EVEN

MORE PIZZA

AI Assignment 1 Checkpoint - Optimization Methods/Meta-Heuristics

Guilherme Santos, up202105090
Lara Bastos, up202108740
Miguel Barros, up202108678

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

REFERENCES

- Lectures Slides
- Geek for Geeks - Hill Climbing
- Problem Resolution - HashCode 2021
- PYQT5 Documentation

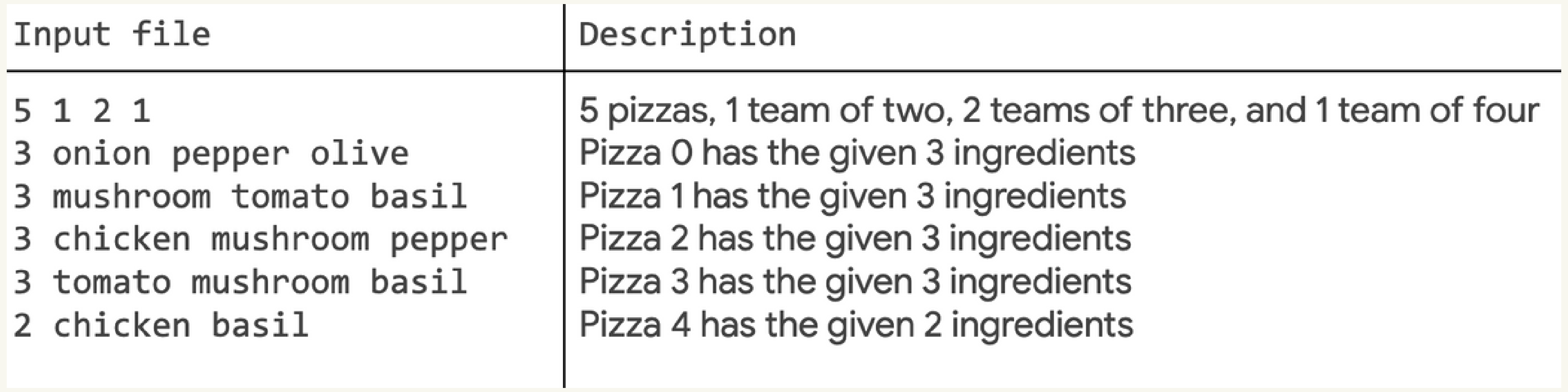
WORK TO BE PERFORMED

PROBLEM:

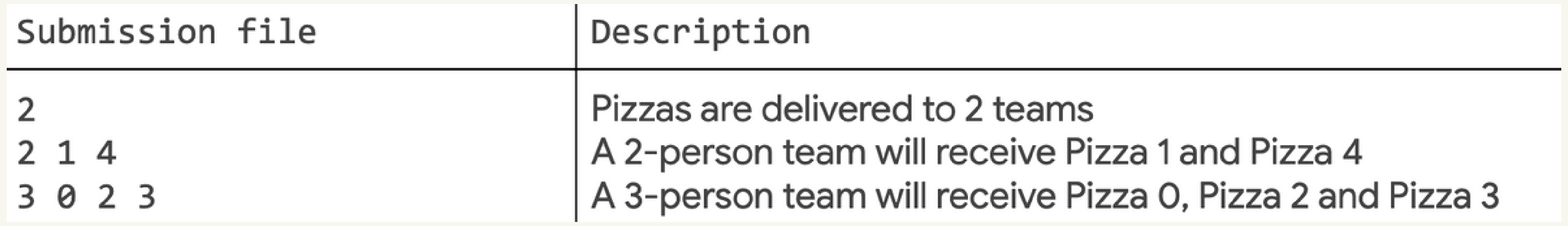
Distributing a given database of available pizzas across different teams according to the following:

- Each team consists of either **2**, **3** or **4 elements**.
- For all N-person teams, either **nobody** or **everybody receives a pizza.**
- **Each pizza** must be part of **at most one order**.
- There are **Tn** or **less deliveries to teams of N**.

