Automated Sokoban player

Nina Suette

March 14, 2022

1 Overview

In the following, the course project for the course "Introduction to Artificial Intelligence" in the Winter Term 2021/22 will be documented. The project being worked on by the student Nina Suette at AAU Klagenfurt. The overall task is to implement and discuss different search algorithms to solve the Sokoban game. It is also required to discuss the deployment of AI methods in industrial applications from an ethical perspective.

In the next section the structure of the programme will is explained, followed by the third section, which discusses the functionallity. The fourth section represents the selectable search algorithms, their functionallity and attributes. In the fifth section it is documented how to run the skript and in the sixth, how to interpret the outputs. In section seven the deployment of AI methods in industrial applications are discussed from an ethical perspective. In the last section a conclusion is made.

2 Programme structure

The author were allowed to use an existing implementation of the Sokoban game from the Github repository[1] that is marked here. For using it in the own project, the author downloaded the repository and made changes and adjustments to the project. Changes were made in the whole structure of the template.

The new structure is as follows:

- levels: Different level sets reside in different directories. In each level set a different levels and different number of levels can be found. To play a different level set change the following line of code into sokoban.py file level_set = "original" to level_set = "magic_sokoban6", level_set = "small_chessboards" or to level_set = "test". Only the level set "original" will work for the implemented search algorithms.
  - magic_sokoban6
  - original
  - small_chessboards
  - test
  - level_parser.py
- themes: Also different themes are availabe to choose from. To change theme change the following line of code into sokoban.py file theme = "default" to theme = "soft", or theme = "ksokoban".
  - default
  - ksokoban
  - soft
- BFS.py: Breadth first graph search Algorithm

[1] https://github.com/kazantzakis/pySokoban

- DFS.py: Depth first modified tree search Algorithm
- DLS.py: Depth limited modified tree search Algorithm
- Environment.py: to setup the game environment
- Level.py: sets up the choosen level
- node.py: data structure constituting part of a search tree, includes parent, matrix, action, costs, plus several functions to play the game with search algorithms.
- Search.py: several functions to play the game.
- sokoban.py: the main, does the set up for choosen parameters.

3 Functionality

3.1 node.py

Node is a data structure that includes parent, matrix, action, costs, plus several functions to play the game with search algorithms. Parameter parent is the parent state of the current node, matrix is the current game matrix, action is the direction taken to generate the current node, costs are the costs to the current state and the functions are explained by the following:

expand(): this function generates all possible succesor nodes from the actual node. Fist the function checks for every direction (right, left, up, down), if the move is possible via possible_move() and possible_push(), then for all possible moves the successor nodes are generated and returned.

goal(): checks via call of function is_completed(), if goal is reached.

is_completed(): By iteratin over the game matrix, this function checks if all boxes are on their provided states, returns true if so.

next(): receives an action and returns the new coordinates based on its action.

possible_move(): checks if moving in the given direction is possible.

possible_push(): checks if pushing the box in the given direction is possible.

get_content(): returns coordinates based on given parameters.

getPlayerPosition(): retruns the players current coordinates in the game matrix.

movePlayer(): moves the player, also moves the boxes.

3.2 sokoban.py

sokoban.py is the main of this programme. It interprets the command line parameters, sets up the game environment and starts the game. As discribed in the "Programme structure" chapter, the theme and the level set can be changed in this file. The default parameters are theme = "default" and level_set = "original", the search algorithm and the level used for the game are taken from the command line parameters. After finding a solution via the choosen search algorithm, sokoban.py evaluates the taken moves and prints out the soluton twice. Once in a list of steps represented as x and y coordinates and once as moves represented as directions. Finally the solutions steps are fed in the movePlayer() function of the Search class, to show them on the game screen.

