Checkpoint: Heuristic Search for Solitaire Games

Artificial Intelligence - IART 2024/2025

Artur Telo Luís
Gonçalo Joaquim Vale Remelhe
Nuno Pinho Fernandes

Specification of Work

FreeCell

- ***Description:*** 52-deck card game, where all cards are dealt face-up into 8 columns;
- ***Goal:*** Move all card to four foundation piles of the same suit and in ascending order (from Ace to King);
- ***Project Objective:*** Develop an AI-based solver using different heuristic methods;
- ***Features:*** GUI, AI-based solving mode (walkthrough), Human mode with AI hints.

Related Work And Materials Used

- Klondlike Solitaire implementation on GitHub;
- Text based Freecell-solitaire AI Solver using BFS, DFS and A* on GitHub;
- Available course resources regarding all heuristics
- PyCharm / VSCode

Problem Formulation as a Search Problem

- ***State Representation:*** Given by the Deck class, which is list of piles (tableau, free cells and foundations);
- ***Initial State:*** full tableau and free cells and foundations are empty
- ***Objective State:*** tableau and free cells are empty, foundations have sequences from ace to king in every suit.

```
initial_deck = {
  "freecells": [null, null, null, null],
  "foundations": [null, null, null, null],
  "columns": {
    "1": ["K_S", "8_D", "5_C", "2_H", "J_D", "6_S", "3_C"],
    "2": ["Q_H", "7_S", "4_D", "A_C", "10_S", "5_D", "2_S"],
    "3": ["J_C", "6_H", "3_D", "K_C", "9_S", "4_H", "A_D"],
    "4": ["10_D", "5_S", "2_D", "Q_C", "7_H", "3_S", "K_D"],
    "5": ["9_H", "4_S", "A_S", "J_S", "8_C", "6_D"],
    "6": ["8_S", "7_D", "Q_S", "10_C", "5_H", "2_C"],
    "7": ["7_C", "6_C", "K_H", "9_D", "4_C", "A_H"],
    "8": ["J_H", "10_H", "9_C", "8_H", "Q_D", "3_H"]
  }
}
```

```
final_deck = {
  "freecells": [null, null, null, null],
  "foundations": {
    "hearts": ["A_H", "2_H", "3_H", "4_H", "5_H", "6_H", "7_H", "8_H", "9_H", "10_H", "J_H", "Q_H", "K_H"],
    "diamonds": ["A_D", "2_D", "3_D", "4_D", "5_D", "6_D", "7_D", "8_D", "9_D", "10_D", "J_D", "Q_D", "K_D"],
    "clubs": ["A_C", "2_C", "3_C", "4_C", "5_C", "6_C", "7_C", "8_C", "9_C", "10_C", "J_C", "Q_C", "K_C"],
    "spades": ["A_S", "2_S", "3_S", "4_S", "5_S", "6_S", "7_S", "8_S", "9_S", "10_S", "J_S", "Q_S", "K_S"]
  },
  "columns": {
    "1": [],
    "2": [],
    "3": [],
    "4": [],
    "5": [],
    "6": [],
    "7": [],
    "8": []
  }
}
```

Problem Formulation as a Search Problem

Operators (Moves in the Game)

1. **Move Card to Free Cell**
   - Preconditions: At least one Free Cell is empty.
   - Effect: Moves a single card from the tableau to a Free Cell.
   - Cost: 1 move.
2. **Move Card to Tableau Column**
   - Preconditions: Target column must have a top card with alternating color and one rank higher than the moving card.
   - Effect: Moves one or more cards as a sequence.
   - Cost: 1 move.
3. **Move Card to Foundation**
   - Preconditions: Must be the next card in the correct suit order (Ace to King).
   - Effect: Moves the card to the foundation.
   - Cost: 1 move.
4. **Move Sequence Using Free Cells ("Supermove")**
   - Preconditions: Enough free cells exist to temporarily hold part of the sequence. Can move up to 2^NºFree Cells - 1 cards.
   - Effect: Moves multiple cards between tableau columns.
   - Cost: Scales with number of free cells used.

Heuristics

- Number of cards in foundations (higher is better).
- Number of blocked cards (lower is better).
- Free cell usage (lower usage preferred).

Approach

Uninformed Search

Depth First Search (DFS) even if it can get stuck for a long time on dead-end branches, all tested boards have a solution.

Breadth-first Search (BFS) takes a lot of memory space to visit all nodes but finds the closest solution (with lowest cost, number of moves).

