Section 1: Formulation of Sokoban Puzzle Solver

The formulation of the Sokoban Puzzle solver is as follows:

INITIAL STATE: The Sokoban puzzle problem class constructor initialises all elements of a warehouse instance - walls, goals, boxes, worker and box weights. Of all the elements that are initialised to a warehouse instance in the problem class constructor, only workers and boxes are initialised as an initial state given that these are the dynamic elements that will be updated throughout the search problem corresponding to the result of an action.

GOAL STATE: The goal state represents one in which each box is mapped to a target such that all the target locations are occupied, and no boxes are left idle on the non-target cells.

ACTIONS: Four absolute movement actions are available to the worker for a given state *s*, defined relative to the viewpoint of the worker - Up, Down, Right, Left. The a*ctions()* function returns the available legal actions for the input state as per the current locations of the dynamic elements. An action implies an agent move which is only legal if: (1) the adjacent cell to the agent is empty; (2) the adjacent cell contains a box, and there is an empty cell on the other side of the box.

TRANSITION MODEL: The *result()* function maps a state and action to the resulting state. Given the input action is legal, the adjacent cell can either contain nothing or a box. For empty cell, only reflect the new worker position in the resulting state. Otherwise, it is a push action which requires an update of the box and worker location.

ACTION COST: The Sokoban solver has its own unique implementation of calculating the action cost, which represents the incurred accumulated cost to the current state. Instead of a constant action cost of 1, the action cost is multiplied by the box weights if the action changes the box state from a push. Pushing a weightless box incurs the same cost as a pure move action to an empty cell. In other words, an action will cost 1 regardless of the type of action (push or move) if the boxes weigh nothing in the first place.

HEURISTIC COST:

While our focus is on achieving the cost admissibility by keeping the estimated cost under or equal to the true cost), the heuristic function is designed to compute a cost dominant enough to reflect true cost in order to distinguish the optimal solutions from the nonoptimal, as to minimise the number of nodes to be expanded. This is achieved through separating the function into two parts:

1. **Worker to Box**: First, calculate the distance of current mover location to the the furthest box of boxes that are not on the target. Consider the following scenario where both boxes are closer to the neighbouring boxes than the worker:

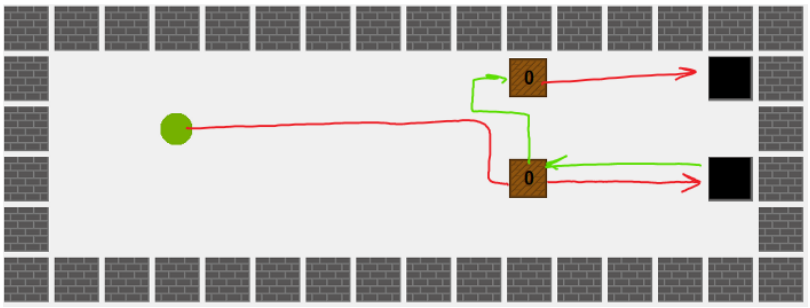

Figure 1*: The optimal path is shown in the diagram. Green is the backtracking distance.*

Maximum worker-to-box distance is used instead of the total distance as it keeps the cost admissible by disregarding the backtracking distance from the first target to the second box. Cost will surpass the true cost if both box distances are accounted for as the green arrow (see Figure 1) shows that the path could be smaller than distance between the current mover to the other box.

2. **Box to Target**: Lastly, compute the sum of the weighted distance of each box to its closest target using weight as a multiplier. The weight consideration is important as it derives a more accurate cost to the true cost.

Lastly, admissibility is achieved through relaxing the problem by removing the consideration of move validity and other existing constraints such as walls and adjacent boxes. As such, manhattan distance is used to compute the estimated path cost for two reasons: (1) it closely resembles the agent's movements on the search space as in reality each move can only bring the agent one tile closer to the goal, and (2) it is admissible as it relaxes the wall constraints.

Section 2: Testing Methodology

Testing Warehouses

To validate the performance of the code, several tests were conducted to explore the Sokoban Solver's solution in various scenarios. Initially, the warehouses provided were utilised to ensure the solver had performed correctly. Below are the results of several warehouses of varying complexities. For each warehouse, 5 tests had been conducted to compile the average analysis time and standard deviation.

Table 1: *Test Results (Mean ± Standard Deviation, Cost) of Sokoban Solver on Warehouses*

Warehouse	Analysis Time (Seconds)	Cost
warehouse_91	20.13783 ± 0.57612	45
warehouse_103	58.02767 ± 2.72125	35
warehouse_09	0.01397 ± 0.00193	396
warehouse_47	0.52341 ± 0.03115	179
warehouse_81	6.83991 ± 0.58124	376

Impact of Weighted Boxes on Path Creation

It is notable that the solver was able to incorporate the weight allocations and alter the pathing in the case of different box weights to determine a path with the least cost and was observed in the difference of moves between warehouse_8a and warehouse_8b.

To test the solver's ability to incorporate the weighted boxes, a custom warehouse was created in order to test whether the solver would resort to moving an existing box on target square in order to create the most efficient path (Figure 2), the test was successful.

