U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

CAKE SORT PUZZLE

Artificial Intelligence 2024/25
1st Assignment

Duarte Marques up202204973
Maria Vieira up202204802
Marta Cruz up202205028

Definition of the game

- Cake Sort Puzzle is a single-player puzzle game that challenges players to sort and arrange cake slices logically.
- The game is played on a board with multiple columns, where the player gradually places plates containing cake slices of different colors and styles.
- When a plate with a cake slice is placed next to another of the same type, the slices automatically merge to form a whole cake. Once a cake is fully completed, it disappears from the board, freeing up space for new plates.
- The game ends when all available plates have been placed or when the board becomes full.
- The player wins if they manage to place all plates before the board fills up.

State Representation

- Board - A matrix representing cake slices in each plate.
- Available Plates - A matrix representing the plates, with cake slices, the player must place on the board.

Initial State: Empty board and three plates with random available slices.

Objective Test: A plate on the board must have 6 slices of the same kind of cake.

```
board = [
[1,2,2],[4,5],[]   ,[5,3],
[5,3]  ,[]   ,[]   ,[]   ,
[]     ,[]   ,[2,4],[]   ,
[1,4,4],[]   ,[5,3],[]   ,
[2,5]  ,[]   ,[4,4],[]   ]

avl_plates = [
[5,5,3],[2,1,3],[2,5,2]]
```

Example of Board (4x5) and Available plates.

Operators

- **Place(x,y):**
  - **Precondition:** The position (x, y) on the board must be empty.
  - **Effect:** A plate from the available plates is placed on the board at (x, y).
  - **Cost:** Each plate placed on the board costs 1 unit.
- **OptimizePlates(plate1,plate2):**
  - **Precondition:** Both plates must be in adjacent positions on the board.
  - **Effect:** If both plates contain slices of the same cake, the slices are rearranged to maximize the number of same-cake slices per plate. Otherwise, the slices on the plates stay the same.
  - **Cost:** The cost could be proportional to the number of slices moved or simply 1 per operation.

Heuristic functions

Free Slots on the board

- Evaluates the number of available empty slots on the board.
- A higher number of empty slots provides more flexibility for rearranging cakes, while fewer empty slots indicate the player is closer to losing, as there's less room to place new plates.

Missing Slices to Complete Cakes

- Computes the total number of each type of slices needed to complete all cakes on the board.
- A lower value suggests a better board configuration.

Clustered Similar Slices

- Evaluates how well slices of the same cake are grouped together - i.e. if the slices of the same cake are in the same plate, or in plates near each other.
- A higher clustering score means fewer moves will be needed to complete cakes.

Estimated Moves to Finish a Level

- Evaluates the minimum number of moves required to achieve the target score based on current placements and available plates.
- A lower value indicates a more optimal state.

Uninformed Search

Breadth-First Search (BFS):

- Explores all possible moves at the current depth before moving to the next level;
- Guarantees finding the shortest solution path;
- Memory intensive for complex game states.

Depth-First Search (DFS):

- Explores as far as possible along each branch before backtracking;
- Less memory intensive than BFS;
- May not find the optimal solution.

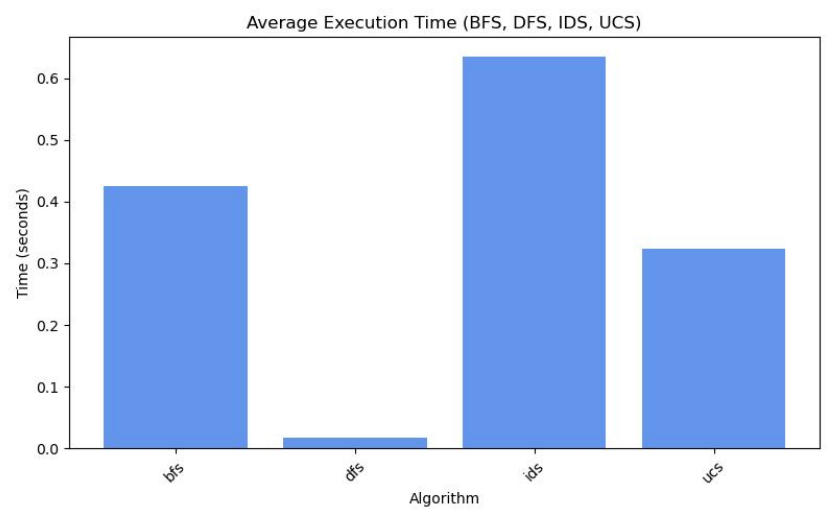
Iterative Deepening Search (IDS):

- Combines the benefits of BFS and DFS;
- Repeatedly applies DFS with increasing depth limits;
- Finds the shortest path without excessive memory usage.

Uniform Cost Search (UCS):

- Explores paths in order of their cumulative cost;
- Guarantees finding the least-cost solution;
- Similar to BFS but considers path cost.

Uninformed Search Algorithms Comparison



Based on the execution time analysis, the Depth-First Search (DFS) is the most efficient algorithm.

Informed Search

Greedy Search:

- Makes locally optimal choices based on heuristic values;
- Very fast but may not find optimal solutions;
- Uses only heuristic value h(n) for decision making;
- Memory efficient compared to BFS.

