**Smart manufacturing**

**Planning using A\* Search**

**Implementation:**

I first read the 5\*5 matrix of the manufacturing table, I use ArrayList<ArrayList<Integer>> as the data structure. For each components(A~E) selected, I remove the corresponding leftmost one in the list, and shift the sequence of components to the left, and then finally pad “-1” as the rightmost element. The final state is that all the elements of the ArrayList<ArrayList<Integer>> are “-1”.

**First Heuristic(unit path cost):**

The first heuristic I use is the sum of the “number of different components” in each column. Since if the components are the same in the certain column, then they can be produced concurrently. And if the components can’t be produced at the same time, the total path cost will be higher in the end. I take the heuristic function and the path cost(the number of rounds I have been through) as the evaluation function. Since the heuristic function is admissible(the true cost will also be less than the heuristic), so the solution is optimal.

**Second Heuristic(weighted path cost):**

The second heuristic I use is the sum of the “sum of the ‘least distance to another’ of the different components” for each column. This method replaces the “number of different components” with the actual “least path cost” of them. While the path cost function is the distance that has been travelled, I define a 5\*5 cost matrix(distance between each pair of factory). Since the heuristic function is optimal, this will always guarantee to yield the optimal result.

**Random number of rows and columns (3):**

**Implementation:**

I make the rowNumber and colNumber changeable, and the widgets are generated randomly. I compare the A\* search results for minimum number of stops, minimum distance traveled case.

**Uniform cost:**

I create the uniform cost algorithm (Dijkstra’s algorithm) and compare the result with the A\* search. With uniform cost algorithm used, I can be guaranteed that the cost is always minimum, since Dijkstra’s algorithm always starts from the minimum cost path. I remove the heuristic function and only use path cost to build the uniform cost algorithm. The priority queue is sorted by the total path already taken.

**Uniform cost (minimum distance traveled):**

Node expanded:6930

Path cost:12

Path:254314323154→BEDCADCBCAED

(Where 1 means A, 2 means B, 3 means C, 4 means D, 5 means E)