# Report of Project AI

Class IT3160E

Students: Lê Trường Giang: 20194430

Trần Quốc Lập: 20194443

Hoàng Nguyễn Minh Nhật: 20194445

Hoàng Văn Khánh: 20194440, [khanh.hv194440@sis.hust.edu.vn](mailto:khanh.hv194440@sis.hust.edu.vn)

# 1. Presentation of the subject

Topic: Implement a goal-based agent: Intelligent vacuum cleaner for rich people.

Let’s say the family who owns this vacuum cleaner is very rich. On their floor, there is not only dust, but also jewels. The floor is of a size n x m with each tile have the following values

\* 0 for a clean tile

\* 1 for a tile with dust

\* 2 for a tile with a jewel

\* 3 for a tile with a jewel and dust

\* 4 for a tile with an obstacle

A tile is clean if it has no dust or jewel or obstacle.

If tile status is 3, then the cleaner must pick up the jewelry first, to avoid sucking it. (0,0) is the top-left corner and is initially clean, and it contains a box. When picking up a jewel, the agent must bring it back to the box at (0,0) before continuing sucking the dust.

The agent must avoid the obstacles.

The vacuum cleaner agent can perform the following actions:

\* Move up

\* Move down

\* Move left

\* Move right

\* Suck dust

\* Pick up jewel

\* Put jewel in the jewel box

# 2. Description of the problem

**PEAS of our agent:**

* *Performance Measures:*
* Current state
* Position (of agent)
* Next stop
* *Environment:* (Partially observable, deterministics)
* Floor
* Tiles
* Dust and jewel
* *Actuators*:
* Turn left/right
* Move up/down
* Suck (dirts)
* Pick up (jewels)
* Put down (jewel to the box)
* *Sensors*:
* Sensors to recognize environment

**What is inside a goal-based agent ?**

* States: current state - initial state and the cleaner’s position
* Actions
* Goal test
* Path cost
* Solution

**-> In our agent:**

* State:
  + Initial state: The floor represented by a m\*n matrix, with each entry having value of (0,1,2,3,4) represents the state of relative tile. The cleaner starts at (0,0) “tile” of the matrix
  + Current state and cleaner’s position are determined after each action
* Actions:
  + Turn left/right:
    - position(x,y) <- (x-1,y) or (x+1,y)
  + Move up/down:
    - position(a,b) <- (x,y-1) or (x,y+1)
  + Suck (dirts):
    - floor(m,n) = floor(m,n) - 1
  + Pick up (jewels):
    - floor(m,n) = floor(m,n) - 2
  + Put down (jewel to the box):
* Goal test:
  + Floor is cleaned when all tiles (all entries of the matrix) are cleaned or obstacles (value equals to 0 or 4)
* Path cost:
  + Path cost = 0 initially
  + Step cost = 1
  + for each action: path cost = sum of step costs
* Solution:
  + For each action: the solution list is expanded by action.

# 3. Selecting the algorithms to be used for solving the problem

While standing on a tile, the vacuum must decide what action to take. If the tile has dust only or has jewelry, the vacuum cleaner must suck dust and or pick up jewelry, then it has to decide which tile to move to.

For the sake of optimizing, before moving to the next tile, the vacuum should have queried environment status so as to:

\* skip tiles that are already clean or tiles that have obstacles.

\* find the nearest tile that has dust or jewelry and move to that tile

In order to do so, we represent the board in terms of graph and then use Floyd-Warshall algorithm to find the shortest path between any pair of tiles. By applying Floyd-Warshall algorithm, handling obstacles becomes easy.

## Algorithm description:

1. Number tiles on the board consecutively from to , where .

2. View the board in terms of graph, where 2 vertices i, j of the graph are adjacent if tiles i, j in the board are adjacent.

3. Build a table matrix to save the cost travelling from tile i to tile j. Simultaneously, we maintain a table path where path[i, j]indicates the tile to traverse on the path from i to j in order to achieve the shortest path from i to j.

