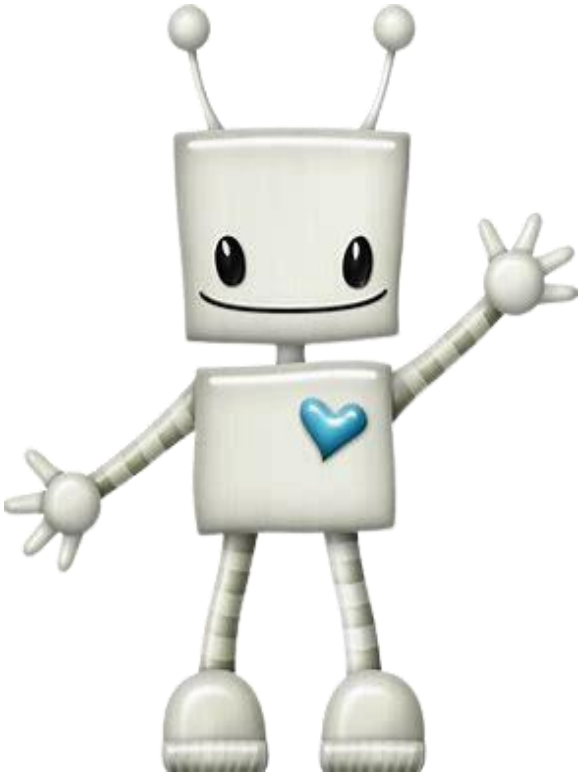
Café Robot



By Sara Amjad Anabtawi 212071

Table Of Contents

Sections Page

List of Figures

Figure Page

1. Description:

The Robot is an employee in a café called Café Robot and is responsible of serving coffee to the university Teaching assistants as well as the Module Leaders which might be sitting in either the Ta's office or the H-CR-104 classroom. The robot gets the coffee from the Coffee shop that is located in H105 (Room 1) before delivering it to the university staff.

2. Approaches and Plans

2.1 Assumptions:

- The robot is an omnidirectional Robot.
- The robot can carry and serve 1 or 2 coffees at a time.
- The robot can only serve at the tables that aren't empty and has someone sitting on it.
- The robot can serve two cups of coffee to the same table.
- The robot can only serve the teaching assistants or module leaders.
- The robot's initial state is in the Coffee shop that is located in H105 (room 1) with one or two coffees to be delivered to TAs or MLs.
- The robot heads to the coffee shop once they are done delivering.
- The robot goal state is for the robot to be back to the coffee shop 'Café Robot' after delivering the Coffee's successfully.
- The Robot does not serve outside its defined environment such as going near the stairs and the rest of the floor 1 or the rest of the building.
- The Robot serves coffee to room H104(Room2) only if the one who requested the coffee was a module leader.
- The doors are opened and closed automatically.

2.2 Plans

2.2.1 Ideal Scenario 1

The Robot was waiting in the Coffee shop that is located in H105(Room1) waiting for any requests, then receives 1 coffee request after receiving that request it went to the coffee machine to collect that coffee order then heads to the table that requested the coffee(goal node) while heading to the table it chooses the shortest path while avoiding any obstacles that it might end up bumping into resulting in spilling the coffee and after delivering it successfully it went back to the coffee shop (initial state).

2.2.2 Ideal Scenario 2

The Robot was waiting in the Coffee shop that is located in H105(Room1) waiting for any requests, then receives 2 coffee requests after receiving that request it went to the coffee machine to collect 2 coffees then heads to the 2 tables that requested the coffee(goal nodes), while heading to the 2 tables it chooses the shortest path while avoiding any obstacles that it might end up bumping into resulting in spilling the coffees and after delivering both orders successfully it went back to the coffee shop (initial state).

2.3 Approaches

2.3.1 Graph Representation

Graphs consist of two main components: nodes (vertices) and edges (arcs). Nodes represent entities or elements, while edges represent relationships or connections between these entities.

