U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO	L.EIC – BSc/First Degree in Informatics and Computing Engineering **Artificial Intelligence**	2022/2023 (3rd Year) 2nd Semester
Luís Paulo Reis, Henrique Lopes Cardoso, Telmo Barros, Ricardo Urjais, Miguel Ferreira		

Assignment No. 1

Heuristic Search Methods for Problem Solving

Adversarial Search Methods for Games

Optimization Methods/Meta-Heuristics

Theme/Topics

IART's first practical assignment consists in the development of an assignment related to one of three possible main topics and choosing a specific subject for the assignment.

Topic 1: Heuristic Search Methods for One Player Solitaire Games

A solitaire game is characterized by the type of board and pieces, the rules of movement of the pieces (possible operators/plays), and the conditions for ending the game with defeat (impossibility to solve, maximum number of moves reached, time limit reached) or victory (solitaire solved), together with the respective score. Typically, in the event of a win, a score is awarded depending on the number of moves, resources spent, bonuses collected and/or time spent.

In addition to implementing a solitaire game for a human player, the program must be able to solve different versions/levels of this game, using appropriate search methods, focusing on the comparison between uninformed search methods (breadth-first search, depth-first search, iterative deepening, uniform cost) and heuristic search methods (greedy search, A* Algorithm, Weighted A*, …), with different heuristic functions. The algorithms employed should be compared with several criteria, with emphasis on the quality of the solution obtained, number of operations performed, maximum memory used, and time spent to obtain the solution. All games, if it is possible, should have variable board size and a set of puzzles to solve with different difficulty levels.

The application should have a text or graphical use interface to show the evolution of the board and interact with the user/player. You should allow a game mode in which the PC solves the solitaire alone using the algorithm and respective configuration as selected by the user. Optionally, you can allow a Human game mode in which the user can solve the game, while asking the PC for “hints”.

Topic 2: Adversarial Search Methods for Two-Player Board Games

A board game is characterized by the type of board and pieces, the rules of movement of the pieces (operators) and the finishing conditions of the game with the respective score. In this work, the aim is to implement a game for two players and solve different versions of this game, using the Minimax search algorithm with αβ cuts and its variants.

Human-human, human-computer and computer-computer game modes should be developed, where the computer should exhibit different skills (levels of difficulty). Computer performance should be compared regarding the different skills (e.g., hard, medium, easy), corresponding to different evaluation functions, different depth levels of Minimax, different successor generation ordering and/or variants of the Minimax algorithm. Monte Carlo Tree Search with different configurations may also be use. Emphasis should be placed on the analysis of the results of the computer players (wins, draws, losses, and other quality parameters, such as the number of plays to obtain the win/loss) and average time spent to obtain the plays. All games, if it is possible, should have different versions with variable board size.

The application to be developed must have a proper text or graphical user interface, to show the evolution of the board and interact with the user/player. Different game modes must be included, as explained above, allowing the selection of the game mode, type of each player, and skills of computer players. You should allow different skilled computer players to play against each other. You may also consider providing human players with movement "hints".

Topic 3: Metaheuristics for Optimization/Decision Problems

An optimization problem is characterized by the existence of a (typically large) set of possible solutions, comparable to each other, of which one or more are considered (globally) optimal solutions. Depending on the specific problem, an evaluation function allows you to establish this comparison between solutions. In many of these problems, it is virtually impossible to find the optimal solution, or to ensure that the solution found is optimal and, as such, the goal is to try to find a local optimal solution that maximizes/minimizes a given evaluation function to the extent possible.

In this work, the aim is to implement a system to solve an optimization problem, using different algorithms or meta-heuristics, such as hill-climbing, simulated annealing, tabu search, and genetic algorithms (you may include other algorithms or variations of these). Multiple instances of the chosen problem must be solved, and the results obtained by each algorithm must be compared. Different parameterizations of the algorithms should be tested and compared, in terms of the average quality of the solution obtained and the average time spent to obtain the solutions. All problems should have different size/difficulty to solve.

The application to be developed should have an appropriate visualization in text or graphic mode, to show the evolution of the quality of the solution obtained along the way and the final (i.e. local optimal) solution, and to interact with the user. You should allow the selection and parameterization of the algorithms and the selection of the instance of the problem to be solved.

Programming Language

Any programming language and development system can be used, including, at the language level, C++, Java, Python, C#, among others. The choice of language and development environment to be used is the entire responsibility of the students. However, the use of Python is typically preferred. Students may use any library that may be useful for the work (NumPy, SciPy, PyGame, MatplotLib, among many others),

Groups

Groups must be composed of 3 students (exceptionally 2). Individual groups or groups composed of 4 students are not accepted. Groups should be composed of students attending the same practical class (although exceptions are possible). All students must be present in the checkpoint sessions and presentation/demonstration of the work. Groups composed of students from different classes are discouraged, given the logistic difficulties of performing work that this can cause.

