SlideQuest

Heuristic Search Methods for One Player Solitaire Games

Lara Neves and Helena Costa
Artificial Intelligence
Faculty of Engineering, University of Porto
February 2024

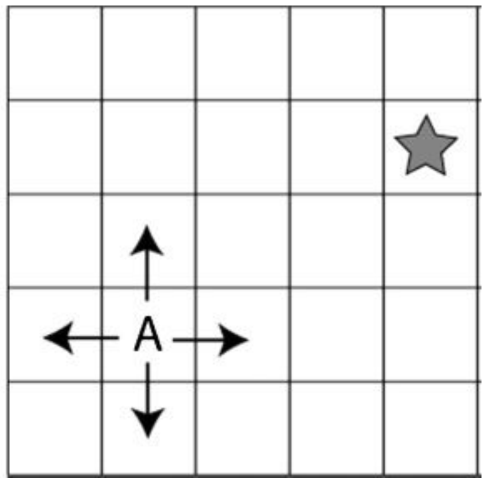
Problem description

This Solitaire Puzzle Solver project aims to develop a computer program capable of solving various levels of difficulty and comparing different search methods and heuristics to assess their effectiveness in finding solutions to the game.

GRID: The puzzle is made up of a NxN grid, with a start and goal positions. Some spots in the grid can be obstacles and are blocked.

MOVEMENT: Limited number of movement (Left, Right, Up, Down), depending on the puzzle.

OBJECTIVE: The objective is to, for a given puzzle, decide the correct or best sequence of movements for the piece to achieve the goal.



Problem Formulation as a Search Problem

State Representation: The state of the game (S) can be represented by the positions of the player's piece (an array of size N of the movements chosen), the goal's piece, and obstacles.

Initial State: The initial state is determined by the level selected.

Objective State: The objective state is when the player reaches the goal position.

Operators: Operators define how the player can move from one state to another.

Operator	*Pre-condition*	*Effect*	*Cost*
MoveUp	player's position must not be at the top edge of the grid, and the cell above the player must not be an obstacle	Update the player's position based on the given direction, if the move is valid.	1
MoveDown	player's position must not be at the bottom edge of the grid, and the cell below the player must not be an obstacle		
MoveLeft	player's position must not be at the left edge of the grid, and the cell to the left of the player must not be an obstacle		
MoveRight	player's position must not be at the right edge of the grid, and the cell to the right of the player must not be an obstacle		
(all)	The direction must be one of 'up', 'down', 'left', or 'right'. The player's position must be within the bounds of the grid		

Implementation

LANGUAGE: Python

DATA STRUCTURES:

Lists: `obstacle_positions, possible_moves, steps`

Tuples: `(x, y)`

Deque, Heap, Sets

ALGORITHMS: Simulations of sequences of movements

FEATURES:

Pygame - graphical representation of the game board

Pygame_Menu - graphical elements such as buttons and menus for intuitive interactions

Search Methods Overview

Uninformed search methods

- Breadth-First Search
- Depth-First Search
- Depth-Limited Search (limits 5 and 8)
- Iterative Deepening Search
- Uniform Cost Search

Heuristic search methods

- Greedy Search
- A* algorithm
- Weighted A* algorithm

Note: As the cost of our operators is 1, Uniform Cost Search is equivalent to Breadth-First Search

Heuristics

Start: $(x1, y1)$
Goal: $(x2, y2)$

- Manhattan Distance (H1)
  - Calculates the distance between two points by summing the absolute differences of their x and y coordinates.
  - $H1 = |x2 - x1| + |y2 - y1|$

- Max Distance (H2)
  - Calculates the distance between two points by taking the maximum absolute difference of their x and y coordinates.
  - $H2 = \max(|x2 - x1|, |y2 - y1|)$

Metaheuristics Integration

- **Hill-Climbing**
  - Continuously moves towards the direction of increasing elevation until it reaches a local maximum, where no neighboring solution yields a better outcome
  - can get stuck in local optima
  - simpler and computationally less expensive

- **Simulated Annealing**
  - Occasionally accepts solutions that are worse than the current one, with a probability that decreases over time according to a cooling schedule
  - balances exploration and exploitation
  - more effective for complex problems

