CS440 MP2 Report

Andong Jing ajing2 - 3 credit In charge of

Siping Meng smeng10 - 3 credit In charge of 1.1 A\* Search

Siyu Tao siyutao2 - 3 credit In charge of Part2 Five in a Row

# 1. Smart Manufacturing

## 1.1 Planning Using A\* Search

In this part we implemented a powerful A\* search with 2 different heuristics, one for unit step search, which is designed to find the smallest possible number of stops, and the other is for finding the smallest possible distance path.

* Minimum Step Explanation

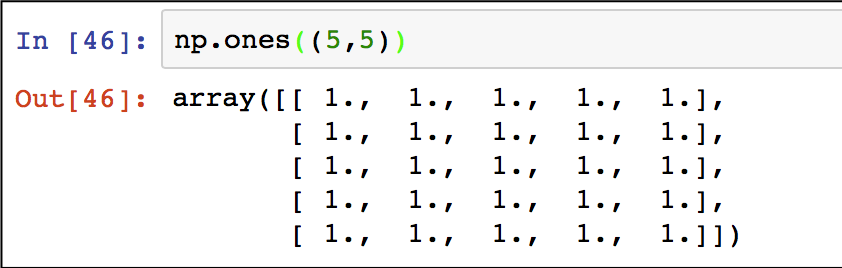
This first heuristic, in our solution is to define a matrix that equals the size of the widget table. For example, it looks like the picture below. Each row represents a specific widget and 1 means we did not get the component, and 0 means we have already got the component.

Figure Sample of our heuristic for min step

The way we calculate the value h() is the length of the longest path we left right now. For example, the following figure represents a specific state with heuristic value equals to 5.

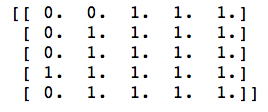


Figure Sample heuristic table

The reason we do this heuristic calculation is that whatever the combination of factory sequences would be, the minimum step of finishing making all the widgets is 5. In the following case, we can only travel 5 times to get every widget satisfied. And the gscore, which is the cost from the starting point should increment 1 after each step. In finding minimum step searching, we ignore the distance between each city.

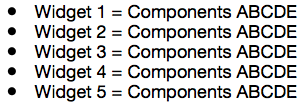


Figure Sample of condition that takes minimum steps

To make the whole process faster, we used “heapq” library in python so that we can get the smallest f\_ value from the list in linear time.

|  |  |  |  |
| --- | --- | --- | --- |
| Starting point | Node expanded | Result | Length of the result |
| **A** | **2704** | **ABEDCADBCDE** | **11** |
| **B** | **3925** | **BDAEDCBACDE** | **11** |
| C | 5314 | CBDAEDCBACED | 12 |
| **D** | **2018** | **DABEDCABCDE** | **11** |
| E | 5314 | EBDAEDCBACED | 12 |

Table Result of running A\* search for the minimum step

Starting from A, B and D will give the minimum step path of length 11.

* Minimum Miles Traveled Explanation

For this heuristic, we implemented the similar table strategy to indicate if our searching is in the goal state or not. However, the biggest difference in this part is that the miles counts to the gscores and fscores. For gscore, it is simply adding the new distance between the current city and the next city, but the trick is that some cities can have faster route if we travel to another middle city. Here is an example for this condition:

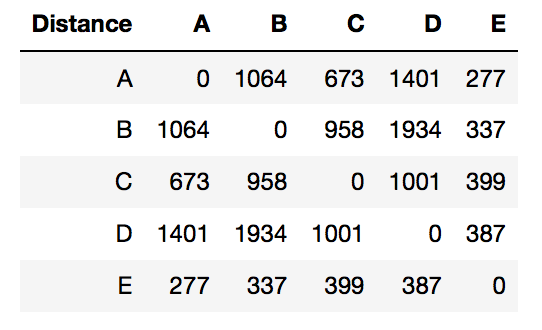


Figure Distance between each cities

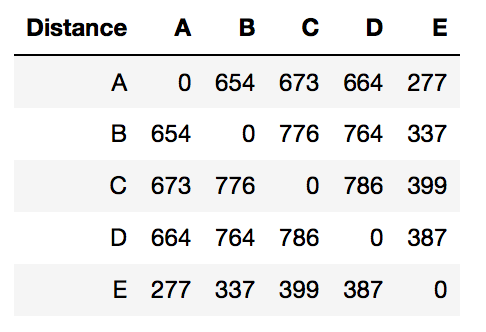
If we are going to travel from A to B, the distance should be 1064 mile, however, if we travel from A to E and then travel E to B, the total distance would be only 614 miles. In order to put this into consideration, we first calculate the minimum path of both cities.

