SOKOBAN ASSIGNMENT REPORT

Intelligent Search – Motion Planning in a Warehouse

Student Name: Timothy Jeoung, Naga Kaushik Jandhayala, Sooraj Chirangara Narayanan

Student ID: n10887601, n10660381, n10569294

Lecturer: Frederic Maire

Tutor: Will Browne

Introduction:

The Sokoban puzzle is complicated, even though the rules seem straightforward because even a tiny warehouse requires many calculations to solve the problem. Thus, the game of Sokoban has a vast search space with few goals located deep in the search tree. Nevertheless, the admissible heuristics can be applied easily, such as Manhattan distance in a search algorithm of informed search approach using A*. Here a weighted variant of normal classical Sokoban is considered for analysis. In this Sokoban variant, each box has individual pushing costs, and the goal is to move the boxes into target locations with minimum cost. This report will explain a clear state of our representations, heuristics, and other essential features to understand the solver. Also, it will show the testing methodology of the codes and describe the performance and limitations of the solver.

State representation

Our state representation for the Sokoban game is a state-space search problem. The level of the state is defined by the position of the player and each box. The game of Sokoban can be seen as a transition from a state to another state. Therefore, each level of the game will need its own state graph. The graph is directed as a movement can be irreversible and different moves can lead to an identical state so the graph could have cycles.

Thus, a state-space search is represented as following tuple **S: (S, A, Action(s), Result (s, a), Cost (s, a),** where '*S'* is the possible states, '*A'* is possible action and '*Actions(s)'* is a function telling which action is available to perform in a certain state. Moreover, '*Result (s, a)' r*eturns the state of action *'a'* in state *'s'*. Lastly, *'path_cost (s, a)'* will be the action cost of *'a'* in a state *'s'*.

To solve Sokoban problems, it is significant to find a path from the initial state to a goal state. This can be done by collecting all the pushes that have been executed with the paths and other inserted moves in between. It's done by searching the path in the recent state from the last position of the player to a position where a next push is required. In our case, we have mainly used the heuristic search called A* search graph algorithms to find a path in the game space.

Informed search algorithms

A* search is best-first graph search, represented as f (n) = g (n) +h (n). A graph search algorithm expands the nodes by memorizing the visited or explored nodes. In our Sokoban game case, referring to the A* equation, the **g(n)** is the cost of a worker moving from the initial state to the current state, which is the sum of all the worker movements from the first move. The heuristic function **h(n)** finds the shortest path between the initial and goal states. It is not an actual cost but is an estimated cost to reach the final state. It was important to make sure that there was no overestimation of the cost.

Heuristics

(Manhattan distance)

The heuristic used is to estimate the minimum number of pushes required to solve the Sokoban problems. In our case, we are only interested calculating the distance between each current state of the box to the goal state. Therefore, we have used the 'Manhattan distance' as an admissible heuristic that never overestimates the distance to the goal that finds the shortest path.

$$d_1((A_x, A_y), (B_x, B_y)) = |A_x - B_x| + |A_y - B_y|$$

heuristic(n) sub-function in worker_path function makes the worker take the shortest number of moves inside the warehouse to push the box.

Path Cost

A dictionary is used for the location and weight of the box. For each location action, we will record the new position of the box and its corresponding weight to maintain the consistency of all the boxes' weight throughout the program. This dictionary will be used for computing the path cost. For each movement, the cost for moving worker and the cost for moving boxes will be computed and added together to get the total cost 'C' for the solution.

Testing methodology

Test 1 has been run to output the analysed time and cost of Best First Graph Search and Best First Tree Search in warehouse 8 & 11:

Table 1:

Informed search algorithm	*Warehouse 8a Time/cost*	*Warehouse 11 Time/cost*	*Warehouse 8b Time/cost*
*Best First **Graph** Search*	*0.204003 / 715*	*0.077725 / 90*	*0.180505 /29*
*Best First **Tree** Search*	*Not responding due to repeating nodes*	*Not responding due to repeating nodes*	*Not responding due to repeating nodes*

Test 2 has been run to output the analysed time and cost of A*Graph Search and A* Tree Search in warehouse 8 & 11:

Table 2:

Informed Heuristic Search	*Warehouse 8a Time/cost*	*Warehouse 11 Time/cost*
*A** ***Graph*** *Search*	*0.678026/720*	*0.257427/82*
*A** ***Tree*** *Search*	*2.033852 / 720*	*0.705812 / 82*

Referring to table 1 & 2, it clearly shows that graph searches are more efficient than tree searches with shorter analysing times. This is because graph search does not visit or expand the nodes that already has been visited or expanded. Oppositely, tree search can visit or expand the node multiple times. This brings to producing a tree that may contain the identical nodes several times. Therefore, the tree search takes much more time to analyse than graph search.

Furthermore, best first graph search has slightly shorter time than A* graph search. However, A* Graph search is more suitable because best first graph search is not complete and not optimal because it can be polynomial due to complexity of time and space. Though, A* is complete as it uses an admissible heuristic function which means it will always find the optimal path between the starting point and the goal without overestimating. Therefore, best first graph search can find the cheapest path to the goal with over estimating but A* Graph search finds the cheapest path to the goal without overestimating due to the admissible heuristic function.

Performance & Limitation

During running the A* Graph search heuristic search, some of the warehouses has an error which won't let the program to run. This can be reason of certain warehouse search spaces being extremely huge which can lead to A* to become impractical.

Moreover, there is a limitation of our codes in calculations it is outputting a higher cost than expected cost from the lecturer. Furthermore, another limitation is that the heavier box is moving to the furthest goal and the lightest box is moving to the nearest goal which needs to be opposite to be efficient. However, if these limitations can be solved the A* graph search using Manhattan distance would be the right solution as it guarantees to find a path between starting state and the goal state without any overestimation which is optimal.

Conclusion

In conclusion, the suitable algorithm for Sokoban Puzzle game will be using an informed search algorithm called A* Graph search using Manhattan distance for heuristic. The tree search is not efficient as it will take more time than the graph searches due to revisiting and expanding the nodes. Thus, the best first graph search seems to be the appropriate solution, but A* graph search is better a solution as it's a complete and optimal search that uses an admissible heuristic function h (n) which doesn’t overestimate the cost. However, there are several improvements required such as improving the searching speed and allowing to search more space to avoid impractical of A* search. Therefore, more research is required to find specific search enhancements in the future. Furthermore, the limitation of the codes for calculation and the heaviest box moving to the furthest goal issue will require more study to develop an efficient code that can output a correct path cost and move the boxes by their weights efficiently.

Reference List

- *Norvig, P., & Russell, S. J. (2019, November).* Artificial Intelligence: A Modern Approach (3rd edition). *https://faculty.psau.edu.sa/filedownload/doc-7-pdf-a154ffbcec538a4161a406abf62f5b76-original.pdf*
- *Botea, A.; Muller, M.; and Schaeffer, J. 2002.* Using abstrac- ¨ tion for planning in sokoban. In International Conference on Computers and Games*, 360–375. Springer.*
- *Virkkala, T. 2011.* Solving sokoban*. Master’s thesis, University Of Helsinki.*
- *Junghanns, A., and Schaeffer, J. 2001.* Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence 129(1):219–251.*