Game Modes

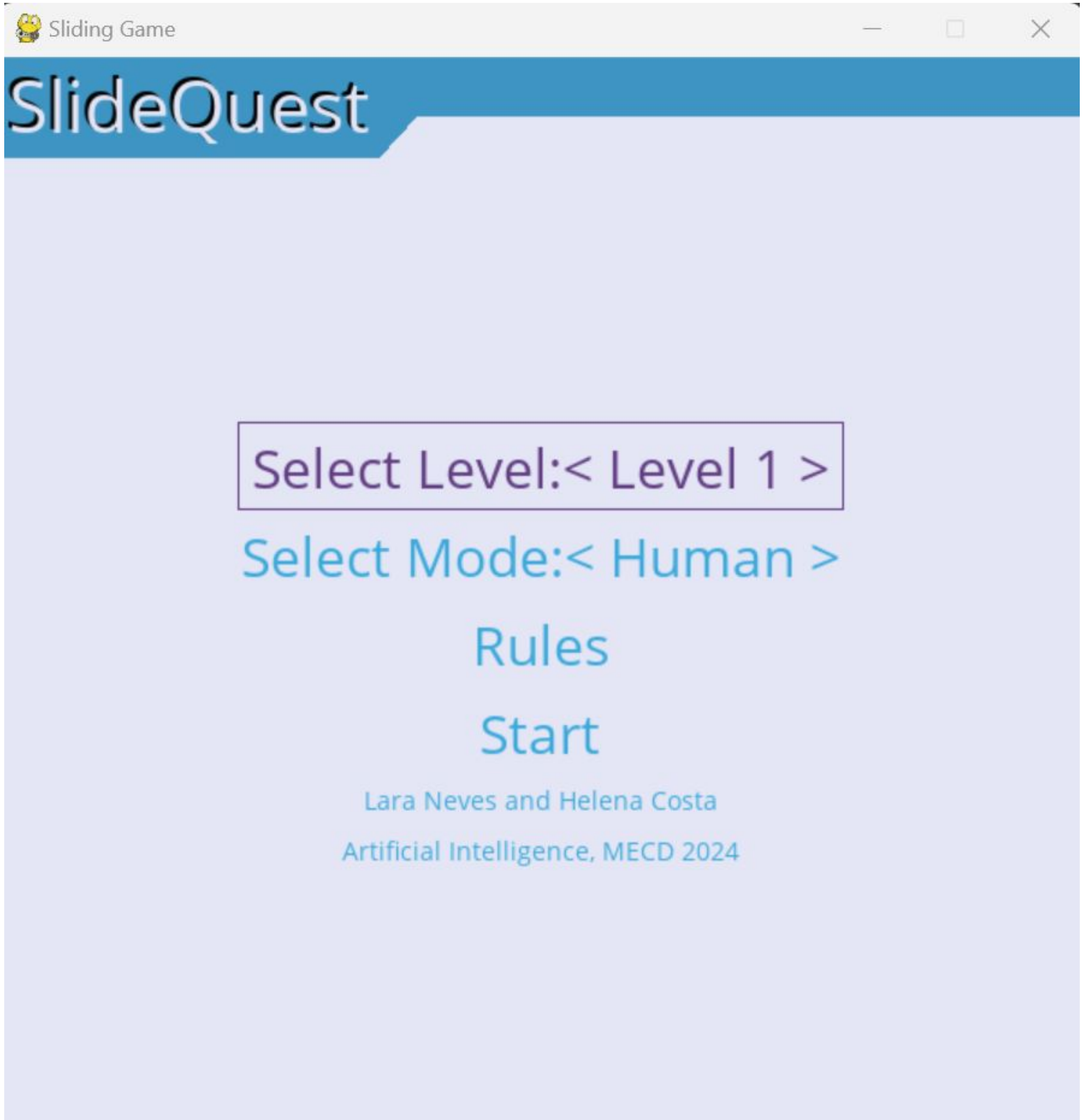
SELECT LEVEL

Level 1: 4x4 Grid, 3 obstacles
Level 2: 4x4 Grid, 5 obstacles
Level 3: 4x4 Grid, 4 obstacles
Level 4: 6x6 Grid, 12 obstacles
Level 5: 6x6 Grid, 12 obstacles
Level 6: 6x6 Grid, 11 obstacles
Level 7: 8x8 Grid, 18 obstacles
Level 8: 8x8 Grid, Impossible to solve