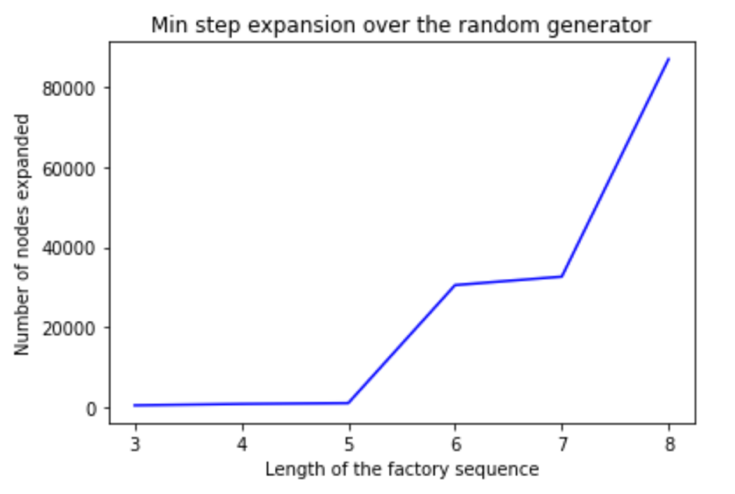
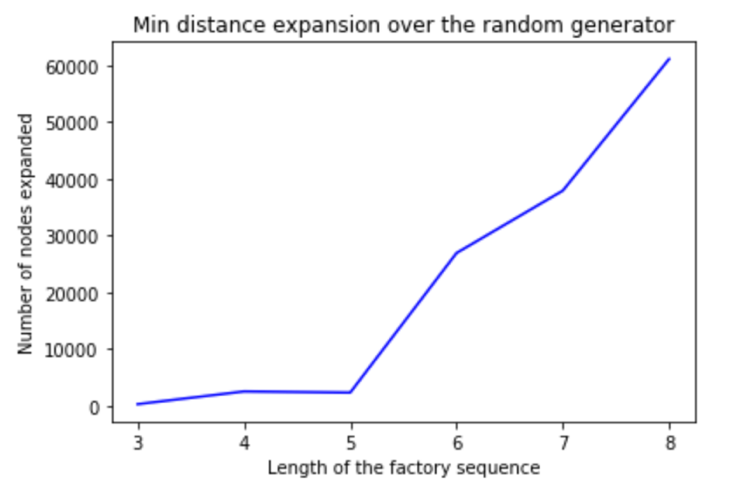
Figure Minimum distance between cities

After making sure the smallest possible distance between cities, we could apply our heuristic on the table. To calculate the heuristic, every time we get to a city, I will calculate the minimum distance between this city and its neighbors and so on. For example, if we are currently in city A, the starting point, the heuristic for A is A->E->A->E->A, which equals 277\*5 = 1385. If we are in city B, the heuristic is B->E->A->E->A which equal to 1168 because this is the lowest mile the current state could go.

|  |  |  |  |
| --- | --- | --- | --- |
| Starting point | Node expanded | Result | Distance |
| A | 3825 | AEBEDECAEDEBECED | 5867 |
| B | 2037 | BEDEAEDEBECEBEAED | 5726 |
| C | 3712 | CEBEDEAEBEDECEBEAED | 6502 |
| **D** | **1415** | **DEBEAEBEDECEBEAED** | **5716** |
| E | 3741 | EBEDEAEBEDECEBEAED | 6103 |

Starting from D will give the minimum distance traveled.

