CAB320: Sokoban Assignment

Queensland University of Technology (QUT)

3rd May 2022

Name	Student Number	Contribution (%)
Zac Wolter	n10471227	40
Max Spokes	n10468439	30
Sebastian Poh	n10524304	30

Table of Contents

Introduction

Sokoban solver is a widely referred to problem within the AI domain. Its complexity requires an equally intricate path planning agent to be developed. This report will detail the process taken to develop such an algorithm to satisfy the problem space. The warehouse will be evaluated to determine taboo cells along with the resultant actions to identify illegal moves. Finally, the A* algorithm will be employed with the functions developed to solve the weighted Sokoban problem. Once implemented, detailed testing will be conducted to fix bugs where possible. Concluding testing, a strong understanding of the projects performance and limitation will be known and outlined.

State Representation

To store and track static and dynamic aspects of the warehouse environment and conditions efficiently, the SokobanPuzzle class contains two separate stores of information regarding the warehouse. The first is a static representation of the warehouse (named “problem” in the class’ initialisation function), containing information regarding the warehouse walls, targets, box weights and other information that would remain static throughout the puzzle. The second is a state representation of the dynamic aspects of the warehouse – this includes the worker and box locations, as at least one of these two pieces of information changes with each iteration of the A* algorithm.

Admissible Heuristic Formation

The key to a successful path planning algorithm for the Sokoban puzzle is largely dependent on the heuristic. A greedy heuristic is to be avoided because it will lead to inefficient guidance and in the worst case, a heuristic that does not direct the path planning toward its goal. To avoid a greedy heuristic, all factors of the problem space have to be taken into consideration. Simply taking the boxes and target locations into account will lead to the heuristic ignoring the workers location and weighting of the boxes. As such, for a good admissible heuristic it is crucial to first understand the requirements of the problem.

An admissible heuristic is one that returns the true cost of an action in the long-term. Short-term cost includes the step size – one grid space – and the weight of the box being pushed, if applicable. This short-term cost has been implemented in the cost path_cost method which summates the workers movements and weight of the box being pushed, if applicable. With the workers short-term considered, the long-term goal must be implemented to the path-planning to ensure an optimal route is found. The heuristic must take into consideration:

- The worker's distance to the heaviest box
- The heaviest box's Manhattan distance to the closest unclaimed target

The value() method within the SokobanPuzzle class implements these two requirements to form the admissible heuristic utilised. At each state, the standard path cost is calculated by summating the action – 1 – and the weight of the box – 0 to 99. Additionally, the heuristic is calculated by firstly assigning the heaviest boxes with the closest targets entirely. After the Manhattan distance is calculated for each box-target pair for the problem space and added to the path cost. For Warehouse 08a in its initial state, seen in Figure 1, the path cost can be calculated as 14.