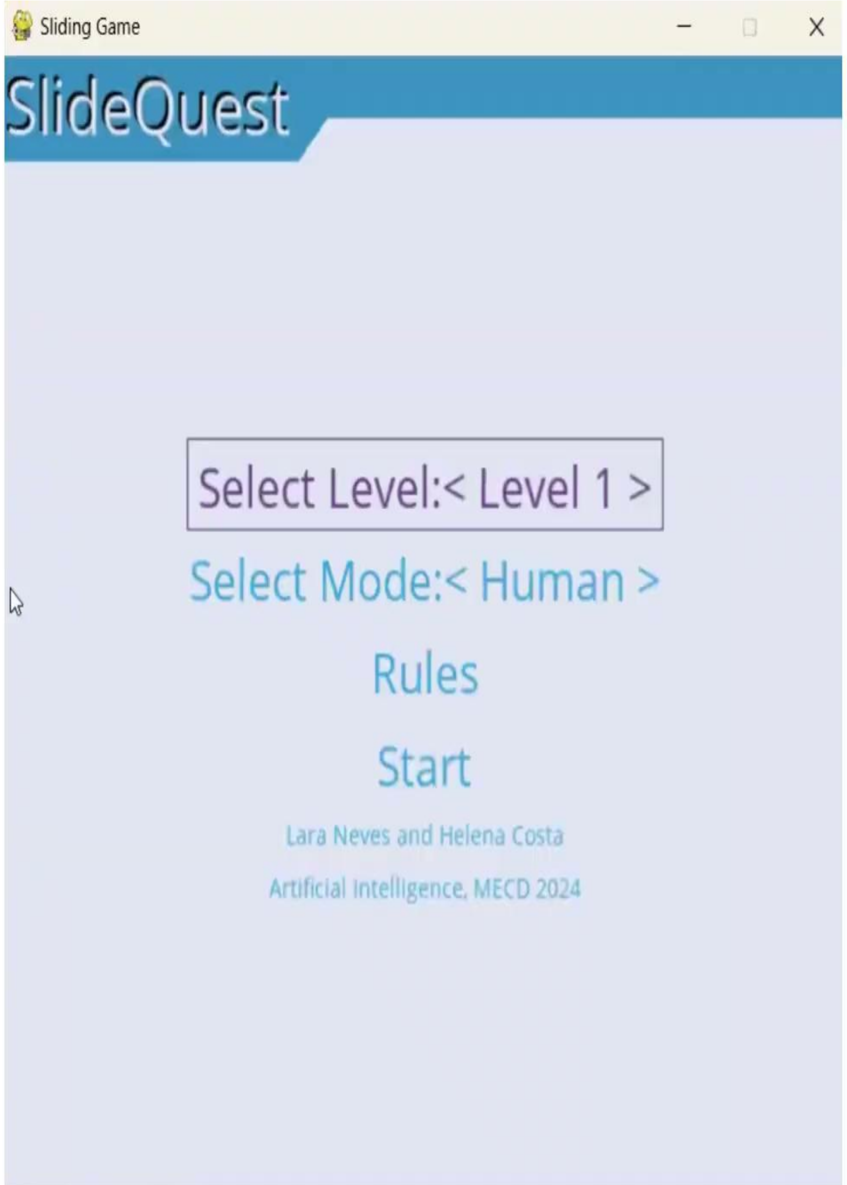
SELECT MODE

Human
Random AI
Breadth-First Search
Depth-First Search
Depth-Limited Search (limits 5 and 8)
Iterative Deepening Search
Uniform Cost Search
Greedy Search
A* algorithm
Weighted A* algorithm
Hill-Climbing
Simulated Annealing