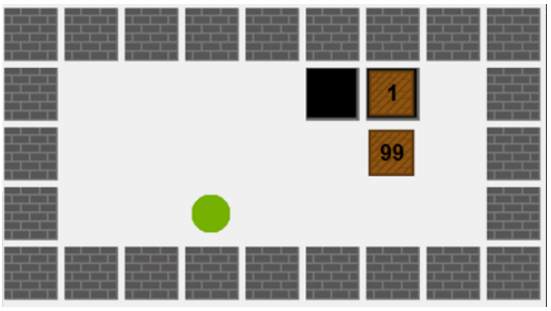

Figure 2: Initial state of *custom warehouse to test the movement of box on target square.*

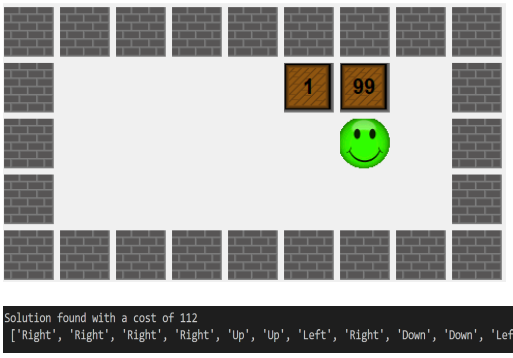

Figure 3: *Using A* search the cost is 112*

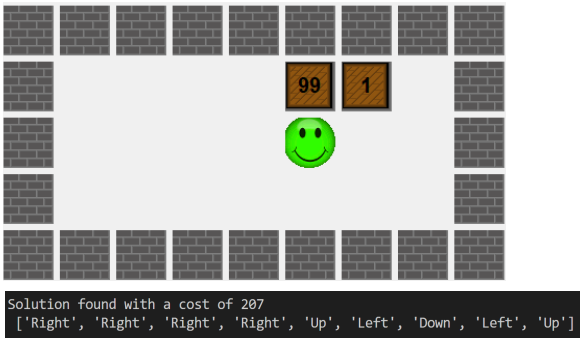

Figure 4: *Using breadth_best_first_search the cost is 207*

Impossible Warehouse Case

To test the solver's ability to recognise impossible warehouses, the warehouse_3_impossible (Figure 5) was also tested and resulted in the correct output of no possible solution.

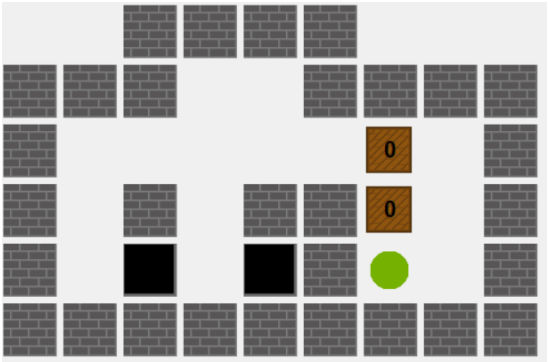

Figure 5: *Impossible Warehouse*

Section 3: Performance and Limitations

Impact of Warehouse Space on Solver Performance

It is observed that the number of moves has a minimal impact on analysis time, indicating that search space to be a more significant factor and limitation to the analysis time due to a larger number of possible path permutations. To test this, a simple warehouse (Figure 6) with a large space was created, resulting in a longer analysis time despite an easy solution. The solver utilising the A-Star Graph Search had taken 401 seconds to create the optimal path of 44 moves with a cost of 1146. The solver utilising uniform cost search had taken 1704 seconds with the same number of moves, resulting in a cost of 1146. This leads to further investigation of the search algorithm.

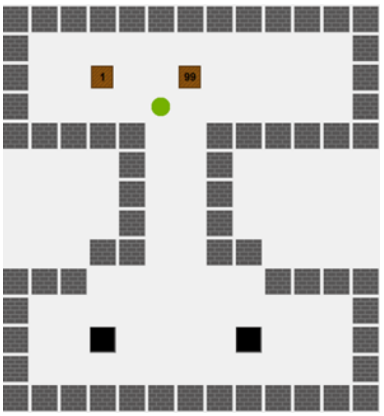

Figure 6: *Custom warehouse to examine the impact of warehouse space on analysis time.*

Search Algorithm Analysis

To investigate the impact of the Uniform Cost Search and A-Star Graph Search, the algorithms had been implemented on five warehouses for comparison purposes. 5 tests had been conducted to determine the average computation time and standard deviation.

Table 2: *Comparison Metrics (Mean ± Standard Deviation) of Search Algorithms on Warehouses*

Warehouse	Analysis Time (Seconds)	
	Uniform Cost Search	A-Star Graph Search
warehouse_01	0.01564 ± 0.00177	0.02211 ± 0.00411
warehouse_61	111.48467 ± 1.04395	176.23764 ± 8.85342
warehouse_19	0.05971 ± 0.00332	0.06743 ± 0.00190
warehouse_8a	10.42494 ± 1.58274	1.74583 ± 1.00386
warehouse_9	0.05756 ± 0.00167	0.03533 ± 0.00696
warehouse_23	0.04615 ± 0.00562	0.04928 ± 0.00449
warehouse_43	41.73204 ± 4.91833	70.63590 ± 7.61590

It is observed that the uniform cost search is more efficient in analysis for the solver in particular scenarios where directions are restricted. For instance, with warehouse_61, a narrow pathway results in less directions to explore, decreasing the analysis time, whereas the A-Star Graph Search algorithm will compute the heuristic function at each node, increasing the overall analysis time. However, in certain stages where weights are introduced and the warehouse space is larger, such as warehouse 8a and warehouse 9, the A-Star Graph Search algorithm is more efficient, it will estimate the cost given a particular direction and find the least cost path. In contrast, while the uniform cost search algorithm will also find the path with the least cost, it will explore every direction in a larger warehouse space, increasing the overall analysis time.