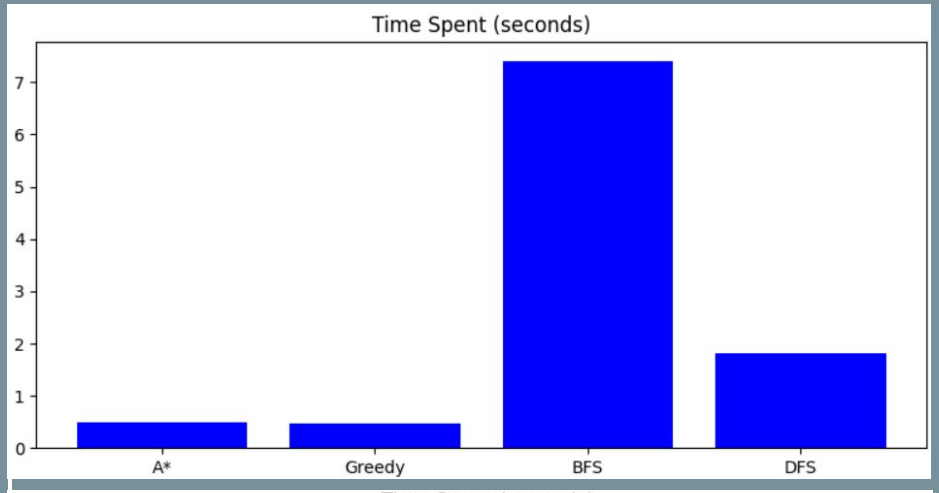
Informed Search

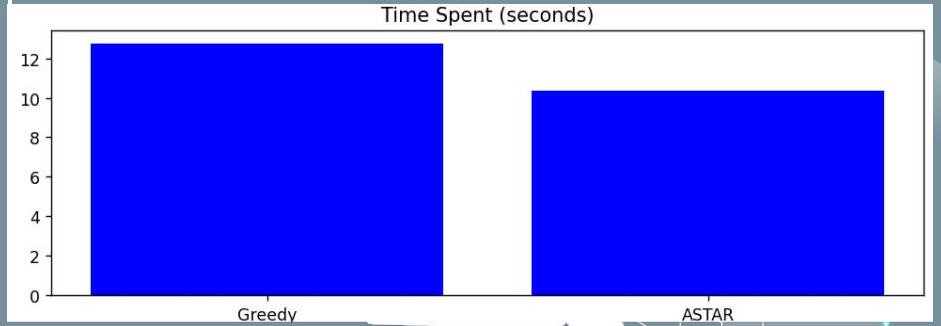
A* Algorithm, to guide the solution based on heuristic functions that give an informed degree to the research. In the game it is important to know the distance between pieces of the same color and the number of moves performed.

Greedy Algorithm makes the locally optimal choice at each step with the hope of finding a global optimum. It uses a heuristic to evaluate the desirability of each move, but unlike A*, it does not consider the cost to reach the current state. This can lead to faster solutions in some cases, but it may also get stuck in local optima and not find the best solution.

Results – Time Spent

Depth First Search (DFS) was the best uninformed method, but still very slow compared to A* or Greedy
Breadth-first Search (BFS) was the slowest due to its inefficiency by going through all states at the same level
A* was the best algorithm, even in a more challenging comparison
Greedy was still very good compared to A*, even though it was a bit slower





