|  |
| --- |
|  |
| Sokoban Assignment |
| Intelligent Search – Planning |
|  |
|  |
| Report  Submitted By:  n9400001 Moira Quinn  n7226209 Maurice Cafun  n9708651 Christopher O'Rafferty  02 May 2017 Background |

Sokoban is a computer puzzle game in which the player pushes boxes around a maze in order to place them in designated locations. While Sokoban is just a game, it models a robot moving boxes in a warehouse. As such, it can be treated as an automated planning problem. Sokoban is an interesting problem for the field of artificial intelligence largely due to its difficulty. The difficulty of Sokoban comes not from its branching factor of 4 (up, down, left, right), but because of the huge depth of the solutions. Additionally, a move may leave the puzzle in a state in which it is impossible solve, creating a state of deadlock. An important feature of this problem is the player can only push a single box at a time and is unable to pull any box.

# Assignment Aim

|  |
| --- |
|  |

The aim of this assignment is to design and implement a solver for Sokoban.

# Solver Overview

In order to find a solution to this problem, a number of uninformed and informed search methods were available. After considering each method carefully, it was decided that the A star graph search algorithm would be implemented for this assignment.

The A star search is an informed search algorithm. This means it searches all paths for the path which incurs the smallest cost, while first considering which path appears to lead to the solution quickest. This path is determined by whichever action minimises the following equation:

*n = last node on the path*

*g(n) = cost of the path from the start node*

*h(n) = heuristic that estimates the cheapest path from n to the goal*

This form of intelligent search allows the solver to search the problem space more efficiently than other methods, provided the heuristic is admissible. More details about heuristics and performance of the solver presented are discussed below.

## Heuristics

Heuristics are functions which estimate how close the current state is to the goal state. They are specifically designed for a particular search problem, with examples including the Manhattan distance or Euclidean distance for a shortest path finding problem. However, heuristics are only optimal if they are admissible. A heuristic is only admissible if it satisfies:

*h(n) = estimated heuristic cost*

*h\*(n) = true cost to the nearest goal*

Overall heuristics are good for larger problems that humans can't solve themselves. For the solver presented, the chosen heuristic uses the shortest Manhattan distance from a box to a goal as shown in Figure 1 below.

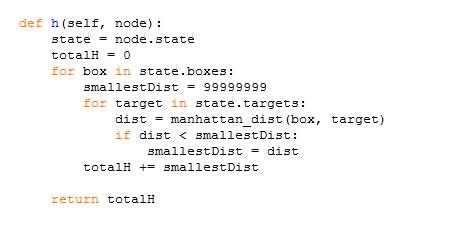


Figure 1: Python Implementation of Heuristic

## Performance

When compared with a breadth first graph search, which searches the entire problem space for a solution, it was found that the A star search was XX times faster as shown in Figure 2 below.

(Insert side by side screenshot of python output)

Figure 2: Breadth First Graph Search vs A Star Graph Search

The performance of an A star search was significantly faster for large search spaces when compared to a breadth first search (BFS). However, for smaller search spaces the A star search did take slightly longer than the BFS. Given the small overall time taken for both to solve the problem (a second or less), this loss of performance can be seen as negligible for these smaller problem spaces. Table 1 below summarises the performance of both BFS and A star searches for different problem sizes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Search Type | Number of Boxes | Number of Goals | Solution Found | Time Taken to Solve |
| Breadth First Graph Search |  |  |  |  |
| A Star Graph Search |  |  |  |  |
| Breadth First Graph Search |  |  |  |  |
| A Star Graph Search |  |  |  |  |
| Breadth First Graph Search |  |  |  |  |
| A Star Graph Search |  |  |  |  |

Table 1: Performance of Breadth First and A Star Search

# Conclusion

(INSERT ONCE FINISHED)