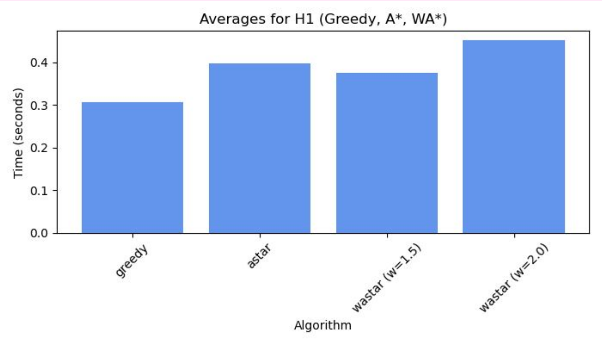
Weighted A* Search:

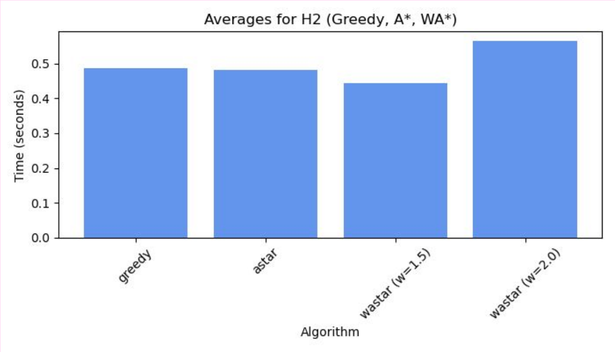
- Variant of A* that weights the heuristic component;
- Uses $f(n) = g(n) + w * h(n)$, where $w > 1$;
- Trades optimality for faster search;
- Often finds good solutions more quickly than A*;
- Useful for time-critical applications.

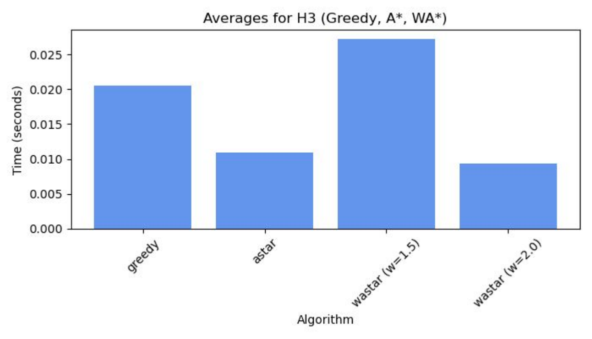
A* Search:

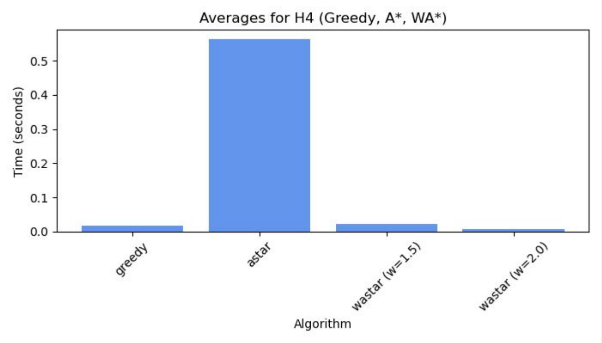
- Uses heuristic functions to guide the search;
- Combines path cost g(n) and heuristic estimate h(n);
- Efficiently finds optimal solutions when using admissible heuristics;
- Balances exploration and exploitation;
- Guarantees optimal solution if heuristic is admissible.

Informed Search Algorithms Comparison



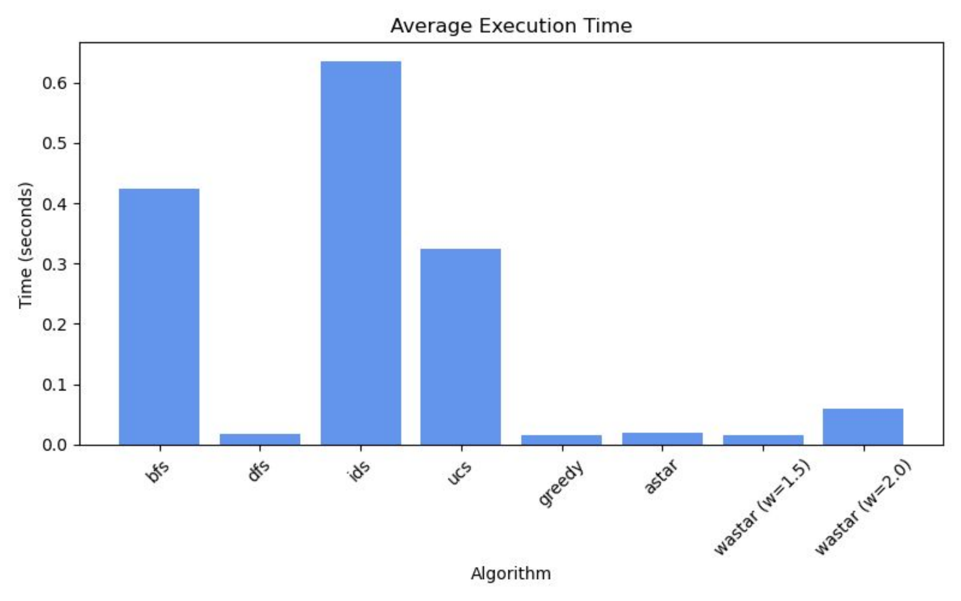






Based on the execution time analysis, heuristic 3 (clustered_slices_heuristic) was identified as the most efficient. It evaluates how well slices of the same cake are grouped together.

Uninformed vs Informed Search Algorithms Comparison



The execution time analysis shows that, among the uninformed algorithms, DFS is the most efficient, while among the informed algorithms, Greedy Search achieves the best performance.

Conclusions

- The Cake Sort Puzzle demonstrates the effectiveness of heuristic search in game AI
- Greedy algorithm with the combined implemented heuristics provided the best balance of solution quality and performance.
- Future work could explore machine learning approaches to automatically generate effective heuristics

Material Usage

- **Programming language**: Python
- **Development environment**: Visual Studio Code
- **Data Structures**: Lists of lists to represent:
  - Cake slices in each plate
  - Available Plates, with cake slices, the player must place on the board

References

- Official game url
- CMU School of Computer Science
- Materials available in Moodle