4 Search algorithms

The goal of the game is to get all the boxes into one of the spaces provided without deadlocking. Normally, this game can be controlled using the arrow keys on the keyboard. In this project, search algorithms are used that can solve this task by themselves. Breadth-first-search (BFS), Depth-first search (DFS), and Depth-limited search (DLS) were used in this project. The search algorithms are described in more detail below:

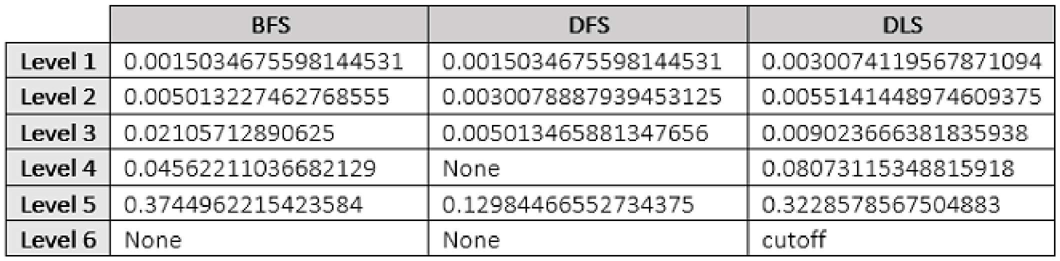
Breadth first graph search Algorithm:
Breadth first graph search (BFS) is an algorithm for traversing or searching graph data structures. It starts at some arbitrary node of a graph and explores the neighbor nodes first before moving to the next-level neighbors. BFS is complete, if the maximum branching factor is finite. It's time complexity is exponential in depth of the least-cost solution. As it keeps every node in memory, it's space complexity is also exponential in depth of the least-cost solution. This algorithm is optimal, if the step costs are all identical [1].

Depth first modified tree search Algorithm:
Depth first modified tree search (DFS) is also an algorithm for traversing or searching tree or graph data structures. It starts at some arbitrary node of a graph and explores the next-level neighbors first before moving to the neighbor nodes. DFS is only complete in finite spaces, it fails in infinite-depth spaces, spaces with loops require checks for repeated states along path. It's time complexity is terrible if the maximum depth of the state space is much larger than the depth of the least-cost solution, but if solutions are dense, may be much faster than breadth-first. It's space complexity is linear, maximum branching factor of the search tree multiplied by maximum depth of the state space. This algorithm is not optimal [1].

Depth limited modified tree search Algorithm:
Depth limited modified tree search (DLS) equals to depth first modified tree search with a depth limit. Nodes at the depth of the depth limit have no successors. The space complexity is exponential in depth limit, the space complexity is linear, maximum branching factor of the search tree multiplied by the depth limit. The DLS algorithm is also not optimal [1].

Testing the search algorithms:

Timed results of all three algorithms for all six levels. "None" means no solution or no solution in appropriate time. Cutoff means that a cutoff occurred, depth limit of 6 has been exceeded.



	BFS	DFS	DLS
Level 1	0.0015034675598144531	0.0015034675598144531	0.0030074119567871094
Level 2	0.005013227462768555	0.0030078887939453125	0.0055141448974609375
Level 3	0.02105712890625	0.005013465881347656	0.009023666381835938
Level 4	0.04562211036682129	None	0.08073115348815918
Level 5	0.3744962215423584	0.12984466552734375	0.3228578567504883
Level 6	None	None	cutoff

5 Running the script

To run the script with a fresh installation you may install pygame and argparse first:

```
pip install pygame
```

```
pip install argparse
```

If you run the script from command line you have to choose between the three search algorithms.

```
python sokoban.py bfs 1
```

Instead of 'bfs' you can also choose 'dfs' and 'dls' for another search algorithm.
Instead of 1 you can also choose any other level form 1-6.

6 Interpreting the outputs

The output is a list of moves represented in x and y coordinates and a list of moves which tells the direction of the taken movements to get the solution. Additional the time it takes to find the result is printed in the third line.