Initially, matrix[i, i] <- 0 for all tile i, as the cost to move from a tile to itself is .

matrix[i, j] <- 1 for all adjacent tiles i and j.

matrix[i, j] <- +∞ if i and j are not adjacent or at least one of them has an obstacle.

Initially, path[i, j] = j, that means the shortest path from i to j is to move directly from i to j.

4. Apply Floyd-Warshall algorithm to update path and matrix, i.e to calculate the shortest path between every pair of tiles (i, j) and the cost of that path.

5. Start cleaning: at each step when the vacuum cleaner has to determine the next tile to move to, we lookup in matrix and pathto find the nearest tile. In case there are several candidates, we order them by clean level, say 1 -> 2 -> 3. Then we follow the corresponding route in path**.**

At step 4, we applied the Floyd-Warshall algorithm. It’s a method used to find shortest paths between any pair of vertices (u, v) in a weighted graph. In our circumstance, we use matrix as the cost table.

The idea of the Floyd-Warshall algorithm is: For each vertex k, we consider every pair of vertices (u, v), we can minimize the path from u to v by the formula:

matrix[u, v] = min(matrix[u, v], matrix[u, k] + matrix[k, v])

That means: if the current path from u to v is longer than the path from u to k plus the path from k to v, then we remove that current path and accept the new path.

Function Floyd:

For each tile i in board:

For each tile j in board:

path[i, j] = j

for each tile k as intermediary:

for each tile i as origin:

for each tile j as destination:

if matrix[i, j] < matrix[i, k] + matrix[k, j]:

matrix[i, j] < matrix[i, k] + matrix[k, j]

path[i, j] = path[i, k] # move to k right after leave i

The advantage of Floyd-Warshall algorithm lies in the reusability because the shortest path between all pairs of vertices is calculated in advance. Therefore, querying what the next tile to move to and how to move to that tile requires minimal cost.

However, the Floyd-Warshall algorithm requires auxiliary time complexity of and space complexity of . We have to maintain 2 tables matrixand pathtill the end of the program, especially when is very large. Moreover, under no circumstance we have to use the entire table.

At step 5, by looking for the nearest tile viewed from the current position of the vacuum, we’re applying the greedy method. Though it may not turn out an optimal solution with lowest cost, it produces a high quality solution in a reasonable computation time, especially when the data set is large.

# 4. Implementing the algorithms to be used for solving the problem

board, starting\_tile <- Create\_Data(m, n)

matrix, path <- Floyd\_Warshall(board)

X <- <container for vacuum's journey, initially empty>

WHILE <the board is not clean>:

destination <- Find\_Tile(starting\_tile) # return the nearest and less untidy tile (but still not clean yet)

route <- Shortest\_Path(path, starting\_tile, destination)

IF <destination has dust only>:

X.APPEND(starting\_tile, destination, route, "SuckDust")

starting\_point <- destination

ELSE:

X.APPEND(starting\_tile, destination, route, "Pickup")

route <- Shortest\_Path(path, destination, box)

X.APPEND(destination, box, "PutDown")