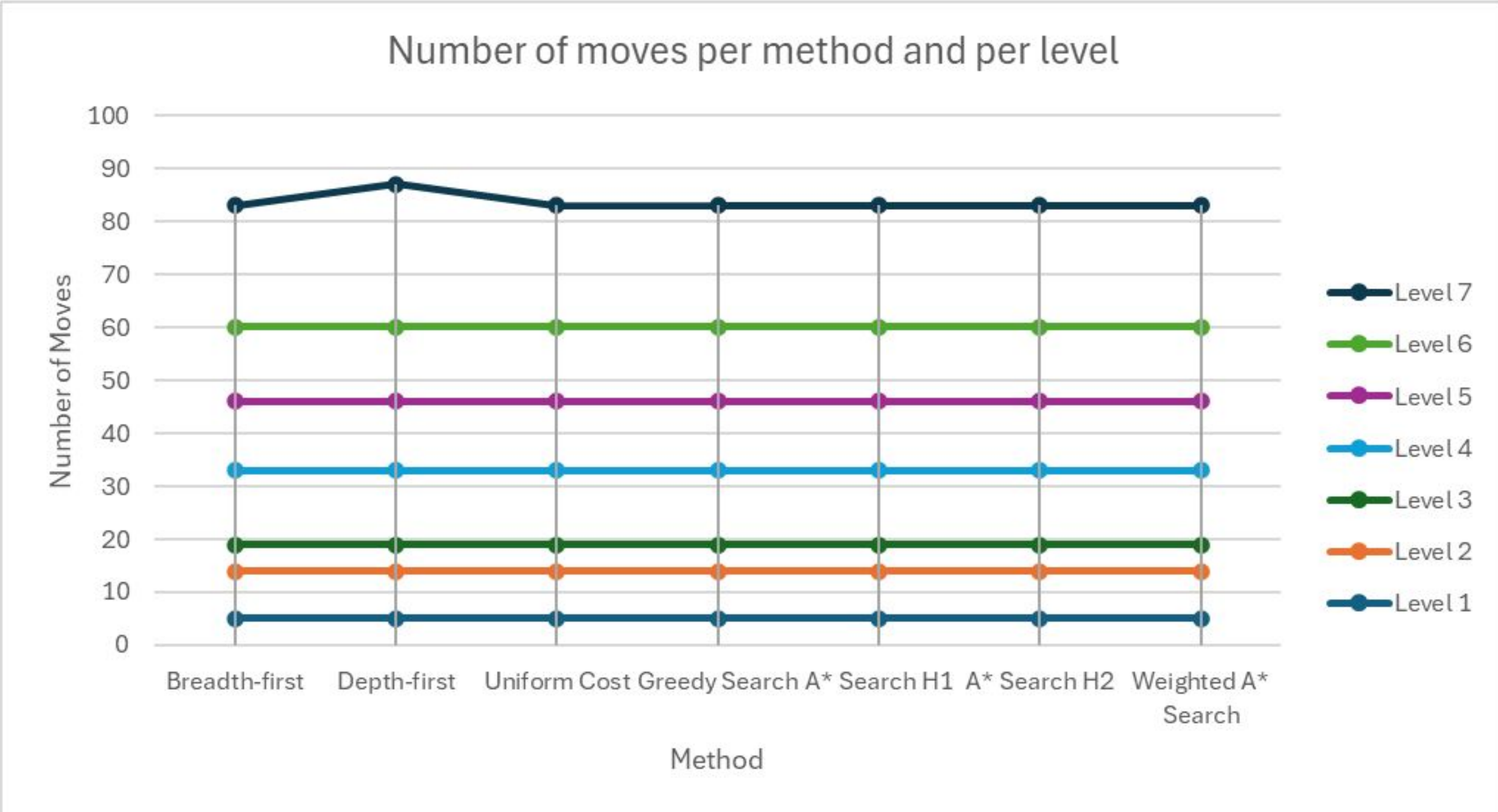
Game Modes


Sliding Game
SlideQuest
Select Level:< Level 1 >
Select Mode:< Human >
Rules
Start
Lara Neves and Helena Costa
Artificial Intelligence, MECD 2024