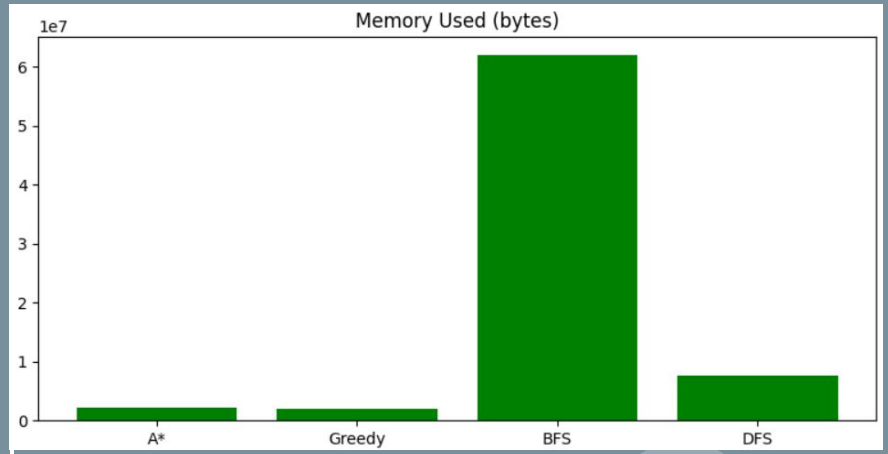
Results – Memory Consumed

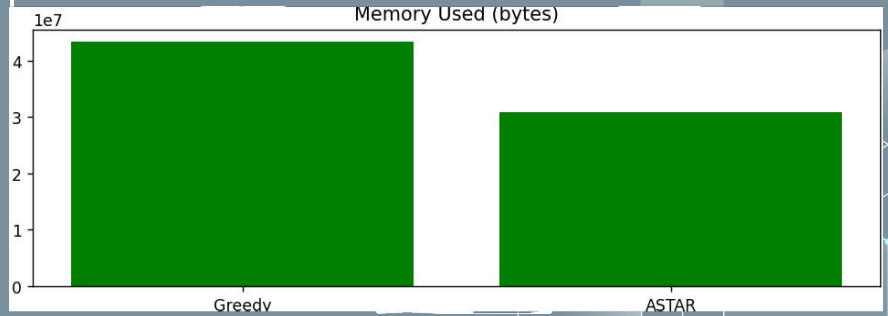
Depth First Search (DFS) was again the best uninformed method, but still quite expensive compared to A* or Greedy

Breadth-first Search (BFS) was by far the worst method for the same reasons as the previous slide

A* was the best algorithm, even in a more challenging comparison

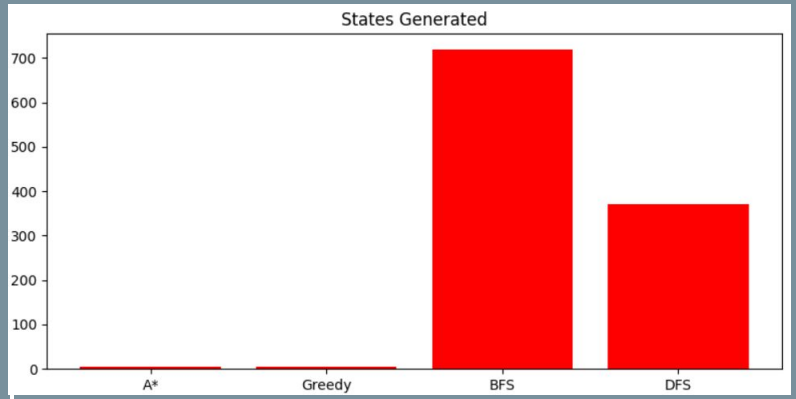
Greedy was still very good compared to A*, even though it used a bit more memory

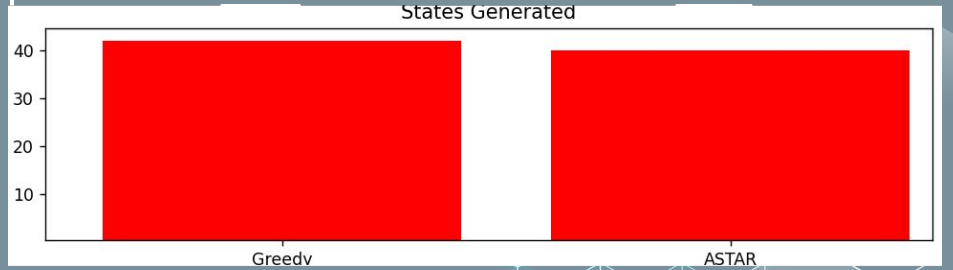




Results – States Generated

BFS and DFS had quite bad results. Overall, BFS was the worst method in any result, followed by DFS
A* and Greedy had a little difference, with A* being a little more efficient compared to Greedy





Conclusion

Efficiency

The implemented search algorithms vary in efficiency. A* and Greedy algorithms are generally more efficient due to their use of heuristics, while DFS and BFS can be less efficient due to their exhaustive search nature.

Time Spent

A* is generally the fastest in finding the optimal solution due to its heuristic guidance. Greedy is also fast but may not always find the best solution. DFS can take a long time if it explores deep branches, and BFS can be time-consuming due to the large number of states it needs to explore.

Overall, the choice of algorithm depends on the specific requirements of the problem, such as the need for the shortest solution, memory constraints, and time efficiency.