U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO	L.EIC – BSc/First Degree in Informatics and Computing Engineering **Artificial Intelligence**	2024/2025 (3rd Year) 2nd Semester

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 of 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/moves), 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*, …), all using with different heuristic functions. The algorithms employed should be compared with several criteria, with emphasis on the quality of the solution obtained, the number of states generated, maximum memory used, and time spent to obtain the solution. All games, if it is possible, should have variable board sizes and a set of puzzles to solve with different difficulty levels. The application should be able to read the puzzle state (board position) from text files and to store the results (including the puzzle solution, time and memory taken for each algorithm, final results, etc.) also in text files.

Students should focus on firstly developing a simple version of the game with a small size board, the algorithms and testing that everything works (algorithms are able to solve the simple small puzzles), before proceeding to more complex and bigger scenarios.

The application should have a graphical user interface to show the evolution of the board and interact with the user/player. It should allow a game mode in which the PC solves the solitaire alone using the algorithms implemented and configuration selected by the user, giving the details of the solution, and time and memory used to solve the puzzle. The application should also allow a Human game mode in which the user can interactively solve the game while asking the PC for “hints” (next move).

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 (typically win for player 1, win for player 2 or draw). 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 and with Monte Carlo Tree Search 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 should also be used. 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 the average time spent and memory used to obtain the computer moves. All games, if it is possible, should have different versions with variable board sizes. The application should be able to read the game state (board position) from text files and to store the results (including the moves done by each player, time taken, final results, etc) also in text files.

Students should focus on firstly developing a simple version of the game with a small size board, the algorithms and testing that everything works (algorithms are able to show intelligent/correct moves in simple game situations), before proceeding to more complex and bigger scenarios.

The application to be developed must have a proper 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, algorithms used and skills of each of the computer players. The application should enable selecting differenting levels and heuristics for each of the computer players to play against each other. The work should also enable providing human players with the next movement suggestion "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 locally 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. Other algorithms or variations of these algorithms may also be included. Multiple instances, with different sizes and complexities, 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. In the algorithm comparison try to make all the algorithms take similar time in achieving the solution and compare the quality of the solutions (for example making 20x more iterations for a tabu search algorithm than for a genetic algorithm with population of 20). All problems should have different sizes/difficulties to solve. The application should be able to read the problems from text files and to store the results (including the path to the solution, quality, time, etc.) also in text files.

Students should focus on firstly developing a simple version of the application with a small simple problem, the algorithms and testing that everything works (all algorithms achieve the optimal solution for the simple problem), before proceeding to more complex and bigger scenarios.

The application to be developed should have an appropriate visualization in graphic mode, to show the evolution of the solution and its quality obtained along the way. The application should also enable to analyze the final (i.e. local optimal) solution(s), and to interact with the user. You should provide in the interface the means for 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). Groups should be composed of students attending the same practical class. All students in a group 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) a 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) Jelly Field Puzzle (known sequence of pieces):

https://play.google.com/store/apps/details?id=com.game.juicy.field&hl=en

1B) Bird Sort 2 - Color Puzzle:

https://play.google.com/store/apps/details?id=com.globalplay.birdsort2.color.puzzle&hl=en

1C) Wood Block - Puzzle Games (known sequence of pieces):

https://play.google.com/store/apps/details?id=com.block.puzzle.free.wood&hl=en

1D) Cake Sort Puzzle (known sequence of cakes):

https://play.google.com/store/apps/details?hl=en_US&id=com.falcon.dm.water.cake.sort.puzzle&utm_source=chatgpt.com

1E) FreeCell Card Solitaire:

https://play.google.com/store/apps/details?id=at.ner.SolitaireFreeCell&hl=en

1F) Bakers Dozen Card Solitaire:

https://www.solitaire.org/bakers-dozen/

Topic 2: Two-Player Adversarial Games

2A) Ultimate Tic-Tac-Toe

https://michaelxing.com/UltimateTTT/v3/

https://en.wikipedia.org/wiki/Ultimate_tic-tac-toe

2B) Yinsh

https://boardgamegeek.com/boardgame/7854/yinsh

2C) Yonmoque-Hex

https://boardgamegeek.com/boardgame/436894/yonmoque-hex

2D) Abacus

https://boardgamegeek.com/boardgame/437010/abacus

2E) Churn

https://boardgamegeek.com/boardgame/437052/churn

2F) Partitions

https://boardgamegeek.com/boardgame/439137/partitions

Topic 3: Optimization Problems

3A) Router placement

Hash Code 2017 Final Task

Hash Code 2017 Final Round Input Data

3B) Streaming videos

Hash Code 2017 Qualification Task

Hash Code 2017 Qualification Round Input Data

3C) Delivery

Hash Code 2016 Qualification Task

Hash Code 2016 Qualification Round Input Data

3D) Patient Allocation in Hospitals

https://www.sciencedirect.com/science/article/pii/S2211692323000139

https://data.mendeley.com/datasets/3mv4rtxtfs/1

3E) Stock Portfolio Optimisation (optimization with meta-heuristics)

https://www.kaggle.com/datasets/ashbellett/australian-historical-stock-prices

3F) Wedding Seater Planner

https://medium.com/analytics-vidhya/building-a-wedding-seating-plan-using-probabilistic-methods-simulated-annealing-8f31d8987026