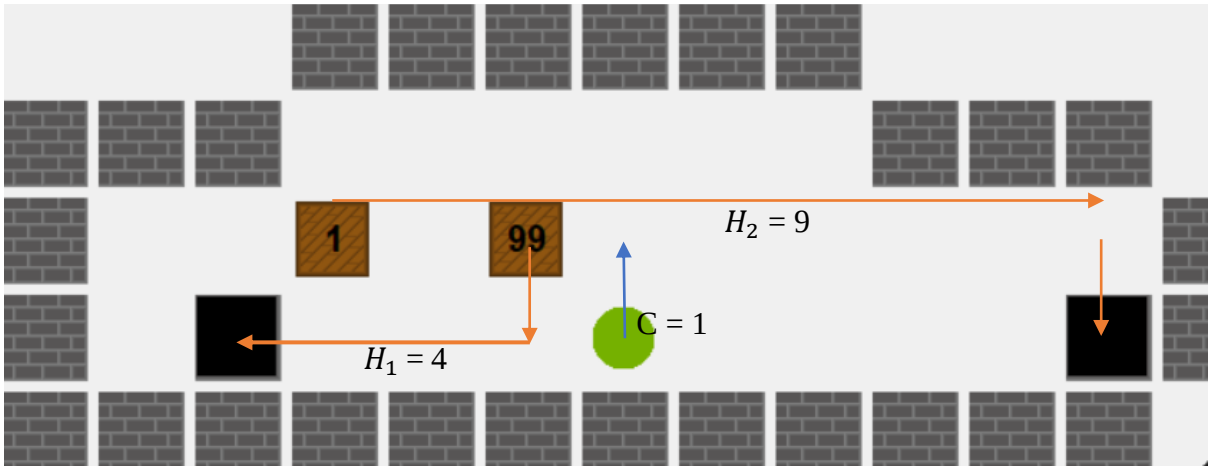

Figure 1 Warehouse 08a path cost illustration

$$total\ cost\ (F) = distance(worker \rightarrow nearest\ box) + \sum_{n=smallest\ weight}^{largest\ weight} distance(box \rightarrow nearest\ unclaimed\ target)$$

$$= 1 + (4 + 9)$$

$$= 14$$

Testing Methodology

The testing methodology that the algorithm developed for this assignment involved firstly designing a solution around unique problem cases that are prone to cause errors. These began in identifying potential causes of error that would fail the requirements set in the assessment brief. Such as the taboo cell identification process. Hence compensating for certain warehouse layouts provided in the assessment material provided that might cause issues was necessary. Particularly with layouts that had spaces outside the working area (not contained within the walls) that result in a wrong taboo-cell identification. And the consideration of certain combination of weights in a particular question (e.g if all weights are equal/same). Therefore, by considering such cases, the robustness of the code/solution fit in line with the requirements and full scope set for the assignment. This was fully tested using the check_elem_seq function. Supplementing said script with the known solutions to each puzzle provided by the assessment brief. To see if the algorithm developed produced the same set of actions for an optimal solution. After these first set of tests, additional considerations such as if the worker moved into a wall, moved a box into a wall or another box into a box were tested. And if any of these occurred the test would be failed. Additionally, comparing the costs calculated from the algorithm with the cost from the FAQ was a method we tested to determine the effectiveness of our proposed method/solution.

Performance and Limitations

To measure the quantitative performance of the Sokoban Solver program, several warehouse tests were run, measuring the time taken to plan the most efficient route, the cost of the produced route and the accuracy of the route compared to some of the examples received in the task sheet (note: not all warehouses tested were in the task sheet, so accuracy scores are unavailable for some tests). Below in table 1 is a layout of program test results on several warehouses.

Table 1 Solver Program Test Results

Warehouse No.	Solving Time (s)	Path Cost	Cost Accuracy (%)	Path Accuracy (%)
07	248.162	26	100	100*
01	0.007	33	N/A	N/A
09	0.006	396	100	100
81	0.394	376	100	100
5n (impossible)	1.944	None	None	None
3_impossible	0.021	None	N/A	N/A
47	0.174	179	100	100

*Regarding warehouse_07, the path produced was not the same as the task sheet path, however, the same path cost was produced – this is due to the spacious nature of warehouse_07, giving multiple efficient paths.

It was concluded that the performance of the solver program was adequate and met the specifications outlined for this task, as the solving times were short or close to the specified calculation times in the task sheet and cost/path accuracy was always matched exactly.

Regarding limitations and improvements that could be made to the solver, there is potential for optimisation in the heuristic function – further code could be added to ensure that when

calculating the worker's distance to the nearest box it doesn't include boxes that are already on targets, thus potentially reducing the number of nodes explored by the A* graph search algorithm. Another improvement that could be made (which is outside of the scope of this unit) would be the incorporation of parallelism in the search algorithm, which would drastically reduce computation time whilst utilising more of the CPU when performing calculations and exploring the search graph.

Conclusion

In conclusion, applying informed search theory was crucial to the success of the path planning agent being optimised. The admissible heuristic takes into consideration the complete problem space and requirements to result in a consistent optimal solution at each warehouse. Testing identified numerous shortfalls in the code which were rectified to increase robustness. Additionally, it allowed for the identification of limitations within the code which given more time may have been rectified to provide slight theoretical performance boosts. Overall, it was found that the path planning agent was constructed in a consistent and efficient way which resulted in the Sokoban puzzle being able to be solved.