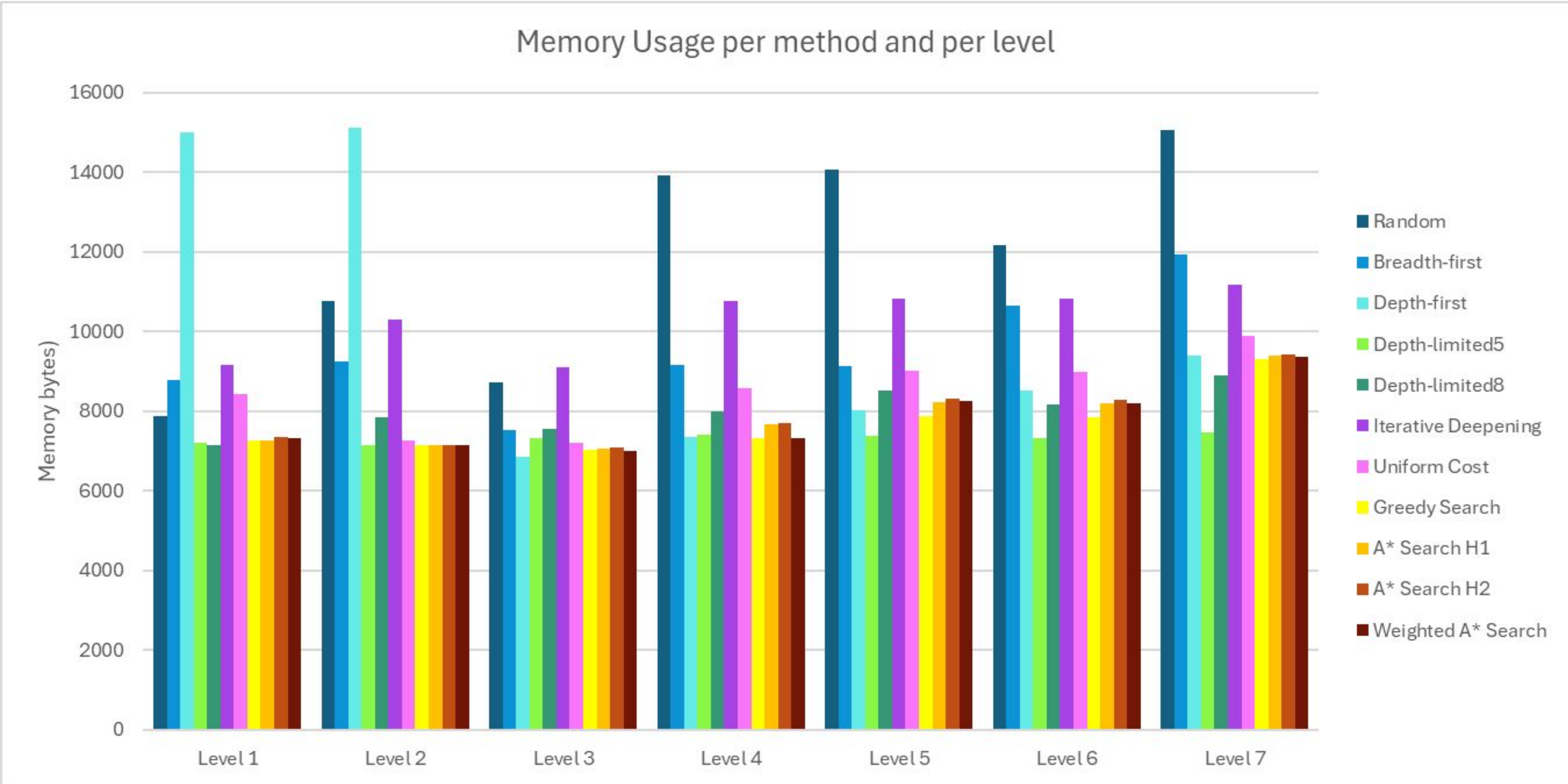
Game Modes


Sliding Game
SlideQuest
Select Level:< Level 1 >
Select Mode:< Human >
Rules
Start
Lara Neves and Helena Costa
Artificial Intelligence, MECD 2024

