**Sokoban Assignment**

Intelligent Search – Motion Planning in a Warehouse

Team member:

1. Cedric Oliveira da Silva Costa: 11369574
2. Abhishek Sapkal: 10831908

Unit: Artificial Intelligence

# **Introduction:**

The Sokoban puzzle is a computer game that allows players to push boxes around the maze to place them correctly in suitable locations. This game is based on a warehouse scenario consisting of a worker, walls, boxes, and a goal location. This assignment is used for an automated planning problem for AI robots through the interpolation of artificial intelligence techniques.

Calendar

Description automatically generated

***Fig1: GUI of Sokoban puzzle***

# **State representation:**

The puzzle contains field objects including a worker, walls, boxes, and targets. The basic solution layout is to use the current positions of the above objects to perform a series of actions that can be used to complete the puzzle. Moreover, an array of actions which consists of UP, DOWN, RIGHT, and LEFT act as the nodes in our search problem.

We use the standard coordinate system to represent the field objects with special characters and consider the top left corner (0,0) to calculate the actions around the puzzle. For instance: an upward movement is -1 and vice-versa and right action is +1 and vice-versa.

Chart, scatter chart

Description automatically generated

***Fig2: Representation of field objects***

The coordinates data is saved in the list of tuples. They contain the x and y coordinates.

The next step is to find the current position of the worker and determine the next best possible action available. This can be done by calculating the proximity of the worker to walls and boxes. It is to be done in such a way that we don't put the worker in a Taboo cell. The criteria to find taboo cells are defined below:

1. If the cell coordinate is a corner, it is a taboo cell
2. If the coordinate is not a corner but is next to a wall, it is a taboo cell

We determine our actions by examine our position one step ahead. Is it a taboo cell or a cell we can move forward in achieving the set of actions to our solution? For every action, the coordinates are passed to the taboo function. By doing this we improve the time efficiency. Once we get our actions that are stored in nodes, can be used in our heuristic search algorithm

Next, we calculate the path cost using path\_cost() function. Cost of every movement is 1, but the total path cost depends on the weight of the boxes as well. The weight of the box gets added to each movement.

# **Heuristic search:**

We have used the A\* heuristic search algorithm for this solution. A\* is a popular search algorithm and it considers starting node distance(g) and heuristic distance to the solution(h). There is an alternative algorithm (IDA – Iterative Deepening A\*) which consumes less memory, but it visits the same state more than once. A\* algorithm skips traversing all the possible states which use Breadth-first search or Depth-first search. Instead, a priority queue (FIFO) is used to fetch the closest solution. This closest can be found using

f(n) = g(n) + h(n), where g= cost to move from starting point to a selected node on a grid

and h = the estimated movement cost to move from that given square on the grid to the destination.

The above search effectivity depends on the selection of a smart h value. To calculate this smart value, Euclidean and Manhattan search functions can be used. We have used the Manhattan search function and the Manhattan distance is calculated between boxes and targets and used to find the smallest distance between them. The Manhattan distance can be defined as -

distance to box = abs (object[0] - target[0]) + abs(object[1] - target[1])

Since we are following a breadth-first approach to finding the shortest distance, we will start from the left-hand side and move to the right-hand side to find which nodes are to be expanded first.

Once we get the set of moves/actions to solve the Puzzle, we pass it to the result function. The result function returns a solution which consist of set of nodes(actions) with a minimal path cost. These actions can be used as direction to solve the puzzle.

**Limitations:**

The A\* algorithm computes quickly and is faster as compared to other options, however, if we continue to add more boxes the complexity of the problem/game increases and it becomes more difficult to calculate the shortest path and reaching a conclusion can take some time.

Although this algorithm works best for the current problem at hand, we will need to find a better algorithm which can compute faster as the complexity of Sokoban increases, such as more boxes being added. Since our algorithm uses the BFS approach, it is easier to find shorter paths but if all the solutions have longer paths, it becomes difficult for the algorithm to process and hence increase the computation time for solving the game.

Apart from this, A\* algorithm is the best suited solution for Sokoban in our case.