Checkpoint

Each group must submit in Moodle a brief presentation (max. 5 slides), in PDF format, which will be used in the class to analyze, together with the teacher, the progress of the work. The presentation should contain: (1) specification of the work to be performed (definition of the game or optimization problem to be solved), (2) related work with references to works found in a bibliographic search (articles, web pages and/or source code), (3) formulation of the problem as a search problem (state representation, initial state, objective test, operators (names, preconditions, effects and costs), heuristics/evaluation function) or optimization problem (solution representation, neighborhood/mutation and crossover functions, hard constraints, evaluation functions), and (4) implementation work already carried out (programming language, development environment, data structures, among others).

Final Delivery

Each group must submit in Moodle two files: a presentation (max. 10 slides), in PDF format, and the implemented code, properly commented, including a "readme" file with instructions on how to compile, run and use the program. Based on the submitted presentation, the students must carry out a demonstration (about 10 minutes) of the work, in the practical class, or in another period to be designated by the teachers of the course.

The file with the final presentation should include, in addition to the aforementioned for the checkpoint, details on: (5) the approach (heuristics, evaluation functions, operators, ...) and (6) implemented algorithms (search algorithms, minimax, metaheuristics), as well as (7) experimental results, using appropriate tables/plots and comparing the various methods, heuristics, algorithms and respective parameterizations for different scenarios/problems. The presentation shall include a slide of conclusions and another of references consulted and materials used (software, websites, scientific articles, ...).

Suggested Problems

Topic 1: One Player Solitaire Games

1A) Atomix – https://en.wikipedia.org/wiki/Atomix_(video_game)

1B) Symmetry Puzzles – https://erich-friedman.github.io/puzzle/symmetry/

1C) Space Block – Roll the Block - https://play.google.com/store/apps/details?id=com.zawor.spaceblock&hl=en&gl=US

1D) Puzzle Packed IQ Games – Klotski - https://play.google.com/store/apps/details?id=saiwen.game.klotski&hl=en&gl=US

1E) Cohesion Free - https://play.google.com/store/apps/details?id=com.NeatWits.CohesionFree&hl=en&gl=US

1F) ASAE Route Optimization (Topic 3) using Search Algorithms

Topic 2: Two-Player Adversarial Games

2A) LESS – https://www.kickstarter.com/projects/less/less-like-chess-but-less

2B) CIFRA Code25 – https://www.kickstarter.com/projects/logygames/cifra-code25

2C) Wana – https://boardgamegeek.com/boardgame/364012/wana

2D) Bound – https://boardgamegeek.com/boardgame/375975/bound

2E) 18 Ghosts - https://boardgamegeek.com/boardgame/70116/18-ghosts

2F) Black-Hole Escape - https://boardgamegeek.com/boardgame/64229/black-hole-escape

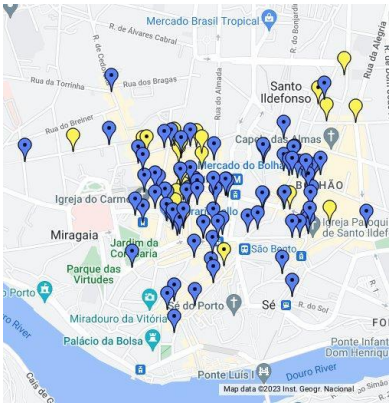
Topic 3: Optimization Problems - Optimal Inspection Routes Problem

Introduction

ASAE is the Portuguese Economic and Food Safety Authority is a specialized authority responsible for food safety and economic surveillance in Portugal. With over 3 million of registered establishments spread nationwide, their task of monitoring all of them becomes very challenging. One of their responsibilities is actively inspecting these establishments. The "Optimal Inspection Routes Problem" addressed here is a simplified version of ASAE's problem of planning and resources' allocation to cover the establishments' inspections in the most optimal way.


Input Data Set

The input data set is based on the real-world dataset and contains establishments from the Porto district. It is divided into two csv files: (i) "establishments.csv" contains the overall information of 1000 establishments and (ii) "distances.csv" contains the time distances between every one of these establishments. Their structure is explained below



establishments.csv

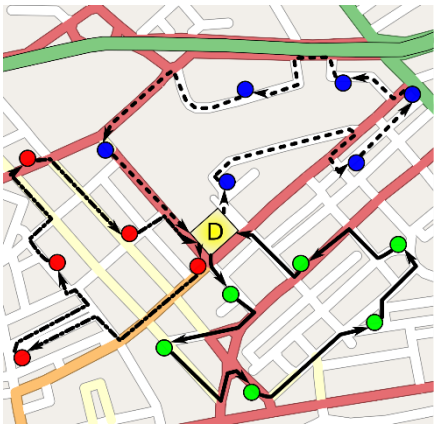
- ID *(integer)* - unique identifier of the establishment
- Name *(string)* - Establishment's name
- Latitude *(float)* - Geographical location's latitude
- Longitude *(float)* - Geographical location's longitude
- Inspection duration (integer) - Estimated duration, in **minutes**, of the visit/inspection duration based on the establishment
- Inspection utility *(float)* - Value from 0 to 1 that represents the utility of inspecting that establishment. This value is directly related to the risk this establishment poses to society.
- Opening hours *(list)* - Set of hours during the day when establishments can be inspected by the inspectors. This is a list of 24 binary values indicating whether the establishment is open or closed in each hour of the day. (See examples below)