EXAMPLE:



Input file	Description
`5 1 2 1`	5 pizzas, 1 team of two, 2 teams of three, and 1 team of four
`3 onion pepper olive`	Pizza 0 has the given 3 ingredients
`3 mushroom tomato basil`	Pizza 1 has the given 3 ingredients
`3 chicken mushroom pepper`	Pizza 2 has the given 3 ingredients
`3 tomato mushroom basil`	Pizza 3 has the given 3 ingredients
`2 chicken basil`	Pizza 4 has the given 2 ingredients



Submission file	Description
`2`	Pizzas are delivered to 2 teams
`2 1 4`	A 2-person team will receive Pizza 1 and Pizza 4
`3 0 2 3`	A 3-person team will receive Pizza 0, Pizza 2 and Pizza 3

GOAL:

Maximazing, per team, the number of different ingredients used in all their pizzas.

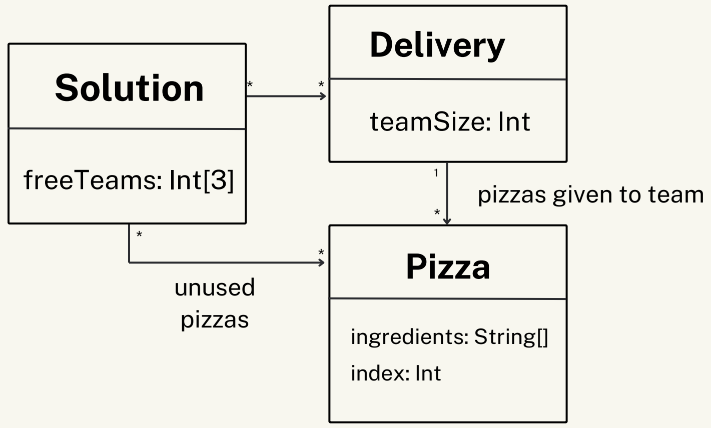
PROJECT:

Implement a system to solve an optimization problem, using different algorithms or meta-heuristics as well as multiple instances and different parameterizations of the problem for comparison.

FORMULATION OF THE PROBLEM AS A SEARCH PROBLEM

SOLUTION REPRESENTATION:

- **Solution Class:**
  - List of Deliveries.
  - List with unused pizzas.
  - Number of free teams.
- **Delivery Class:**
  - Team size.
  - Pizzas given to team.



NEIGHBOURHOOD/ MUTATION :

- Swap 1 pizzas between any 2 deliveries
- Swap 1 pizza between a delivery and an unused pizza.
- Remove a team.
- Add a new team and fill it with unused pizzas.

To select a neighbour, the four mutation functions are performed and the **neighbour with the higher score** is selected.

EVALUATION FUNCTION:

- For each delivery, the score is given by **the square of the total number of different ingredients of all the pizzas in the team**.
- The evaluation function is the **sum of all the scores**.

Evaluation function

$\Sigma i \in i(T)$ = unique_ingredients(i)^2

CROSSOVER FUNCTIONS:

- According to fitness, we choose one parent Solution P1 and select another one randomly P2.
- One or two crossover point are chosen according to how good are each part of P1.
- Swap the pizza selections between the two parents at that point to generate two new offspring solutions.

HARD CONSTRAINTS:

- Each team consists of either **2**, **3** or **4** elements.
- For all N-person teams, either **nobody** or **everybody** receives a pizza.
- Each pizza must be part of at most one order.
- There are Tn or less deliveries to teams of N

IMPLEMENTATION WORK ALREADY CARRIED OUT

LANGUAGE AND ENVIRONMENT

- The chosen language was **Pyhton**.
- **PyQt5** for the interface.
- Four directories:
    - **src:** source code
    - **data:** input files
    - **results:** output files
    - **docs:** support documents

DATA STRUCTURES

- **Pizza:** with a list of ingredients, number of ingredients and index of pizza.
- **Delivery:** with a team size and a the list of pizzas delivered to them.
- **Solution:** list of deliveries, list with unused pizzas and list of free teams

FUNTIONALITIES IMPLEMENTED

- Initial Randomisation Function.
- Evaluation Function.
- User Interface.
- Mutation functions.

Hill Climbing:

- Generate a random initial solution and evaluate
- Generate neighbour solution
- If **neighbour's score > curr_solution's score**, curr_solution = neighbour
- Continue until the score doesn't improve during **n iterations**

Simulated Annealing:

- Generate a random initial solution and evaluate.
- Set initial temperature and iteration count.
- Repeat until convergence:
    - Generate a neighbour solution and evaluate
    - Accept it if it's better or with a probability based on temperature
- Return final solution.