Inteligência Artificial

Diogo Gomes
Luís Seabra Lopes

Desenvolvimento de um Agente autónomo para o jogo Sokoban

Rui Fernandes, 92952

DETI
Universidade de Aveiro
December 17, 2020

1 Introduction

In this document I will explain how the agent I developed to solve the Sokoban game works. It's important to note that I used the tree_search.py script from the 2nd practical class as a base for my code. Most of the code is in there the rest being in the student.py script supplied by the professor.

2 State and Classes

As reference the state of the problem is represented as tuple composed of a set of the box positions and the single position of the keeper. So: (set(positions of boxes), keeper position), eg: ({(3,2),(4,7)}, (4,4))

The solving is divided into two parts so there are two Domain classes, Sokoban, that is responsible for the pushing of boxes, and Man, that is used for the movement of the keeper from one position to another.

I keep a single SearchProblem class, where most of the deadlock detecting code is at.

The Tree searching classes are also divided: SearchTree and SearchTreeMan.

The SearchNode class is almost identical to the original, only adding comparison between two nodes.

3 Backtracking

To impede both search trees to visit methods already visited before in the given search, I kept a set of visited states in the classes, and in case that the newstate is already in that set then it will not add it to the open nodes. To improve performance it isn't the set itself that is saved but a string version of it, I tried hashing it after converting it into a string but it slightly decreased performance instead.

4 DeadLock Detection

A very important part specific to the Sokoban problem is the dection of states that are deadlocked. In my solver most of the methods that deal with this are in the SearchProblem class, including:

- "is_dead_space" that given a position of the map will check if it is "dead", as in that if a box is on that space then there is no way to push it to a goal, this method is called for every space of the map when the SearchProblem is initialized and the dead spaces are stored in a set.
- "freeze_lock" that given the position of a box and the set of boxes will check if a freeze deadlock is present, this recursively check if adjacent boxes are push-able or locked determining if the original box is also locked.
- "square_lock" since the previous method works recursively it will generate an infinite loop if four boxes are next to each other in a square shape, this method resolves that case and is called inside "freeze_lock".

The method "is_deadlock" in the SearchTree class will call these methods for each box in a given state to check for presence of deadlocks in a new state, those states will not be added as nodes to the open nodes list. It's important to note that deadlock detection is not present at all in SearchTreeMan since the state only changes the position of the keeper so no deadlocks will be generated.

5 Domain Actions

For the Sokoban domain class, an action is a tupple of:

- The identifier, for example "pl" meaning push left
- The current position of the given box
- The position of the box after the push is performed

In this class, to determine the possible pushes on the map, I first perform a BFS using the Man domain, the current state and no goal, this will make it so it doesn't have any solution but its set of visited states will be a set of all spaces reachable by the keeper in that state. This is needed to determine if a push is possible since the keeper needs to get to the position behind a box to push it in a given direction.

As for the Man domain class, an action is a tupple of:

- The actual key equivalent to the move so "a" would be a move to the left
- The current position of the keeper
- The position of the keeper after the move

6 Algorithms and Heuristics

I only used simple algorithms learned in the early classes needed for the problem, so the SearchTree class supports greedy and A* search algorithms, and the SearchTreeMan supports Breadth First search and A*

Both domains have heuristics implemented, both are based on Manhattan Distance, for the Man the distance between the keeper and the goal position, for Sokoban the sum of distances between each box to each goal, so for 3 boxes and 3 goals the sum of 3 distances.

7 Open Nodes

A simple change to the original code made in the classes is that for the SearchTree class the open nodes list is now a Priority Queue that will give priority based on the heuristics and costs or just heuristics dependent on the chosen algorithm. This heavily improves performance when faced with sorting an increasingly bigger list.

8 Student.py

In the solver method I simply create a Sokoban Domain and Problem classes for the given map and then I choose the search algorithm based on the number of boxes on the map. I do this because the A* algorithm is much slower and under performs heavily on maps with more than 4 boxes, while we do want better solutions as much as possible, getting through a level gives a lot more points in this assignment so in these cases the much faster greedy algorithm is chosen. The only real problem are the levels above 140 for A*, while both get stuck on 146.

After the search is complete, I iterate over the actions (pushes) and for each one I do an A* search with a Man domain to get the moves needed to get from the keeper position to the position to push the box, adding all of them to the keys string and a final one correspondent to the push.

9 Possible Improvements to get past 146

One possible addition to the code would be the detection of Pi corral deadlocks, pruning the search tree earlier, but in many levels it could make the solver much slower, so there would need to be more optimization of the code in general

10 Groups I Talked With

For either helping, get help or just exchange ideas I talked with a few colleagues, namely:

- 92926 PEDRO ALEXANDRE GONÇALVES MARQUES
- 93446 LEANDRO EMANUEL SOARES ALVES DA SILVA
- 93015 DANIEL DE AMARAL GOMES
- 93195 HUGO FILIPE RIBEIRO PAIVA DE ALMEIDA
- 93248 CAROLINA SIMÕES ARAÚJO
- 93169 MARGARIDA SILVA MARTINS