Open	Representation
All day	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
08:00 - 12:00 and 14:00 18:00	0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0

distances.csv

This file contains the estimated travel time matrix (in seconds) from each one of the establishments to another. Despite existing 1000 establishments the size of the matrix is 1001x1001 and the first row and column is reserved for the departure/arrival point (represented by the **ID: 0**).

Problem Description



This problem is a simplified version of the actual ASAE inspection route problem and is an instance of the Vehicle Routing Problem. ASAE uses inspection brigades to check economic operators for compliance with regulations. Each brigade is assigned a certain number of inspections and must start and end their route at the Depot. Inspections must also align with the establishments'' schedules and can only begin if the operator is open (must wait otherwise). However, once an inspection has started, it can be completed regardless of the operator's scheduled closing time. All the problems must consider 9:00 AM to be the hour of departure for every vehicle. Multiple scenarios are presented below with different constraints and objectives. (Each group may choose **just one primary scenario,** but you are free to explore other scenarios and different constraint combinations, turning the problem harder).

3A) Travel time minimization

In this variant there are a finite number of available vehicles - ***k***. ***k*** can be defined by ***floor(0.1 x n)*** where n is the dataset size. So, for instance, given 200 establishments, 20 vehicles must be used. Their number of working hours is unlimited.
The goal in this problem is to inspect all the establishments in the minimum possible time, considering travel, waiting and inspection time.

3B) Resource allocation minimization

In this variant there are an infinite number of available vehicles. Each ones' route must not exceed the 8 working hours by any chance. The vehicles are not required to finish their routes in the departure point, i.e. the route is considered to be finished immediately after the last inspection.
The goal in this problem is to inspect all the establishments using the minimum number of vehicles possible. The inspection's duration must be considered 5 minutes.

3C) Inspection's utility maximization

In this variant there are a finite number of available vehicles - ***k***. ***k*** can be defined by ***floor(0.1 x n)*** where n is the dataset size. So, for instance, given 200 establishments, 20 vehicles must be used. Each ones' route must not exceed the 8 working hours by any chance.
The goal in this problem is to inspect the establishments that maximize the sum of inspection' utilities. The inspection's duration must be considered 5 minutes.

3D) Inspected establishments maximization

In this variant there are a finite number of available vehicles - ***k***. ***k*** can be defined by ***floor(0.1 x n)*** where n is the dataset size. So, for instance, given 200 establishments, 20 vehicles must be used. Each ones' route must not exceed the 8 working hours by any chance.
The goal in this problem is to inspect the maximum number of establishments.

3E) Waiting time minimization

In this variant there are a finite number of available vehicles - ***k***. ***k*** can be defined by ***floor(0.1 x n)*** where n is the dataset size. So, for instance, given 200 establishments, 20 vehicles must be used. Their number of working hours is unlimited.
The goal in this problem is to minimize the sum of the vehicles' waiting time. This waiting time is the difference between the vehicle's time of arrival and the establishment's opening hour. The waiting time is 0 if the vehicle arrives when the establishment is open.

3F) Multi-day planning

In this variant there are a finite number of available vehicles - ***k***. ***k*** can be defined by ***floor(0.1 x n)*** where n is the dataset size. So, for instance, given 200 establishments, 20 vehicles must be used. Each ones' route must not exceed the 8 working hours by any chance.
The goal in this problem is to find the minimum number of necessary days to inspect all the establishments.

Output

The problem may be addressed gradually and solved considering four instances' sizes:

- **Very Small** – First 20 establishments ($1 \leq ID \leq 20$)
- **Small** – First 100 establishments ($1 \leq ID \leq 100$)
- **Medium** – First 500 establishments ($1 \leq ID \leq 500$)
- **Large** – Whole data set ($1 \leq ID \leq 1000$)

The desired solution should provide a list of the IDs of the inspected economic operators, in the order they were inspected, for each inspection brigade. Additionally, it should include the following metrics: **total travel time, total wait time**, **number of vehicles/brigades utilized**, **total utility for each brigade**, **number of inspected economic operators**. In terms of performance, the following metrics are also required: **time taken to reach the optimal solution, total time spent running the algorithm, number of iterations to reach the optimal solution** and **total number of iterations executed**.

Note: Do not forget the original assignment's specification: "The application to be developed should have an **appropriate visualization in text or graphic mode**, to show the **evolution of the quality of the solution** obtained along the way and the final (i.e. local optimal) solution, and to interact with the user. You should allow the **selection and parameterization of the algorithms** and the **selection of the instance of the problem** to be solved."