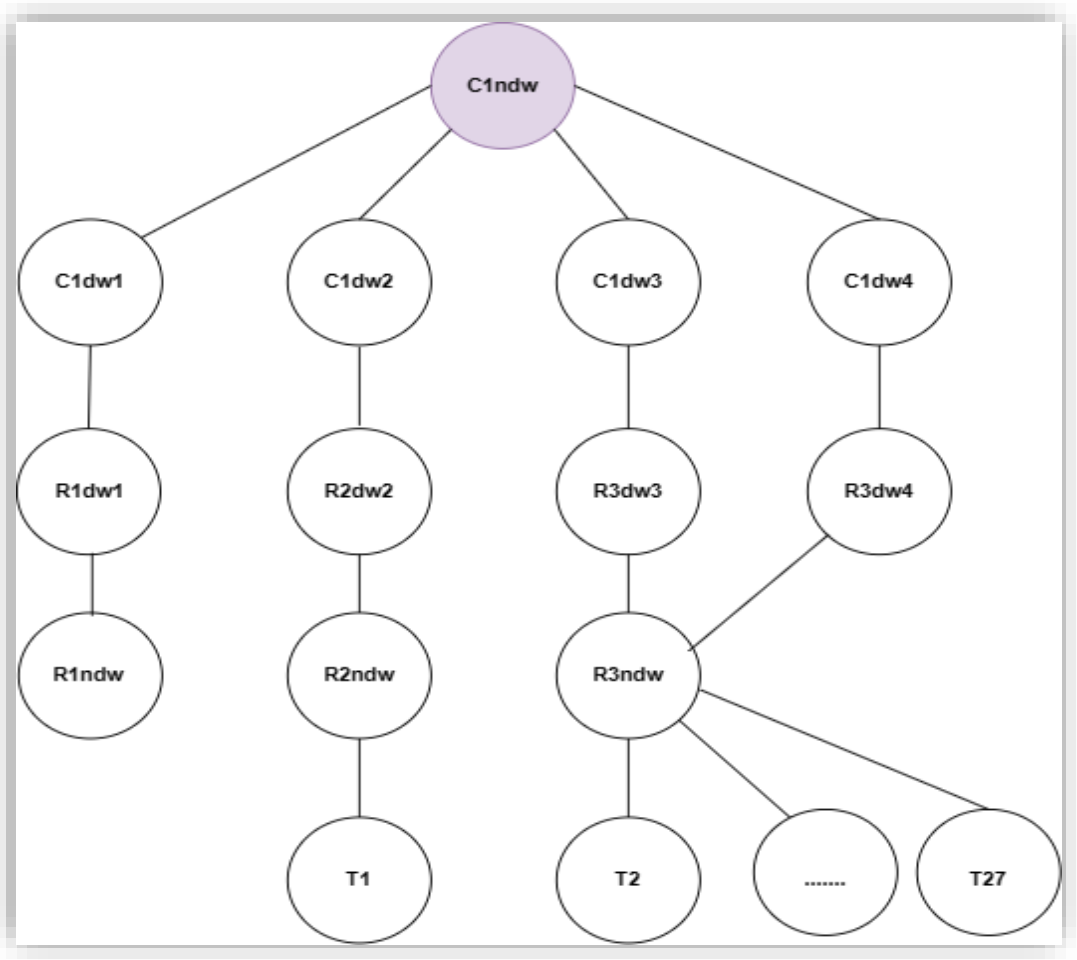


Figure 1

Where

"R": Room,

"C": Corridor.

“dw”: doorway, which symbolizes the area close to a door.

“ndw”: no doorway, which symbolizes a room's empty flooring.

T = [T1, T2, …, TS]: Here, "T" stands for Table, and "S" for the number of tables that can request coffee orders. (including the Module leader and Teaching assistant’s tables).

An adjacency matrix is a square, two-dimensional array used to describe graph notations, where the rows and columns represent the nodes in the graph and the matrix cells denote the existence or lack of edges connecting the nodes. For effectively displaying the connection of nodes in a graph, it is an essential tool.

	C1ndw	C1dw1	C1dw2	C1dw3	C1dw4	R1dw1	R2dw2	R3dw3	R3dw4	R1ndw	R2ndw	R3ndw	T1	T2	T3	T4	T5	T6	T7	.....	T27
C1ndw	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C1dw1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C1dw2	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C1dw3	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
C1dw4	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
R1dw1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
R2dw2	0	0	1	0	0	0	0		0	0	1	0	0	0	0	0	0	0	0	0	0
R3dw3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R3dw4	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R1ndw	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R2ndw	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
R3ndw	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1
T1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
T2	0	0		0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
T6	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
T7	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
.....	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
T27	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 2