Comparisons

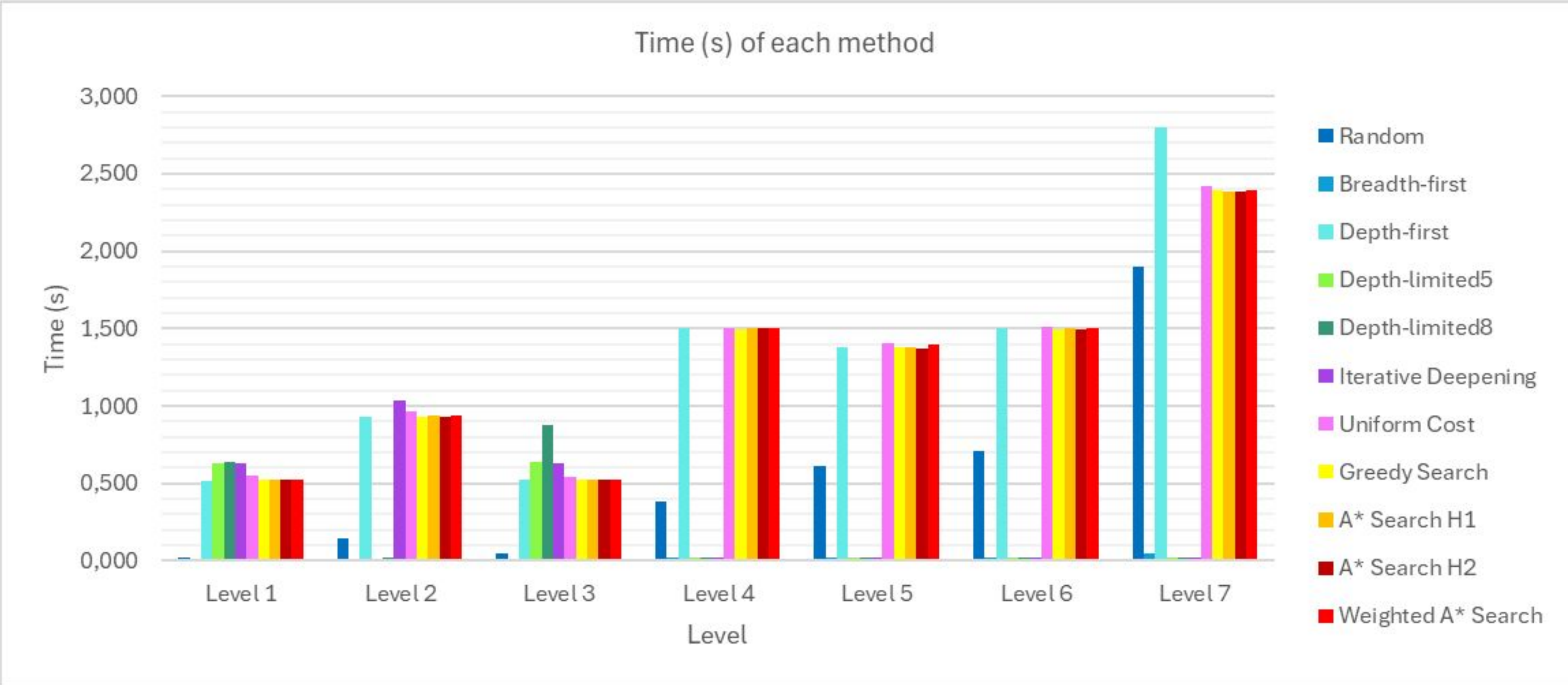


Comparisons



There is a tendency for higher memory requirements for the methods breadth-first and iterative deepening.
Depth first requires a lot of memory and is not efficient for very simple levels!
Random uses a lot of memory at advanced levels but is used as a reference.
Greedy, A* and weighted A* are the lightest.

Comparisons



Time gets higher over time
Greedy, A* and Weighted A* are quite similar
Breath first and depth limited seems to be the best

The winner is

Memory: Depth-Limited (5)

Moves: All (except Depth First)