starting\_point <- box

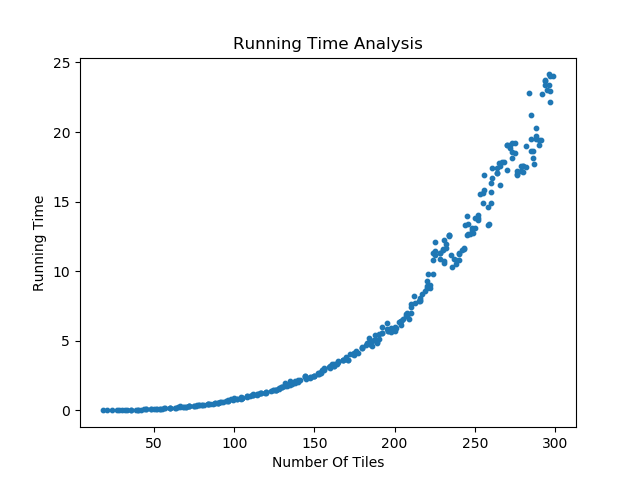
<PRINT X>

# 5. Analysis and explanation of experiment

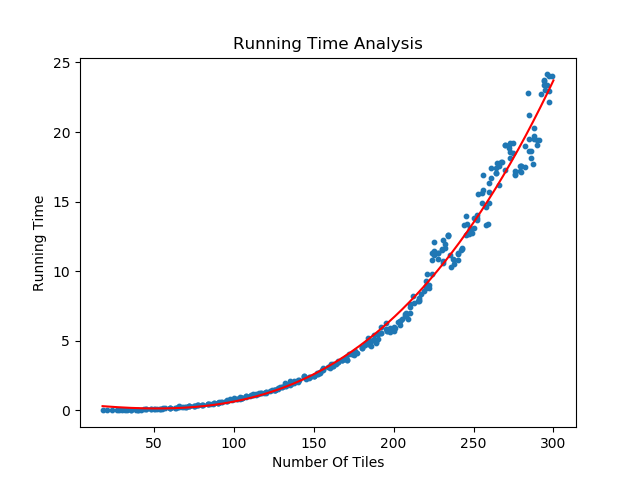
Floyd WarShall algorithm has the time complexity is , the running time mainly depends on Floyd WarShall algorithm, the greedy algorithm takes up a very small fraction of the time with time complexity is . The algorithm has to store all pairs shortest path and cost so the space complexity will be , we also have to store the board, it takes . In conclusion, our algorithm have:

* Time complexity is
* Space complexity is

We generate 300 instances of our problem with the increasing number of tiles ( size of instance: N). We run algorithms with these instances and obtain the result.



We can see the running time in polynomial order of size of instance. With any instance having the same size, the difference between them is insignificant. The tendency of the dot tends to follow the polynomial in order 3. We try to use the polynomial regression and we obtain



The red line represents the tendency of the dot. The experimental results have verified the complexity of the algorithm that we gave at first

# 6. Conclusion and possible extensions

Vacuum cleaner for the rich is a very simple example of a goal-based agent. The implementation of the algorithm that we’ve used is not very complicated. But as working on a project in a team, we’ve learned how to teamwork. Moreover, in the past most of us haven’t practiced much of the algorithms that we’d been taught, as a result we’d faced some difficulties when implementing the program. So this project has provided a chance for us to practice and have a deeper understanding of the algorithms that we’ve used.

If we had more time, we would optimize our source code or try to come up with another algorithm in order to achieve a better performance. Because we never use the entire path and matrix table, the Floyd-Warshall algorithm seems to be auxiliary in time and space complexity. The hard work is to find a way to reduce unnecessary shortest paths. We’re thinking of an approach where, while standing on a tile, the vacuum starts to find the shortest paths to only necessary tiles without having to pre-calculate the whole large table. We may also consider other algorithms such as Dijkstra or Ford-Bellman.

# 7. List of tasks

Problem analysis: Le Truong Giang

Selecting the algorithms: Le Truong Giang

Implement algorithm: Le Truong Giang

Virtualize problem: Le Truong Giang

Writing part 1 of the report: Tran Quoc Lap

Writing part 2 of the report: Hoang Nguyen Minh Nhat

Writing part 3 of the report: Tran Quoc Lap

Writing part 4 of the report: Tran Quoc Lap

Writing part 5 of the report: Le Truong Giang

Writing part 6 of the report: Tran Quoc Lap

Writing part 7 of the report: Tran Quoc Lap

Writing part 8 of the report: Hoang Van Khanh

Writing part 1 of the presentation: Hoang Nguyen Minh Nhat

Writing part 2 of the presentation: Hoang Van Khanh

Writing part 3 of the presentation: Hoang Nguyen Minh Nhat

Presentation refining : Hoang Nguyen Minh Nhat

Create the demo video: Le Truong Giang

# 8. List of bibliographic references

Artificial Intelligence: A Modern Approach, Third Edition, Stuart Russell and Peter Norvig