2.3.2 List Representation

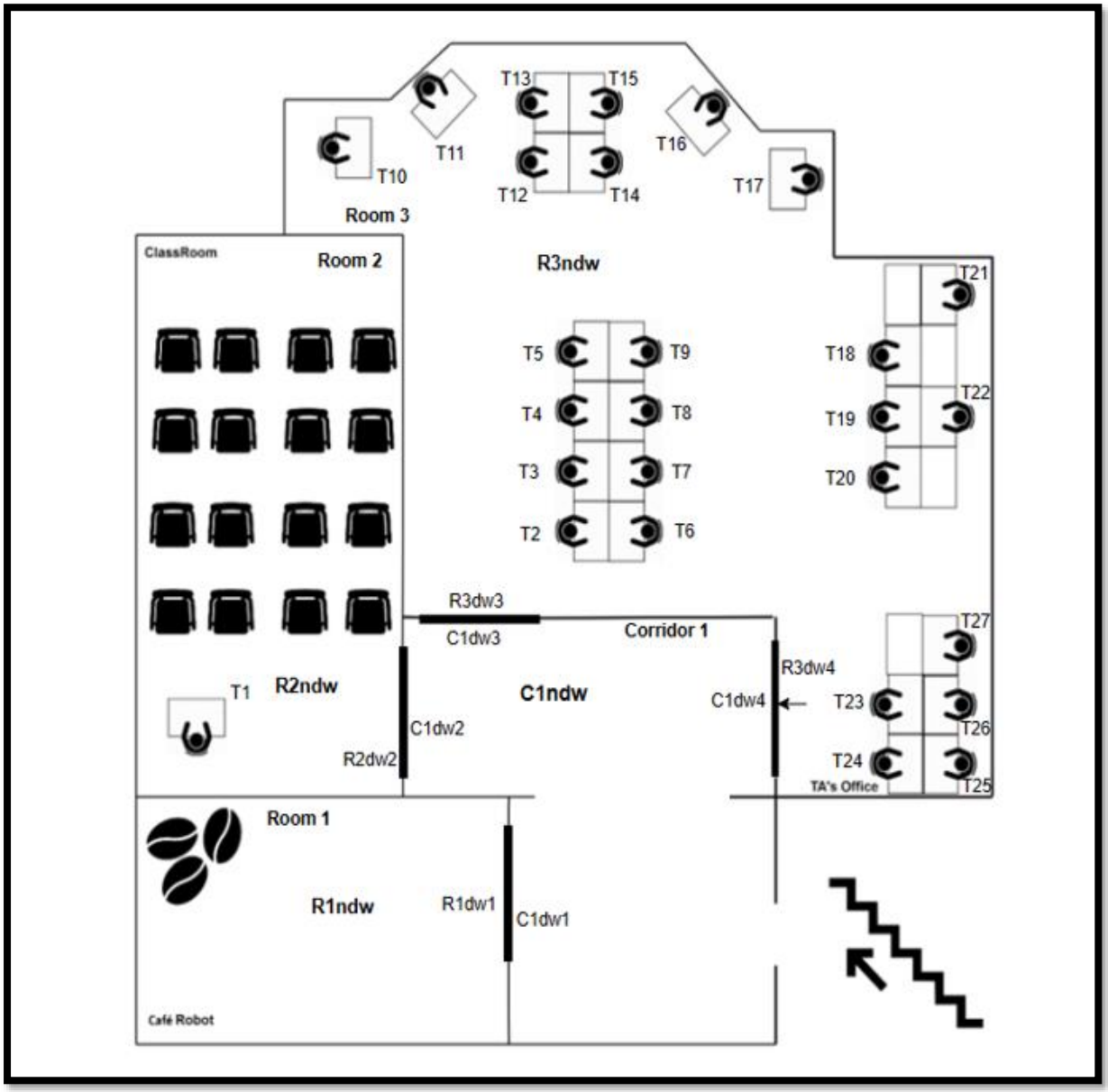


Figure 3

Where

“R”: Room,

“C”: Corridor.

“dw”: doorway, which symbolizes the area close to a door.

“ndw”: no doorway, which symbolizes a room's empty flooring.

T = [T1, T2, …, TS]: Here, "T" stands for Table, and "S" for the number of tables that can request coffee orders. (including the Module leader and Teaching assistant's tables).

2.3.3 Space Representation

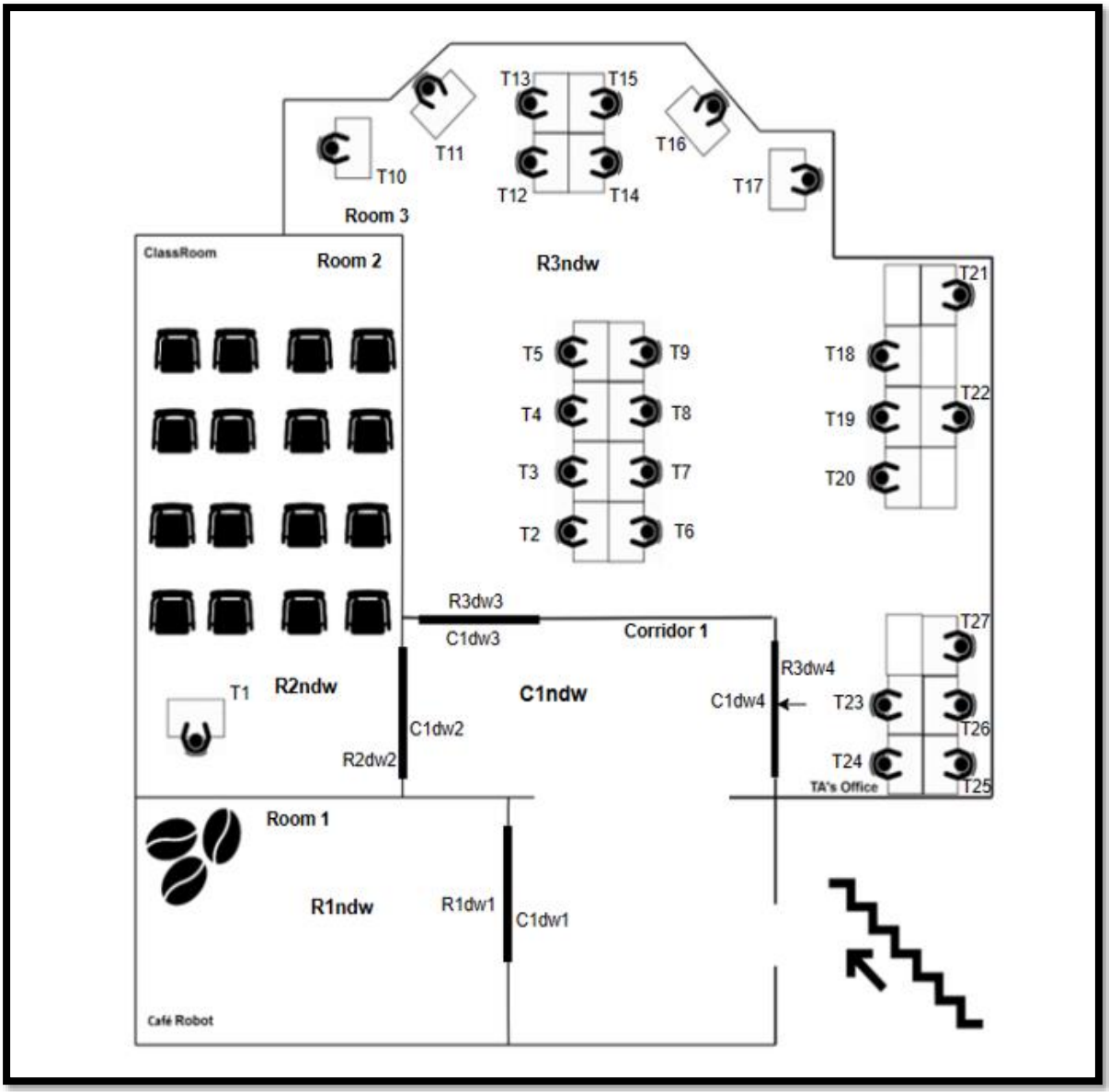


Figure 4

Where

"R": Room,

"C": Corridor.

“dw”: doorway, which symbolizes the area close to a door.

“ndw”: no doorway, which symbolizes a room's empty flooring.

