Project 1

Justin Pham

SID 862114467

Email [jpham079@ucr.edu](mailto:jpham079@ucr.edu)

Date Nov 5 2021

To complete this assignment, I consulted:

* Blind Search and Heuristic Search slides
* Notes take during lecture
* C++ documentation
  + <https://www.cplusplus.com>
* The following YouTube video to understand what counts as one “move”
  + <https://www.youtube.com/watch?v=o4ZDw9oFlP8>
* My fellow students for idea
  + No code was exchanged. No student had access to another student’s work in any way. We discussed possible heuristics and strategies to solve the problem.

All code is original except for subroutines for **vector**, **priority\_queue, unordered\_set**, and **string** functions

* **vector** and **priority\_queue** is used to handle queuing nodes
* **unordered\_set** is used to hash previous nodes, to keep a history of sorts
* **string** is used to represent the nodes.
  + subroutines **compare** and **find** and used to compare and find elements inside the string

Outline:

* Cover Page (this page)
* My report (pages 2-
* Sample trace on an easy problem
* Sample trace on a hard problem
* My Code. Github Link: <https://github.com/Pjsrcool/CS-170-Project-1>

CS 170: Project 1 Extra Credit: Nine Men in a Trench

Justin Pham, SID: 862114467, Nov 8 2021

**Introduction**

Search algorithms can be very useful on puzzles with a grid. Sliding puzzles such as Nine Men in a Trench are a great example. Not only can we use a search algorithm to traverse the puzzle board, we can use search to solve the puzzle itself. The premise of Nine Men in a Trench, is to move the sergeant, represented as the number 1, to the left hand side of the trench. There are recesses and other men in the trench, meaning we need to move men into and out of the trench to relocate the sergeant to the left hand side. The puzzle is solved when the sergeant is on the left hand side and the other men are in their proper positions.

This report details my findings from completing Project 1 of Professor Eamonn Keogh’s Introduction to AI course at UC Riverside, Fall 2021. The following heuristics will be explored: no heuristic, misplaced tiles, Manhattan on the sergeant, obstructing men, left man. The code is written in C++.

**The Heuristics**

The heuristics are: no heuristic, misplaced tiles, Manhattan distance on the sergeant, obstructing men, left man.

For clarity, cost, weight and f(n) will mean the same thing. f(n) = g(n) + h(n)

g(n) will represent the depth of the node

h(n) will represent he the heuristic

In the example boards, ‘ - ‘ will represent an inaccessible space, ‘0’ will represent an open space, ‘1’ will represent the sergeant, and numbers will represent men.

No Heuristic

By using no heuristic, essentially perform a Uniform Cost Search. When performing a Uniform Cost Search with my code, the heuristic function will simply return h(n) = 0 each time a heuristic is requested. Since h(n) = 0, we do not look into the future, but only the past. We essentially prioritize searching nodes with the lowest depth, a Breadth First Search. The following heuristics will look at both the past (g(n)) and the future (h(n)).

Misplaced Tiles

This the probably the best heuristic we have. For Misplaced Tiles, we simply count the number of men who are not in their proper positions. It ignores empty space and inaccessible space. The heuristic function will return h(n) = {number of men who are not in the correct position}. For example, the following board have a h(n) = 3 because the sergeant, man 2, and man 3 are not in the proper positions.

[-, -,0,-,1,-,-]

[3,2,0,4,0,0]

Manhattan on the Sergeant

The Manhattan distance is the distance we travel by moving vertically or horizontally on a grid. For this heuristic, we use the Manhattan distance between the sergeant and the his goal position. The fact that we use distance means that the Manhattan on the Sergeant heuristic is not admissible. For example, if we have the following board:

[-, -,0,-,0,-,-]

[0,0,0,0,0,1]

This board is solvable in one move: move the Sergeant straight to the left. But heuristic returns h(n) = 5. Since, h(n) is higher than the actual solution, it not admissible. As a result, the use of Manhattan on the Sergeant heuristic will often return a solution with a depth 1 or 2 higher than the depth of the optimal solution.

Obstructing Men

In the Obstructing Men heuristic, we simply count the number of men between the sergeant and his destination. We return that count as our heuristic. The intuition is that we much move each obstructing man to make way for the sergeant. Each man we move is one cost. Technically, we should multiply the by 2, since we need to move the obstructing men back, but it would be pointless because it does not improve the search in any way. Consider the following for as an example:

[-,-,-,0,-,1,-,-]

[0,3,0,5,0,2,0,4]

Here, man 2, man 3 and man 5 are between the sergeant and his destination. That is 3 men so h(n) = 3.

Left Man

The Left Man heuristic works by checking each man that is in the proper position. If a man is the proper position, check if the man on his left is in the proper position. If he is not, then we increment the heuristic value. The main weakness of this heuristic is that it relies on men being in the proper position. If no man is in the proper position, then h(n) = 0. Since this happens more frequently than not, Left Man is only marginally faster then No Heuristic, Uniform Cost Search. To illustrate an example, we have the following board:

[-,-,-,1,-,0,-,-,]

[4,2,5,3,0,0,0]

Only man 2 is in the correct position, and the man to his left (man 4) is in the wrong position. So h(n) = 1.

**Performance**

Below are charts comparing the performance of each heuristic. For comparison, we will using the following 2 test cases:

Easy Case:

[-,-,0,-,0,-,-]

[2,3,4,1,0,0,0]

Hard Case:

[-,-,-,0,-,0,-,0,-,-]

[0,2,3,4,5,6,7,8,9,1]

As we can see, the number of nodes.....