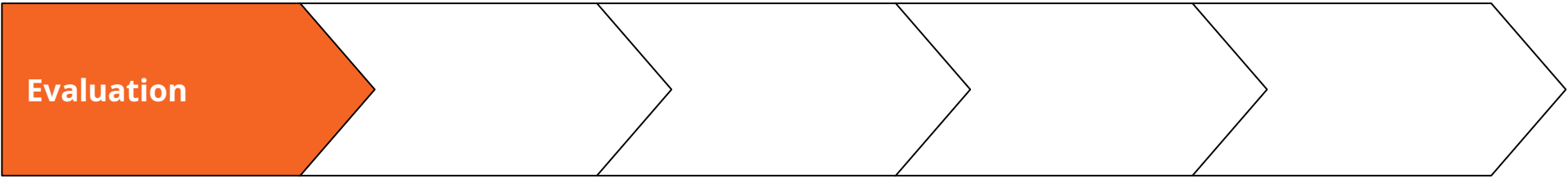




# The solution

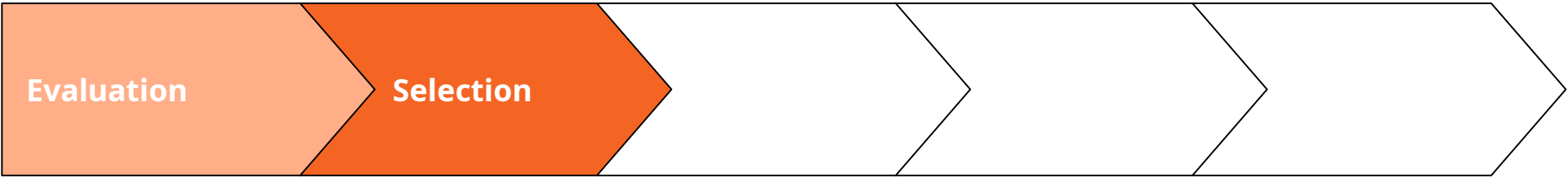
1. Tabu Search (TS);
2. Branch-and-Bound;
3. Genetic Algorithm (GA) with uniform crossover and roulette-wheel selection;
4. **Genetic Algorithm (GA) with one-point crossover and k-tournament selection;**

# Genetic Algorithm



Each individual in the population is evaluated based on a predefined fitness function

# Genetic Algorithm



A subset of individuals are selected from the populations, based on fitness.

In case of the k-tournament selection algorithm, it involves running several "tournaments" among k individuals chosen at random from the population, until the desired amount of population is reached. Each tournament selects the best amongst the k selected individuals

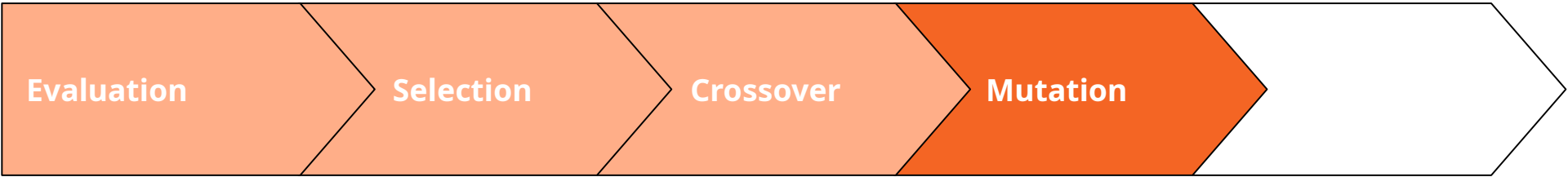
# Genetic Algorithm



Genetic material of the individuals are combined to create offspring

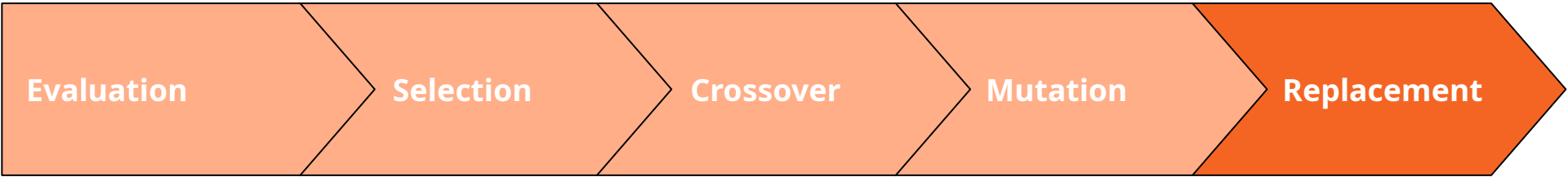
In case of single-point crossover, a random crossover point is selected and the genetic material is exchanged between the parents at that point

# Genetic Algorithm



With a certain probability, random mutations are introduced into the offspring, helping to maintain diversity within the population

# Genetic Algorithm



This newly generated population replaces the old one.