T = [T1, T2, …, TS]: Here, "T" stands for Table, and "S" for the number of tables that can request coffee orders. (including the Module leader and Teaching assistant’s tables).

3. Computer Representations

3.1 Graph Representation

3.1.1 Notations

Using dictionaries to store the locations as the keys and their connections in lists as the values.

```
Building_H = {
    "C1ndw": ["C1dw1", "C1dw2", "C1dw3","C1dw4"],
    "C1dw1":["R1dw1","C1ndw"],
    "C1dw2":["R2dw2","C1ndw"],
    "C1dw3":["R3dw3","C1ndw"],
    "C1dw4":["R3dw4","C1ndw"],
    "R1dw1":["C1dw1","R1ndw"],
    "R2dw2":["C1dw2","R2ndw"],
    "R3dw3":["C1dw3","R3ndw"],
    "R3dw4":["C1dw4","R3ndw"],
    "R1ndw": ["R1dw1"],
    "R2ndw": ["R2dw2","T1"],
    "R3ndw": ["R3dw3", "R3dw4","T2","...","T27"],
    "T1":["R2ndw"],
    "T2":["R3ndw"],
    "T3":["R3ndw"],
    "T4":["R3ndw"],
    "T5":["R3ndw"],
    "T6":["R3ndw"],
    "T7":["R3ndw"],
    "...":["R3ndw"],
    "T27":["R3ndw"]
}
```

3.1.2 Operations

- Move_to (x, y):
  X means a current location or state, Y means the desired location to go or goal state whether a specific room or table that it should go to.
  This operation means that the robot will move from location X to location Y. In other words, X is the source and Y is the destination.
  Pre: is in X
  After: is in Y

- GoBack(s):
  S stands for the starting point. This Operations allows the robot to go back to its initial state. Which is in this problem is R1ndw.

- get_coffee_Request(y):
  y is the table number to which it requested coffee. The table numbers that requested will be added to a queue since the robot can only deal with no more than 2 requests per time.

- Receive_coffee(x):
  x is the number of coffees requested if the number of coffee request is greater than 2 then the robot will receive the first 2 requests only then the rest will be added to the queue so that once the robot deal with the 2 coffee cups the robot is holding, it will start receiving the remaining coffee requests but no more than 2 coffee cups at a time.

- Go_through_doorway(i):

  This operation will allow the robot to pass any doorway it meets. In which the i is the doorway number.

- Serve_coffee(t):
  t is a list of the table numbers that the robot should serve the coffee to.
  Pre: is holding the coffee order that is requested.
  After: not holding the coffee since it was placed on the table, the robot is not holding coffee anymore(served).

3.1.3 Example 1:

“C”: Corridor.

“dw”: doorway, which symbolizes the area close to a door.

“ndw”: no doorway, which symbolizes a room's empty flooring.

T = [T1, T2, …, TS]: Here, "T" stands for Table, and "S" for the number of tables that can request coffee orders. (including the Module leader and Teaching assistant’s tables).

- get_coffee_Request (1)
- Receive_Coffee_order (1)
- Move_to (R1ndw, R1dw1)
- Go_through_doorway (1)
- Move_to (C1dw1, C1ndw)
- Move_to (C1ndw, C1dw3)
- Go_through_doorway (3)
- Move_to (R3dw3, R3ndw)
- Move_to (R3ndw, T2)
- Serve_coffee(T2)
- Move_to (T2, R3ndw)
- Move_to (R3ndw, R3dw3)
- Go_through_doorway (3)
- Move_to (C1dw3, C1ndw)
- Move_to (C1ndw, C1dw1)
- Go_through_doorway (1)
- Move_to(R1ndw)

Which includes these steps:

“R1ndw”: [“R1dw1”],

“R1dw1”: [“R1ndw”,”C1dw1”],

“C1dw1”: ["R1dw1”,”C1ndw”],

“C1ndw”: [“C1dw1”,”C1dw2”,”C1dw3”,”C1dw4”],

“C1dw3”: [“C1ndw”,” R3dw3”],

“R3dw3”: [“C1dw3”,” R3ndw”],

“R3ndw”: [“R3dw3”,” T2”,” T3”,” …”,” T27”],

“T2”: [“R3ndw”],

“R3ndw”: [“R3dw3”,” T2”,” T3”,” …”,” T27”],

“R3dw3”: [“R3ndw”,”C1dw3”],

“C1dw3”: [“C1ndw”,” R3dw3”],

“C1ndw”: [“C1dw1”,”C1dw2”,”C1dw3”,”C1dw4”],

“C1dw1”: [“R1dw1”,”C1ndw”],

“R1dw1”: [“R1ndw”,”C1dw1”],

“R1ndw”: [“R1dw1”]

3.1.4 Example 2:

“C”: Corridor.

“dw”: doorway, which symbolizes the area close to a door.

“ndw”: no doorway, which symbolizes a room's empty flooring.

T = [T1, T2, …, TS]: Here, "T" stands for Table, and "S" for the number of tables that can request coffee orders. (including the Module leader and Teaching assistant’s tables).