Time: Breath-First

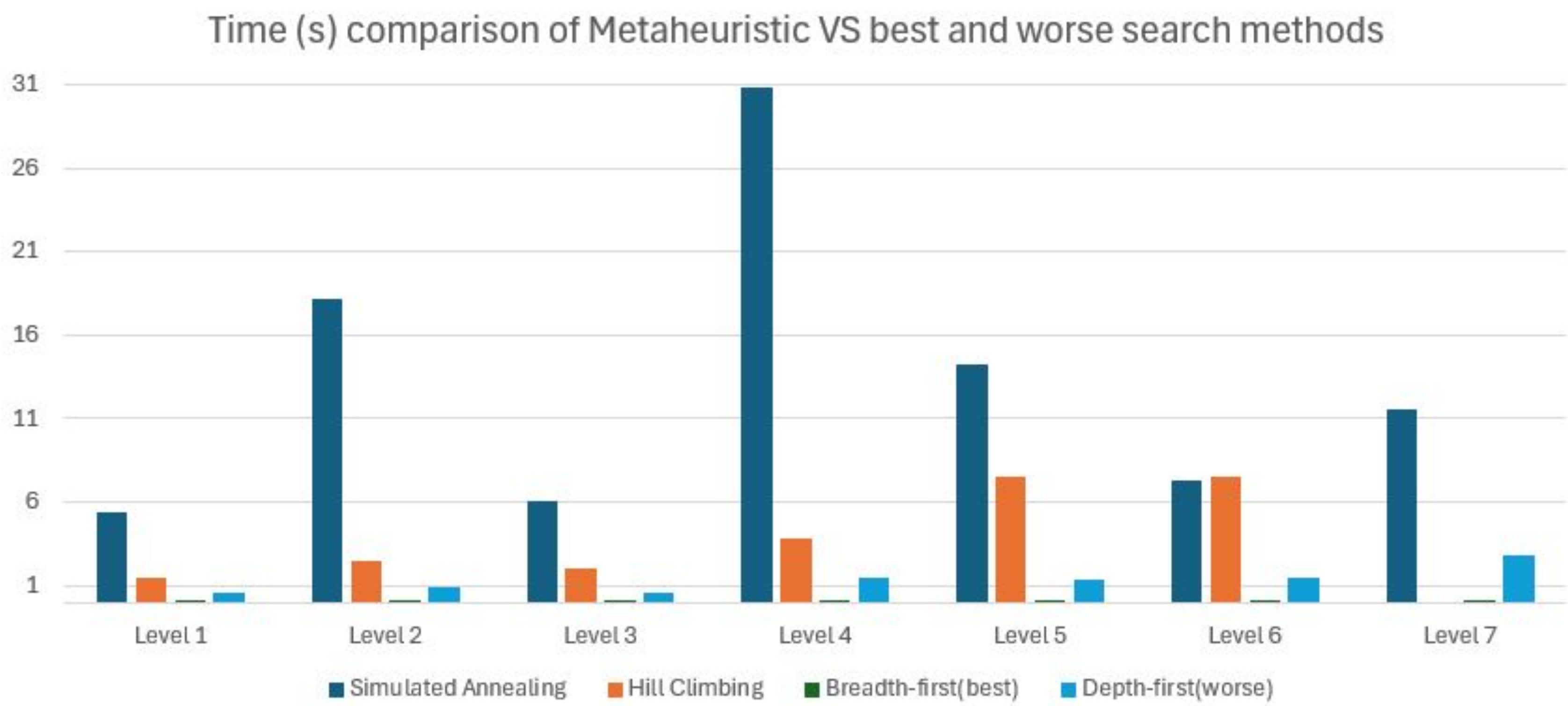
The loser is

Memory: Iterative Deepening

Moves: Depth-First

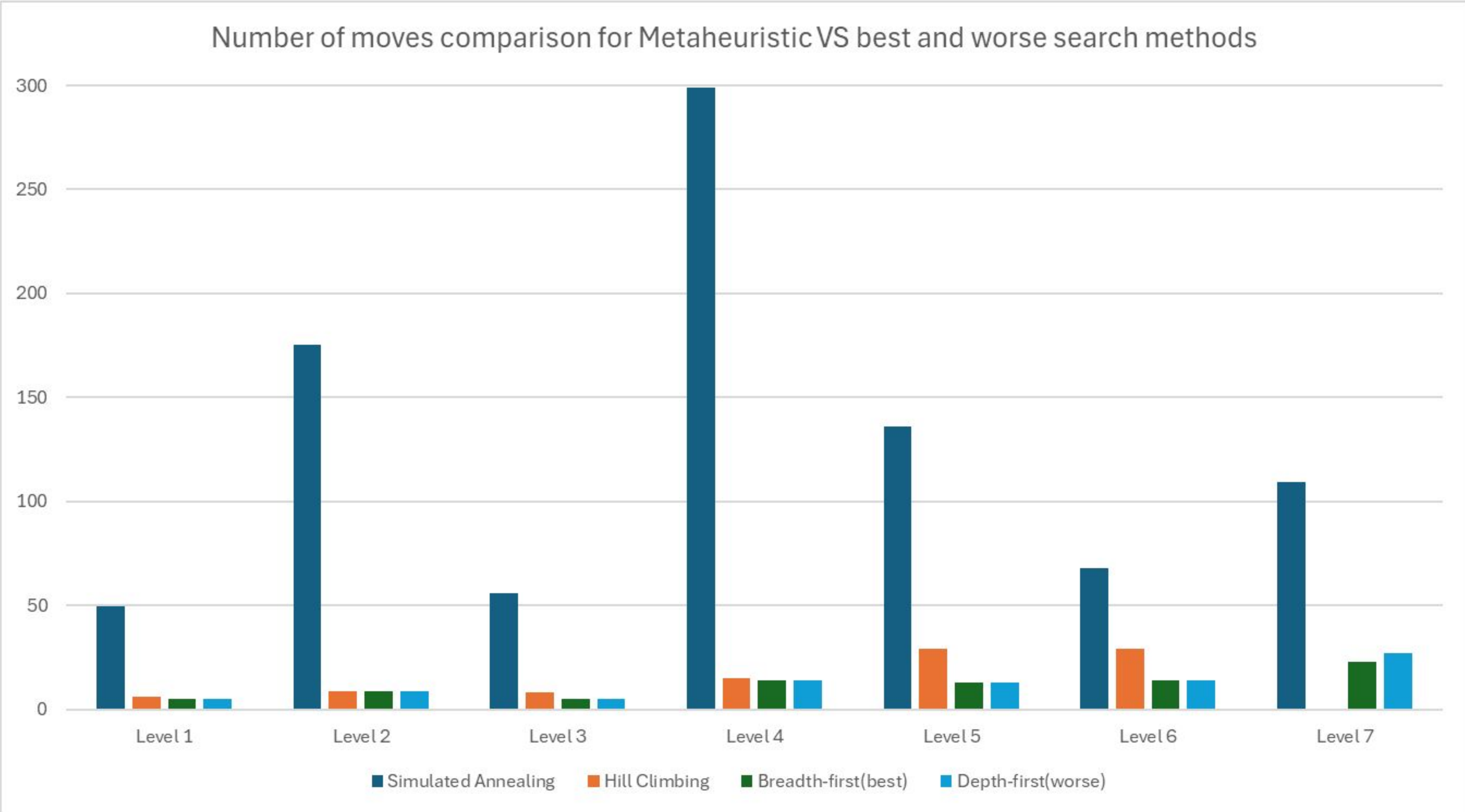
Time: Depth-First

Comparison of Metaheuristics



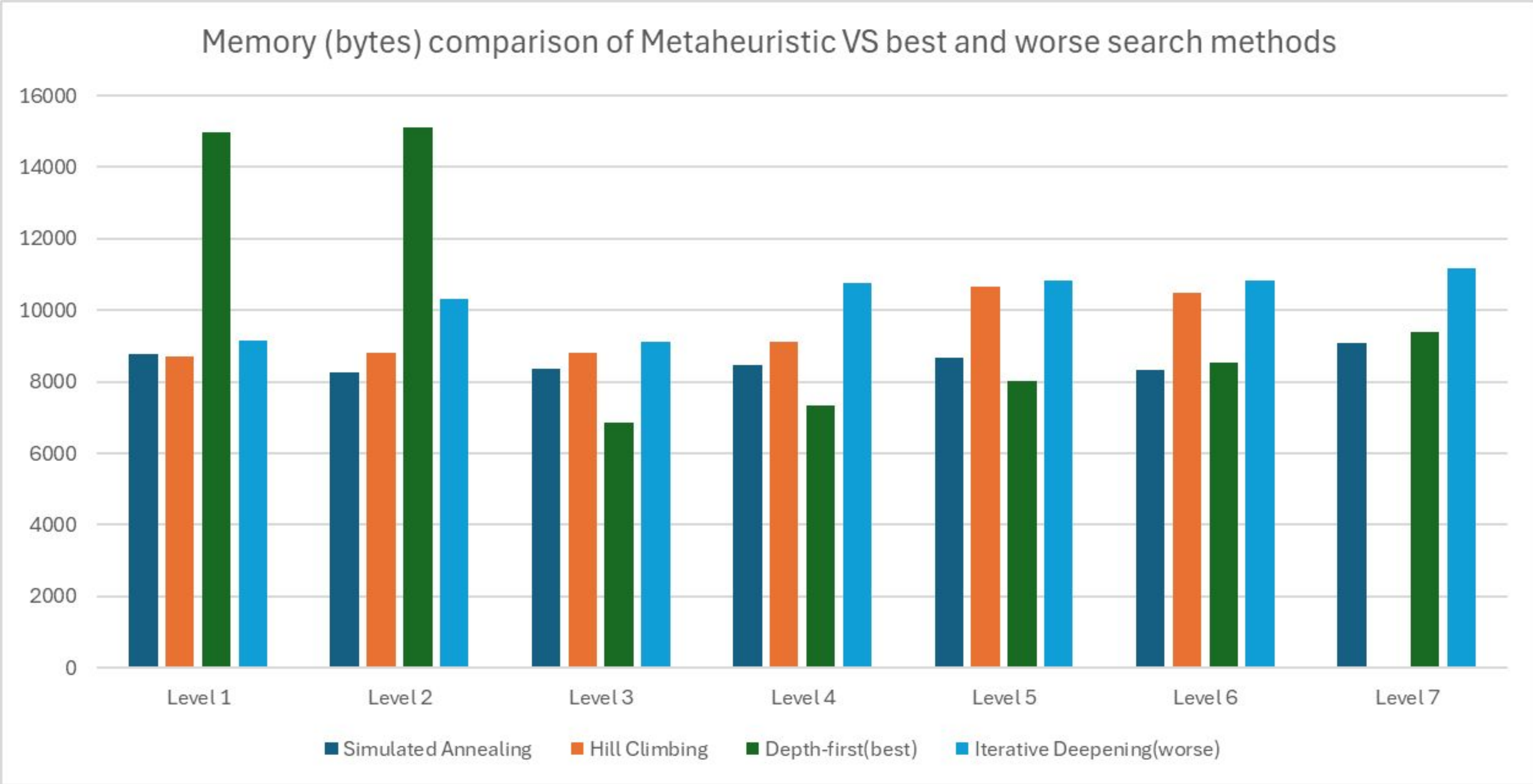
Both of them performed worse than the best and worse search methods

Comparison of Metaheuristics



Simulated Annealing is definitely the worse
Hill Climbing gets worse as the difficulty increases

Comparison of Metaheuristics



In levels 1 and 2 (easy), metaheuristics perform better
In the other levels, they have intermediate performances

Conclusion

The winner is

Memory: Depth-Limited (5)

Moves: All (except Depth First)

Time: Breath-First

The loser is

Memory: Iterative Deepening

Moves: Depth-First

Time: Depth-First

Future directions: create more adapted heuristics
Evaluate the impossible level

Thank you!