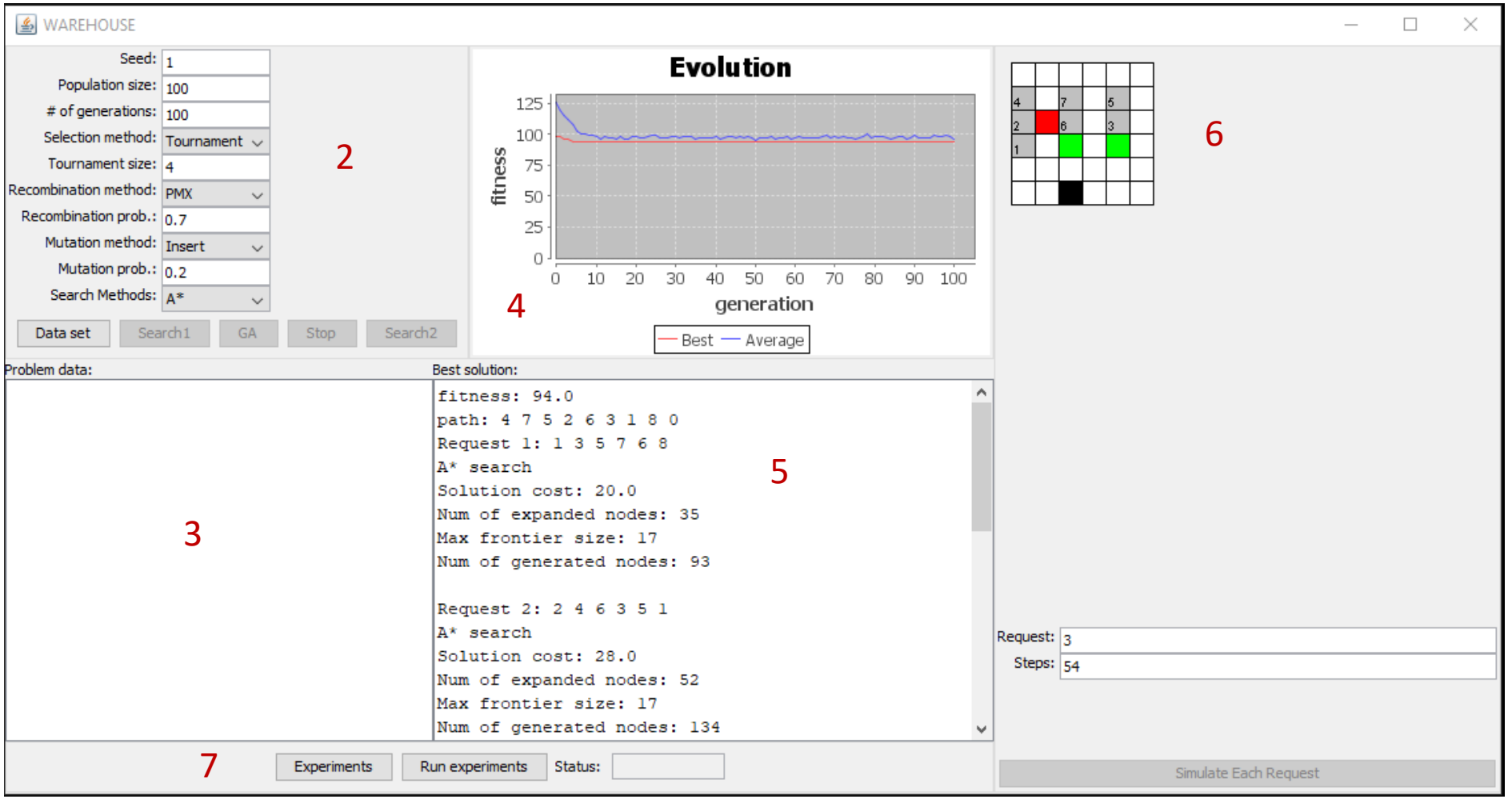
Artificial Intelligence

Project code guide

The program GUI



The program GUI

- It has 7 panels (see panel numbers in previous slide):
    1. Where the user can control the computation of a solution to some problema (see next slide)
    2. Where the user may choose the genetic algorithm parameters
    3. Where the dataset data is presented
    4. Chart that presents the evolution of the average fitness of the population and the fitness of the best individual found so far
    5. Where the best found solution is shown
    6. Where the user can simulate the best solution found
    7. It allows the user to run several experiments in backround