- get_coffee_Request (2)
- Receive_Coffee_order (2)
- Move_to (R1ndw, R1dw1)
- Go_through_doorway (1)
- Move_to (C1dw1, C1ndw)
- Move_to (C1ndw, C1dw2)
- Go_through_doorway (2)
- Move_to (R2dw2, R2ndw)
- Move_to (R2ndw, T1)
- Serve_coffee(T1)
- Move_to (T1, R2ndw)
- Move_to (R2ndw, R2dw2)
- Go_through_doorway (2)

- Move_to (C1dw2, C1ndw)
- Move_to (C1ndw, C1dw3)
- Go_through_doorway (3)
- Move_to (R3ndw, T5)
- Serve_coffee(T5)
- Move_to (T5, R3ndw)
- Move_to (R3ndw, R3dw3)
- Go_through_doorway (3)
- Move_to (C1dw3, C1ndw)
- Move_to (C1ndw, C1dw1)
- Go_through_doorway (1)
- Move_to(R1ndw)

Which includes these steps:

“R1ndw”: [“R1dw1”],

“R1dw1”: [“R1ndw”,”C1dw1”],

“C1dw1”: ["R1dw1”,”C1ndw”],

“C1ndw”: [“C1dw1”,”C1dw2”,”C1dw3”,”C1dw4”],

“C1dw2”: [“C1ndw”,” R2dw2”],

“R2dw2”: [“C1dw2”,” R2ndw”],

“R2ndw”: [“R2dw2”,” T1”],

“T1”: [“R2ndw”],

“R2ndw”: [“T1”,” R2dw2”],

“R2dw2”: [“C1dw2”,” R2ndw”],

“C1dw2”: [“C1ndw”,” R2dw2”],

“C1ndw”: [“C1dw1”,”C1dw2”,”C1dw3”,”C1dw4”],

“C1dw3”: [“R3dw3”,”C1ndw”],

“R3dw3”: [“C1dw3”,” R3ndw”],

“R3ndw”: [“R3dw3”,” T2”,” T3”,” …”,” T27”],

“T5”: [“R3ndw”],

“R3ndw”: [“R3dw3”,” T2”,” T3”,” …”,” T27”],

“R3dw3”: [“R3ndw”,”C1dw3”],

“C1dw3”: [“C1ndw”,” R3dw3”],

“C1ndw”: [“C1dw1”,”C1dw2”,”C1dw3”,”C1dw4”],

“C1dw1”: [“R1dw1”,”C1ndw”],

“R1dw1”: [“R1ndw”,”C1dw1”],

“R1ndw”: [“R1dw1”]

3.2 List Representation

3.2.1 Notations

Using lists to store the locations and for each location, another list will be stored associated with it which is the connections of this location.

```
Building_H = [
["C1ndw",["C1dw1","C1dw2", "C1dw3","C1dw4"]],
["C1dw1",["R1dw1","C1ndw"]],
["C1dw2",["R2dw2","C1ndw"]],
["C1dw3",["R3dw3","C1ndw"]],
["C1dw4",["R3dw4","C1ndw"]],
["R1dw1",["C1dw1","R1ndw"]],
["R2dw2",["C1dw2","R2ndw"]],
["R3dw3",["C1dw3","R3ndw"]],
["R3dw4",["C1dw4","R3ndw"]],
["R1ndw",["R1dw1"]],
["R2ndw",["R2dw2","T1"]],
["R3ndw",["R3dw3", "R3dw4","T2","...","T27"]],
["T1",["R2ndw"]],
["T2",["R3ndw"]],
["T3",["R3ndw"]],
["T4",["R3ndw"]],
["T5",["R3ndw"]],
["T6",["R3ndw"]],
["T7",["R3ndw"]],
["...",["R3ndw"]],
["T27",["R3ndw"]]

]
```

3.3 Space Representation

3.3.1 Notations

Using lists to store the locations and for each location, another list will be stored associated with it which is the connections of this location.

```
Building_H = [
["C1ndw",["C1dw1","C1dw2", "C1dw3","C1dw4"]],
["C1dw1",["R1dw1","C1ndw"]],
["C1dw2",["R2dw2","C1ndw"]],
["C1dw3",["R3dw3","C1ndw"]],
["C1dw4",["R3dw4","C1ndw"]],
["R1dw1",["C1dw1","R1ndw"]],
["R2dw2",["C1dw2","R2ndw"]],
["R3dw3",["C1dw3","R3ndw"]],
["R3dw4",["C1dw4","R3ndw"]],
["R1ndw",["R1dw1"]],
["R2ndw",["R2dw2","T1"]],
["R3ndw",["R3dw3", "R3dw4","T2","...","T27"]],
["T1",["R2ndw"]],
["T2",["R3ndw"]],
["T3",["R3ndw"]],
["T4",["R3ndw"]],
["T5",["R3ndw"]],
["T6",["R3ndw"]],
["T7",["R3ndw"]],
["...",["R3ndw"]],
["T27",["R3ndw"]]

]
```

4 Planning Strategies

4.1 Breadth First Search

Breadth First Search is a graph traversal algorithm.

The breadth-first search begins at the root node of the graph and explores all its neighbouring nodes. The algorithm investigates the neighboring nodes of each of these nodes once again. Until the desired element is located or every node is used up, this process is repeated. The below Figure (Fig.5) explains how it works.

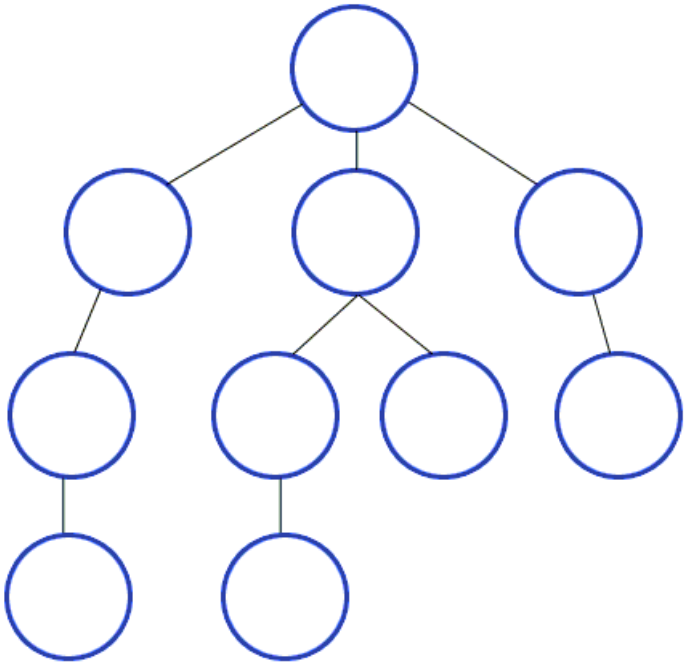


Figure 5

A queue is used as an auxiliary data structure to keep track of the neighbouring nodes.