The process repeats until a stopping criteria is satisfied.



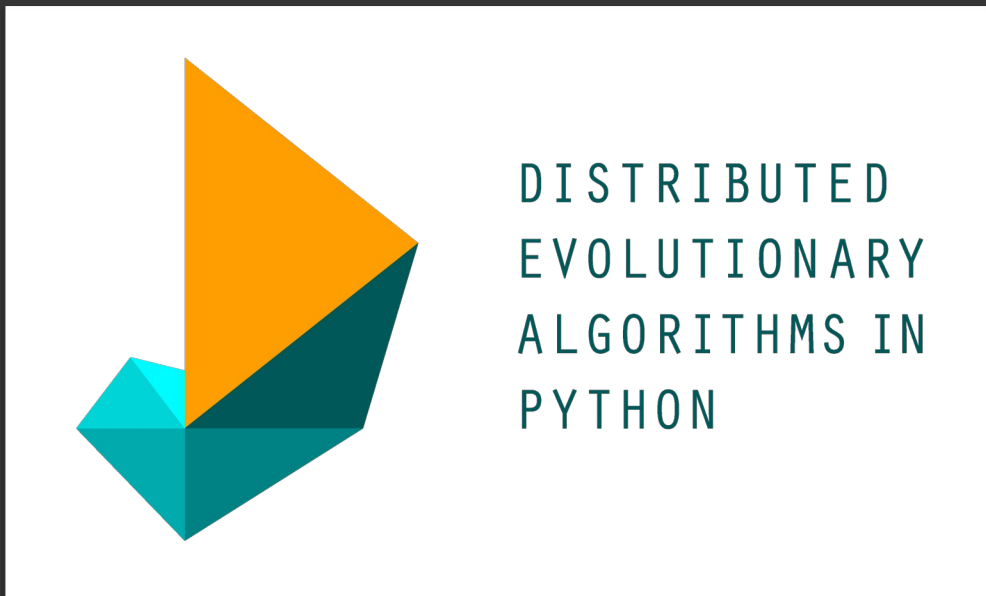
---

# Implementation

- **DEAP Framework**
- **Gene Representation**
- **Fitness Function**
- **Performance Metrics**
- **Parameters**
- **Benchmarking Flow**

# DEAP Framework

DEAP is a Python library that excels at rapid prototyping and testing of ideas



# Gene Representation

Each gene is a binary string where each bit represents the presence of a vertex in the solution

1	1	0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---

# Fitness Function

It calculates the weight of the solution by summing the weights of the vertices present in the solution.

Additionally, it imposes a penalty for each edge that is not covered by the vertices in the solution.

# Performance Metrics

- Fitness Function
- Objective Function
- Number of Evaluations

# Parameters

- Population Size
- Crossover Probability
- Mutation Probability
- K-Tournament Selection Size
- Number of Generations

# Benchmarking Flow

The best parameters are found using an iterative process.

1. **Test instances are loaded**
2. **The algorithm is run**
3. **The results are saved**
4. **The results are analyzed using a Jupyter Notebook**

# Results



# 1st Iteration

The algorithm is run with defaults parameters, to see how it performs.

It turns out that the penalty mechanism isn't effective - a repair function is implemented to transform individuals into valid solutions

## 2nd Iteration

The population size is tuned.

The size of **400** is found to be the best.

# 3rd Iteration

Crossover and Mutation parameters are analyzed.

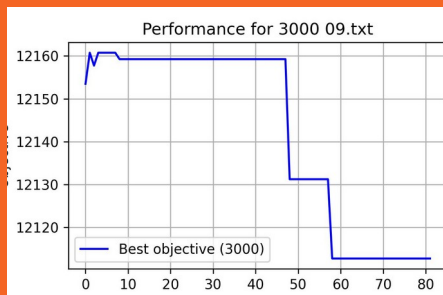
These are studied together since they're tightly coupled.

The best parameters found are **0.4** and **0.25** for crossover and mutation probability, respectively.

# 4th Iteration

Tournament size is modified.

Using  $k=3$  gave the highest results in most the instances, so this was chosen.



## 5th Iteration

To improve efficiency, the number of generations must be tuned.

Analyzing the plots, there's no correlation between optimal number of generation and problem size.

The chosen stopping criteria is to stop the algorithm after a fixed **40 generations of no improvement**.

# Final Results

Problem Class	Test Instance	Average Objective	Average Evaluations
LPI	10000	47784.5	10501.7
MPI	3000	12054.5	12279.1
MPI	2000	6091.0	12021.6
MPI	750	9460.2	16703.0
MPI	500	4768.6	18637.0
SPI	150	1264.0	10272.1
SPI	120	1038.2	10222.2
SPI	60	861.8	10430.5

**Thanks**