U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO	**MECD –Master Degree in Data Science and Engineering** **Artificial Intelligence**	**2022/2023 (1st Year) 2nd Semester**
	TEACHER: Luís Paulo Reis	

Practical Work

Heuristic Search Methods for Problem Solving

Adversarial Search Methods for Games

Optimization Methods/Meta-Heuristics

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 methods applied should be compared at various levels, with emphasis on the quality of the solution obtained, number of operations performed, maximum memory used, and time spent to obtain the solution. The work may also include the use of Metaheuristics and its comparison with the use of search methods.

The application should have a text or graphic view 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 method and its configuration selected by the user. Optionally, you can allow a Human game mode in which the user can solve the game, eventually 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 tiles, the rules of movement of the pieces (operators/possible moves) 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 method 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. The work may also include a Monte Carlo Tree Algorithm and its comparison with Minimax. 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 solutions/plays.

The application to be developed must have a proper user interface in text or graphic mode, to show the evolution of the board and interact with the user / player. You must allow the game modes indicated 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 with 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.

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 solutions, 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#, Prolog, 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 2 students (exceptionally 1 or 3).

Final Delivery

Each group must submit in Moodle three files: a paper/report (in IEEE format), a presentation (max. 20 slides), in PPT 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 presentation with demonstration (about 10 minutes + 5 minutes for discussion) of the work, in the final evaluation class.

The paper/report and the presentation should contain: (1) specification of the work 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, rigid constraints, evaluation functions), (4) implementation details (programming language, development environment, data structures, file structure, among others), (5) the approach (heuristics, evaluation functions, operators, ...) and (6) algorithms implemented (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 paper and presentation shall include also, introduction and conclusions and the references consulted and materials used (software, websites, scientific articles, ...).

Suggested Problems

Topic 1: One Player Solitaire Games

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

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

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

Topic 2: Two-Player Adversarial Games

2A) Mancala - https://play.google.com/store/apps/details?id=com.donkeycat.mancala&hl=en&gl=US
https://en.wikipedia.org/wiki/Mancala

2B) Fanorona - https://play.google.com/store/apps/details?id=com.algotgames.fanorona&hl=en&gl=US
https://en.wikipedia.org/wiki/Fanorona

2C) Halma - https://play.google.com/store/apps/details?id=com.AdelanteGames.Halma&hl=en&gl=US
https://en.wikipedia.org/wiki/Halma

Topic 3: Optimization Problems

3A) Optimal Inspection Routes Problem – ASAE Use Case

3B) Compiling Google:
https://storage.googleapis.com/coding-competitions.appspot.com/HC/2019/hashcode2019_final_task.pdf
https://storage.googleapis.com/coding-competitions.appspot.com/HC/2019/final_round_2019.in.zip

3C) Photo slideshow:
https://storage.googleapis.com/coding-competitions.appspot.com/HC/2019/hashcode2019_qualification_task.pdf
https://storage.googleapis.com/coding-competitions.appspot.com/HC/2019/qualification_round_2019.in.zip

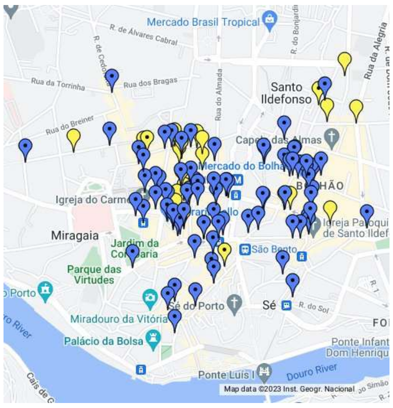
Optimal Inspection Routes Problem – ASAE Use Case

1. 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 millions 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.

2. 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 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**).

3. 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 scenario** but you are free to explore different constraint combinations, turning the problem harder).

3.1. 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.

3.2. 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.

3.3. 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.

3.4. 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.

3.5. 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.

3.6. 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.

4. Output

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

- **Very Small** - First 20 establishments (1 ≤ ID ≤ 20)
- **Small** - First 100 establishments (1 ≤ ID ≤ 100)
- **Medium** - First 500 establishments (1 ≤ ID ≤ 500)
- **Large** - Whole data set (1 ≤ ID ≤ 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**.

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."