Properties

Time Complexity

When the algorithm has to travel through every node in the graph, that is the worst-case scenario. The worst-case situation is therefore equal to the sum of the vertices (V) and the edges (E). The expression for this is O(|E| + |V|).

Space Complexity

All the nodes at one level must be stored during the traverse until their descendant nodes in the subsequent level have been visited. Therefore, the deepest depth of the nodes in the network is the space complexity.

Completeness

This is a comprehensive method because, given any kind of graph, if a solution exists, it will be found.

Algorithm

1. Create a queue whose size is equal to the graph's total number of vertices.

2. Choose any vertex to serve as the traversal's beginning point. Go to that vertex, then add it to the Queue.

3. Insert each of the neighboring vertices of the vertex in front of the queue that has not yet been visited into the queue.

4. When there is no new vertex to visit from the vertex at front of the Queue then delete that vertex from the Queue.

5. Continue to step 3 and step 4 until the queue is empty.

6. After the queue is empty, remove any unnecessary edges from the graph to create the final spanning tree.

Application

BFS is an adaptable algorithm with multiple applications such as:

- In an unweighted network, determining the shortest path between two nodes. This is due to the fact that BFS always looks for the quickest route to a node before looking for any longer ones.
- Locating every node that is within a specific range of a source node. Applications like network routing and locating all surrounding businesses on a map can benefit from this.
- Determining a graph's smallest spanning tree. A subset of the graph with as few edges as feasible connecting every node is called a spanning tree.
- Spotting patterns in a graph.
- Carrying out memory management's garbage collection.

Suitability

When solving issues requiring the determination of the shortest path or nearest node to the starting node, BFS is especially well-suited. It works particularly effectively for issues when it's required to visit every node in a graph in a methodical manner.

BFS is less appropriate for issues when it's crucial to investigate the full graph or discover every potential solution, though. This is because BFS may need to visit a lot of nodes before finding a solution, which makes it inefficient for huge graphs.

The following are some instances of issues where BFS is a wise decision:

- figuring out the shortest route on a map to get from your house to your destination.
- locating every friend on a social network.
- locating every webpage that links to a specific website.
- determining a network's minimum spanning tree in order to reduce wire costs.
- finding cycles in a graph to spot possible issues like deadlocks.

4.2 Depth First Search

Depth First Search is a graph traversal algorithm.

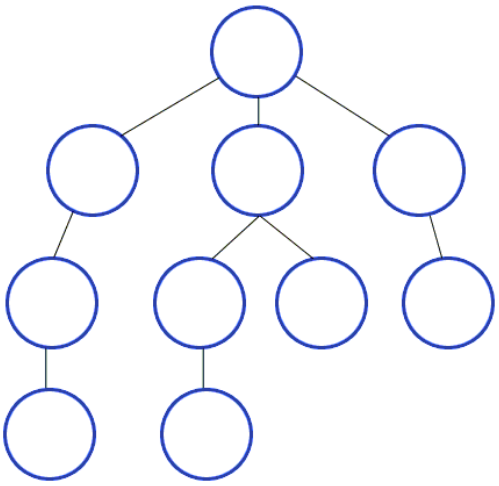


Figure 6

The search starts on any node and explores further nodes going deeper and deeper until the specified node is found, or until a node with no children is found. If a node is found with no children, the algorithm backtracks and returns to the most recent node that has not been explored. This process continues until all the nodes have been traversed.

A stack is used as an auxiliary data structure to keep track of traversed nodes to help it backtrack when required.

Properties

Time Complexity

When the algorithm has to travel through every node in the graph, that is the worst-case scenario. The worst-case situation is therefore equal to the sum of the vertices (V) and the edges (E). The expression for this is O(|E| + |V|).

Space Complexity

The space complexity of a depth-first search is lower than that of a breadth first search.

Completeness

This is a comprehensive method because, given any kind of graph, if a solution exists, it will be found.

Algorithm

1. Define a Stack of size total number of vertices in the graph.
2. Select any vertex as the starting point for traversal. Visit that vertex and push it on to the Stack.
3. Visit any one of the adjacent vertices of the vertex which is at top of the stack which is not visited and push it on to the stack.
4. Repeat step 3 until there is no new vertex to visit from the vertex on top of the stack.
5. When there is no new vertex to visit then use backtracking and pop one vertex from the stack.
6. Repeat steps 3, 4 and 5 until stack becomes Empty.
7. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph.

Application

DFS has a wide range of applications, including:

- figuring out every path that could lead between two nodes in a graph. Even if there are several paths connecting two nodes in a graph, DFS can be used to examine every path that could possibly exist.
- identifying a graph's connected elements. A slice of the graph in which every node can be reached from every other node is known as a linked component. In a graph, DFS can be used to locate every connected component.

- finding patterns in a graph. Cycles in a graph can be found using DFS, which is helpful for topological sorting and garbage collection applications.

- solving puzzles like Sudoku and mazes that only have one solution. DFS can be used to solve puzzles by recursively examining every avenue until it discovers the answer.