Example output of level 2 with BFS:

```
[[1, 0], [1, 0], [0, 1], [1, 0]]
['Move Right', 'Move Right', 'Move Down', 'Move Right']
0.005013227462768555
```

7 Deployment of AI methods in industrial applications from an ethical perspective

The name "Sokoban", which is Japanese, means "warehouse keeper" when translated into English. So this game recreates an overcrowded warehouse, in which the warehouse keeper has to bring the boxes to be stored to the designated free spaces without getting into a deadlock. If you now do some thinking about translating this game into the real world, numerous questions about ethical aspects arise.
In industry, similar robots with artificial intelligence can and are already being used to relieve people of monotonous work, such as working on the assembly line or storing goods in a warehouse. Agents equipped with artificial intelligence can represent both autonomous agents and team-mates. Considerations about ethics in the field of artificial intelligence raise big questions. In the case of agents with artificial intelligence, who makes decisions? What are the moral consequences of these decisions? What are the legal and social consequences of this? Can a system with artificial intelligence be held liable for "its" decisions? If not, then who [2]?
Before thinking through all these questions in detail, it must be kept in mind that these agents that have artificial intelligence, get all their algorithms, data, morality, construction, design, etc. from their designers, who are humans [1]. First of all, the question of who makes decisions and who is responsible for them must be clarified. As described by Coeckelbergh [3], an agent with artificial intelligence can make decisions and carry out actions that may well have ethical consequences, but these agents do not know what they are doing. Agents of this type cannot think and, above all, have no moral thinking, so they cannot be held morally responsible for their decisions and actions. Thus, the action of the agent can be viewed similarly to the action of children and pets, who are not held responsible for their own actions either, but their caretakers. It can be said that although the task and action are handed over to the agent, the responsibility remains with the client [3].

However, not all questions have been answered here. New ambiguities arise from the above statement. When creating an agent with artificial intelligence, countless people are often involved. Who is to blame? Who is liable if the agent once served a purpose other than the one it now serves? The more one thinks about the huge number of stakeholders involved in such a system, the clearer it becomes that responsibility cannot be delegated to a single person. More than one person is responsible for this. Just thinking about the various components of an AI system, for example, programmers, parts manufacturers, software, hardware, sensors, etc. are components that play a crucial role [3].
According to Coeckelbergh [3], responsibility means answerability and explainability. From this it becomes clear that an AI agent cannot be responsible for his actions. Although he can document his actions, he cannot explain why he performed this action. Even if it may seem so at first glance, even programmers and users of the agent often do not know how to explain how the agent came up with its action. This can be attributed to the use of deep learning and neural networks, where the programmer knows how the AI system works, but does not know exactly how the agent decided to act. This problem is called the black box problem and represents problems in the area of transparency and explainability [3]. Opening this black box can be a great advantage not only for transparency and explainability, but also for further development.

If these ethical aspects are now considered for the area of "use of AI methods in industrial applications", the following can be seen. When using artificial intelligence, it is difficult to transfer responsibility to one person due to the many different components involved in a robot, which in industry is responsible for storage in warehouses, for example. Many different programmers are involved in the software creation process, and different component manufacturers are responsible for providing the individual components, such as sensors or other hardware. Therefore, responsibility in this area cannot simply be delegated to a specific person or split up to many of them. Since an agent in industrial applications can also work using deep learning and neural networks, the problem of transparency can also arise here, which makes assigning responsibility even more difficult. Since artificial intelligence applications involve countless people and factors, it becomes almost impossible to assign responsibility.

8 Conclusion

Finally one can draw the conclusion from this project that solving a game like Sokoban or other similar games is possible by implementing different search algorithms. The algorithms selected in this project are uninformed search algorithms and manage to win simple levels, but better performance can be achieved with other algorithms, such as informed search algorithms. Solving these games is also possible by means of logic.

References

[1] G. Friedrich, (2022, March 12). Uninformed Search. [Online]. Available: https://moodle.aau.at/pluginfile.php/1210756/mod_resource/content/3/search_211004.pdf

[2] V. Dignum. (2018 February, 13). Ethics in artificial intelligence: introduction to the special issue. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s10676-018-9450-z.pdf

[3] M. Coeckelbergh. AI ethics / Mark Coeckelbergh. Cambridge, MA: The MIT Press, 2020.