Panel 1

- Panel 1 has the following buttons:
    1. Dataset: allows the user to choose the problem dataset
    2. Search1: allows the user to compute distances between door and shelves and between all shelves
    3. GA: allows the user to run the genetic algorithm
    4. Stop: allows the user to stop the genetic algorithm
    5. Search2: Allows the user to compute the path between the door and the first shelf and between sequential shelves in each request for the best individual found by the genetic algorithm

Packages

- The project is composed of the following packages:
  - `agentSearch`: this package has classes that are needed for defining an agent able to find distances/paths between the warehouse cells
  - `experiments`: this package contains classes that allow the user to run several experiments in backround
  - `ga`: this package contains classes needed to run an instance of a genetic algorithm
  - `gui`: this package contains the classes related to the GUI
  - `search`: this package contains classes needed to run the A* algorithm
  - `statisticsGA`: this package contains classes able to compute and save statistics when several experiments are run on backround
  - `utils`: this package contains classes for file managing and mathematical operations
  - `warehouse`: this package contains classes specific to the problem to be solved

The WarehouseProblemForSearch class

- The `WarehouseForSearch` class represents the problem of finding the distance/path between two cells of the warehouse
- The attributes and methods names are self-explanatory
- You may need to add additional attributes and methods

The WharehouseState class

- The `WarehouseState` class represents a state of the warehouse
- It will be needed in steps 1 and 3 (see project sheet)
- The `lineExit` and `columnExit` attributes represent the door position
- The `steps` attribute has the number of steps already done when simulating a solution
- The remaining attributes and methods names are self-explanatory
- You may need to add additional attributes and methods

The Pair and Request classes

- The `Pair` class represents a pair of shelves; each shelf is represented by a cell (see class `Cell` in the project)
- It is used to save the distance between all pairs of shelves (`value` attribute)

- The `Request` class represents a request of products (see project sheet)
- Each request consists in a sequence of products to be picked from the shelves; products are represented as integer numbers

The WarehouseAgentSearch and HeuristicWarehouse classes

- The `WarehouseAgentSearch` class represents an agent capable of finding the path between two warehouse cells
- `pairs` list: it contains all pairs of door and shelves and of all shelves
- The remaining attributes and methods names are self-explanatory

- The `HeuristicWarehouse` class represents a heuristic to be used by the A* algorithm

The WarehouseProblemForGA class

- The `WarehouseForGA` class represents the problem of finding the best assignment of products to shelves to be solved by the genetic algorithm
- The `getNewIndividual()` method is used to build new individuals in the creation of the initial population
- You may need to add additional attributes and methods

The WarehouseIndividual class

- The `WarehouseIndividual` class represents an individual of the genetic algorithm for the warehouse problem
- It extends the `IntVectorIndividual` from package `ga` (please, analyse this class)
- The `getShelfPos()` method receives the genome and a product number and returns the position of the shelf in the list of shelves in the WarehouseAgentSearch class
- The `getProductInShelf()` method returns the product number if the shelf in cell [line, column] has some product and 0 otherwise
- These methods are already being used in the GUI; You may also need to use them in the `computeFitness()` method
- You may need to add additional attributes and methods

The WarehouseExperimentsFactory class

- The `WarehouseExperimentsFactory` allows to build an experiment (panel 7)
- You may need to change this class after implementing new recombination and mutation operators (search for “TODO”)
- You should use the names defined in this class in the text file where you define your experiments

The Recomination# and Mutation# classes

- You should implement new recombination operators in classes `Recombination2` and `Recombination3`
- You should implement new mutation operators in classes `Mutation2` and `Mutation3`
- You should change the name of these classes; You should change class `WearehouseExperimentsFactory` accordingly
- Hint: if you use a permutation based representation for the individuals, you may look for the following genetic operators in the internet:
  - Order crossover (OX), Cycle crossover (CX), among others
  - Inversion mutation, swap mutation, among others

The MainFrame and PanelSimulation classes

- It may be instructive to analyse the following methods of class `MainFrame`:
  - `jButtonRunSearch1_actionPerformed()`
  - `jButtonRunGA_actionPerformed()`
  - `jButtonRunSearch2_actionPerformed()`
- It may also be instructive to analyse the following methods of class `PanelSimulation`
  - `buttonSimulate_actionPerformed()`
  - `environmentUpdated()`