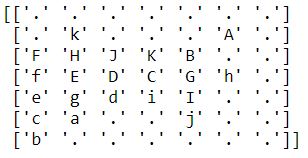
## 1.3 Extra Credit

* **1.3.1**
* **1.3.2**

# 2. Game of Gomoku (Five-in-a-row)

## 2.1 Reflex Agent

* Reflex vs. reflex result



## 2.2 Minimax and Alpha-Beta Agents

* Implementation of minimax and alpha-beta search

1. **Evaluation Function**

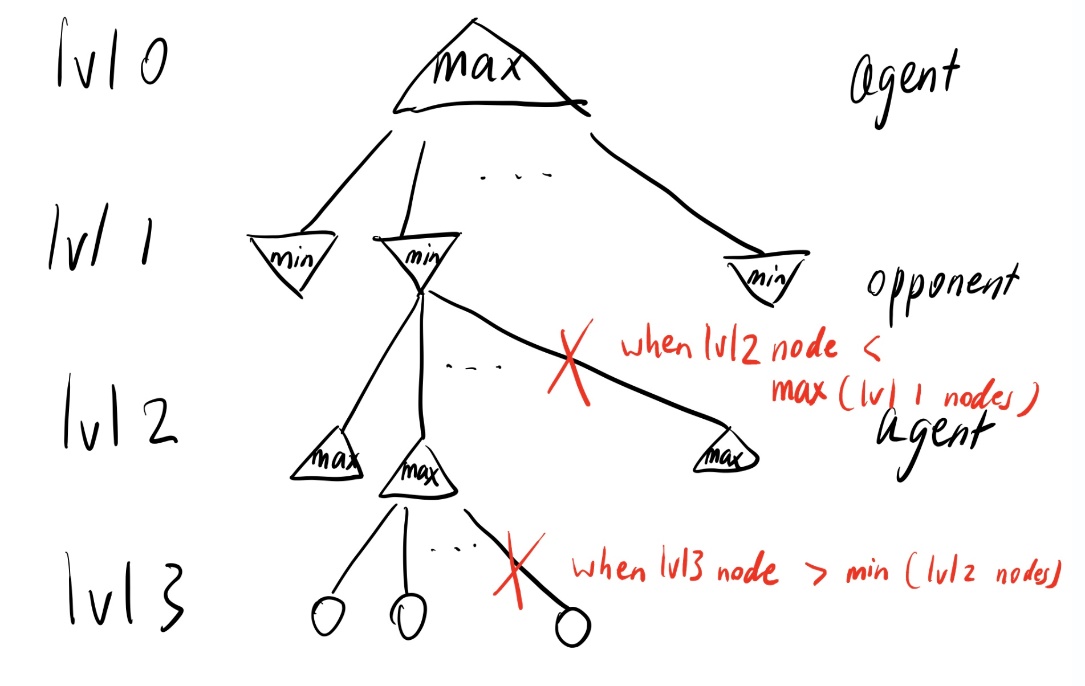
The evaluation function of our group searches for every “winning block”, as defined in rule 4 of the reflex agent, for both the agent side and the opponent side. An empty board has 21 (horizontal) + 21 (vertical) + 18 (diagonal) = 60 such winning blocks for each side.

For each agent’s winning block, if a winning block contains 1 agent’s stone, 1 point is assigned. If a winning block contains 2 agent’s stones, 10 points are assigned. If a winning block contains 3 agent’s stones, 100 points are assigned. If a winning block contains 4 agent’s stones, 1000 points are assigned. If a winning block contains 5 agent’s stones, 10000 points are assigned. For each opponent’s winning block, if a winning block contains 1 opponent’s stone, -1 points is assigned. If a winning block contains 2 opponent’s stones, -10 points are assigned. If a winning block contains 3 opponent’s stones, -100 points are assigned. If a winning block contains 4 opponent’s stones, -1000 points are assigned. If a winning block contains 5 opponent’s stones, -10000 points are assigned. An easy-to-read table is included as follows.



2. **Alpha-beta pruning**

Our team recognized two opportunities of alpha-beta pruning in three levels of minimax search.



As shown by the diagram above, a level three node could be pruned if its value (according to evaluation function) is larger than the minimum of all level two nodes since each level one node is going to minimize its collection of level two nodes. In addition, a level two node could be pruned if its value is less than the maximum of all level one nodes since the root node is going to maximize all level one nodes.

For level three pruning, we keep track of the minimum of all level two nodes each time a node gets added to level two. Once we discovered a level three node that’s larger than that value, we stopped current level three node expansions and continue with the next node in level two to expand. Level two pruning is conducted in a similar manner.

* Match-up results

1. **alpha-beta vs. minimax**



2. **minimax vs. alpha-beta**



3. **alpha-beta vs. reflex** (alpha-beta won, igfhj)



4. **reflex vs. alpha-beta** (alpha-beta won, HDCIJ)



5. **reflex vs. minimax** (minimax won, HDCIJ)



6. **minimax vs. reflex** (minimax won, igfhi)



* # of nodes expanded (minimax vs. alpha-beta)





* Relationship of # of nodes expanded (minimax vs. alpha-beta)

From the tables above and our group’s experimentation with different combinations of matchups, we discovered that at any particular move (say move 2), the minimax agent always expands the same number of nodes. It is because when there are n available positions on the board, the agent always expands n\* (n-1) \*(n-2) nodes. As a result, the number of nodes expanded decreases as the game goes on. The alpha-beta agent always expands less nodes than (strictly less than or equal to) minimax agent at any particular move. The number of nodes expanded doesn’t necessarily decrease as the game goes on. It’s because sometimes early on when there are more possible agent-opponent-agent combinations on the board, the pruning might be more effective, cutting more unnecessary nodes to expand. However, alpha-beta is still always better than minimax in terms of efficiency, while still yielding the same results.