Suitability

DFS is a suitable option in situations where:

- It's critical to investigate the whole graph or discover every potential answer. DFS, for instance, can be used to locate every path out of a maze or every related element in a graph.
- The graph has a lot of sparseness. Because DFS does not have to store every node in memory at once, it is more effective than BFS for examining big and sparse graphs.
- Locating a solution deep within the graph is the aim. Compared to BFS, DFS is more likely to discover a deep solution since it fully pursues every avenue before turning around.

4.3 A* Search

Dijkstra's Algorithm is modified and optimized for a single destination by A*. While A* discovers pathways to a single location or the closest of multiple locations, Dijkstra's Algorithm can identify paths to all locations. Paths that appear to be getting closer to a goal are given priority.

By using Dijkstra's Algorithm, also known as Uniform Cost Search, we may order the paths we should investigate. Rather than investigating every avenue equally, it favor's the less expensive ones. We can set lower fees to incentivize driving on public highways, higher costs to deter adversaries, and so on. We utilize this instead of Breadth First Search when movement costs differ. The figure below should serve as an example. (Fig.7)

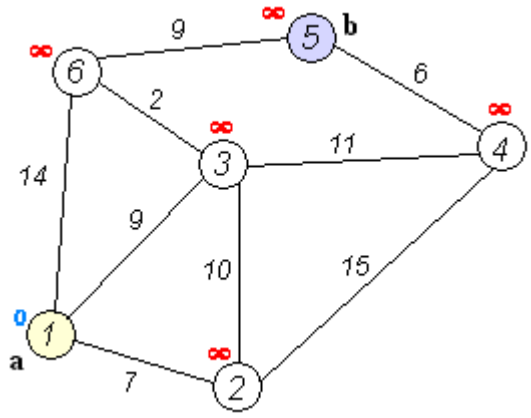


Figure 7

However, The A* Algorithm works as-

- It maintains a tree of paths originating at the start node.
- It extends those paths one edge at a time.
- It continues until its termination criterion is satisfied.

A* Algorithm extends the path that minimizes the following function-

$f(n) = g(n) + h(n)$

Here,

- 'n' is the last node on the path.
- g(n) is the cost of the path from start node to node 'n'.
- h(n) is a heuristic function that estimates the cost of the cheapest path from node 'n' to the goal node.

An Example of the A* is represented below in Fig.8.

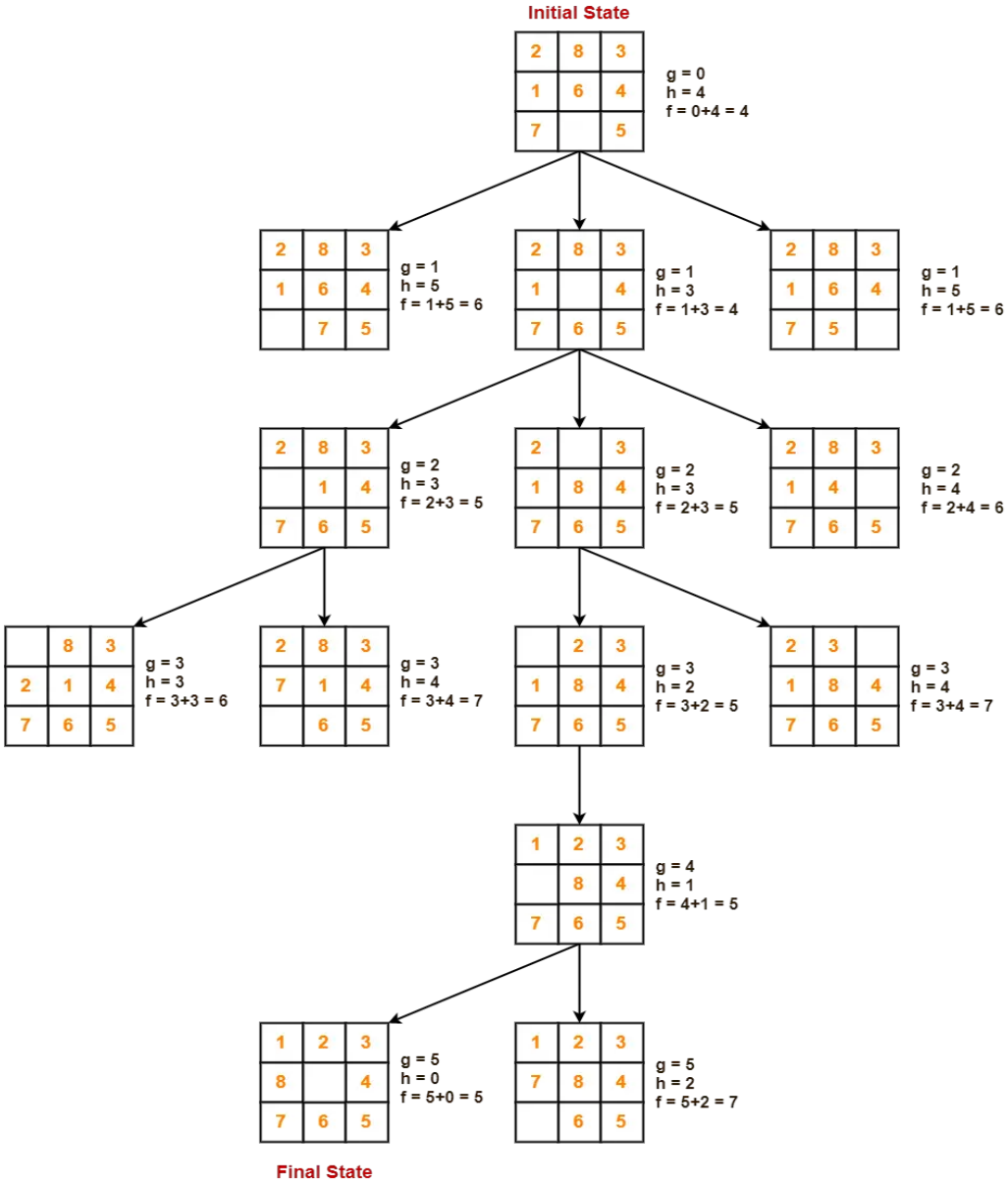


Figure 8

Application

Applications for A* search is numerous and include:

- Robotics and autonomous systems: Robot navigation uses A* search to identify the quickest and most effective route to a destination.
- In computer games, A* search is used to determine the direction non-player characters (NPCs) take and to assist players in real-time in determining the quickest and most effective route to an objective.
- Applications in geography: The shortest path between two points on a map can be found using the A* search method, which takes into consideration variables like elevation and traffic.

- Transportation and logistics: Vehicle scheduling and route issues are optimised through the use of A* search.
- Artificial intelligence: A* search is utilized in several artificial intelligence applications, including natural language processing and machine learning.

Suitability

The A* search is a useful option in situations where:

- Finding the ideal solution or one that is almost there in a fair amount of time is crucial.
- The field of search is broad and complex.
- The heuristic function yields precise and useful results.

A* search is not the best option in situations where:

- The heuristic function is either misleading or imprecise. This can lead to an inefficient exploration of a substantial area of the search space via A* search.
- Small and straightforward is the search space. Other search algorithms, such depth-first and breadth-first search, might be more effective in this situation.
- There is no explicit requirement for real-time. If a speedy result is not required, then alternative search methods might be more suitable.

4.4 Goal Directed Search

Goal-directed search (GDS) is a type of search that is specifically designed to find solutions to a particular problem or goal.

The informal algorithm we will follow:

goal: X

1. If X is achieved, finish.
2. If X is not achieved, find operations and the possible instantiations of these operations that will achieve the goal.
3. For each operation, check each of its preconditions to see if they are achieved.
4. For each precondition that is not achieved, find operations and the possible instantiations of these operations that will achieve the precondition.

5. Repeat 1 to 3 until a plan is found or it is evident that a plan cannot be found.

Application

When there is a specific aim in mind and a large and complex search space, GDS performs exceptionally well. GDS can be used, for instance, to determine the fastest route for a delivery vehicle, the shortest path between two points on a map, or the most effective way to schedule a series of tasks on a computer.

- Robotics and autonomous systems: GDS is utilized in robots and autonomous systems for navigation and pathfinding. A robot navigating through a complicated environment, such a warehouse or a disaster zone, can find its way from one place to another using a GDS algorithm.

- Computer games: In computer games, GDS is used to create paths for non-player characters (NPCs) and assists players in real-time in determining the quickest and most effective route to an objective. A GDS algorithm, for instance, can be used to assist a player in figuring out the fastest route to finish a task or to discover an NPC's path through a maze.

Suitability

Problems where goal-directed search (GDS) is a good option include:

- The domain of search is broad and complex. Large and difficult issues can be solved more quickly with GDS because it can prune the search space more effectively than other search methods.
- A specific objective is in mind. Faster solutions may result from GDS's ability to concentrate its search on the particular objective.
- Subproblems of the main problem can be separated out. Finding solutions to complicated problems may be made simpler by GDS's ability to break the problem down into smaller ones and solve them recursively.

5. Final decision

The robot must navigate through a corridor and variety of rooms, and doorways in this café robot problem. It is quite helpful to use a graph representation since it allows the model to accurately reflect these non-linear interactions between doors, rooms, and corridor.

The robot needs to understand the relationships between doors, corridors, and rooms to serve coffee to certain individuals. Graph representations are specifically designed for this use. Inside the graph, every room, doorway, and corridor are represented as nodes or vertices, and the connections between them are represented as edges. The robot is now able to effectively determine the best route between both locations through this structure.

The core of the cafe robot challenge is figuring out the quickest way to bring coffee to people who are dispersed throughout several rooms. Graphs are excellent at using pathfinding algorithms in this situation, and the A* search technique is a great option. This algorithm effectively determines the best path across the linked graph while taking potential barriers and distance into consideration.

With graphs, the robot is able to examine the relationships between different locations. Through the complexities of rooms, doorways, and corridors, it learns to determine the best route to take to arrive at its destination. The A* planning strategy stands out as a particularly successful method, taking advantage of the structure of the graph to quickly find the shortest path. It includes an estimate of the remaining cost to reach the ultimate target in addition to the present cost of getting to a certain area. This feature comes in very handy when navigating intricately connected areas.

In conclusion, the cafe robot skilfully delivers coffee to Teaching Assistants and the module leader while skilfully navigating across a network of rooms, entrances, and hallways by combining the A* planning approach with a graph representation. The flexibility and effectiveness of graph representations are effortlessly blended in this combination method, guaranteeing a precise and efficient solution that discovers the best routes.

6. References

1. https://www.thedshandbook.com/breadth-first-search/
2. https://www.thedshandbook.com/depth-first-search/
3. https://en.m.wikipedia.org/wiki/File:Dijkstra_Animation.gif
4. https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/?__cf_chl_rt_tk=gR10KK.sIg_.b_0x7klfEiGZlDNDgRKiIYp0wNK2G5s-1698689885-0-gaNycGzNEVA#google_vignette
5. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
6. Lecture 3.pdf
7. https://www.cs.cmu.edu/~mmv/papers/aaai96-yury.pdf