Below is the theoretical explanation for GPS agent problem as part of the Problem Statement1 from group 7 and team members are as follows:

* *AGHAV SAYALI SAKHARAM* (*2023AC05435*) – Contribution 100%
* *DULAL DAS* (*2023AC05041*) – Contribution 100%
* *JAWAHARLAL RAJAN S* (*2023AC05504*) – Contribution 100%
* *LAKSHMISRINIVAS PERAKAM* (*2023AC05540*) – Contribution 100%
* *SUBHRANSU MISHRA* (*2023AC05489*) – Contribution 100%

**PEAS of GPS agent:**

* **Performance measure(P):** The performance measure for the agent is to find the shortest and safest path to the destination (home) while avoiding roadblocks.
  + ***Safety*** is measured by the number of points accumulated along the path. Points are awarded for moving next to buildings and penalized for moving next to roadblocks.
  + ***Path Length:*** The path should pass through the minimum number of squares. The path should maximize the safety score.
  + ***Cost / Points Maximization:***

Add 5 points if the path passes adjacent to buildings.

Subtract 3 points if the path **passes** adjacent to roadblocks.

Subtract 3 points if agent moves diagonally (Only if no straight paths are available).

* + ***Efficiency:*** The agent should find the optimal path within a reasonable amount of time and computational resources (space and time complexity).
* **Environment(E):** The environment is the city grid, which includes buildings, roads, roadblocks, the starting point (office), and the destination (home).
  + ***The city grid*** - represented as a 2D list where:

‘O’ represents office.

‘B’ represents a building.

‘X’ represents a blocked path.

‘H’ represents Home.

* + ***Diagonal*** movements are allowed only when no straight path is available.
  + ***Static Conditions:*** The environment as well as the start and end position are static.
  + ***Agent’s Start and Goal Locations:*** Agent start at office and aim to reach home. This is represented in 2D grid as locations start = (0,0) and end = (5,5).
  + ***Penalties:*** Movement restrictions and penalties for diagonal movements and proximity to buildings and roadblocks are part of the environment.
* **Actuator(A):** The agent cannot directly affect the environment.
  + ***Movement Commands:*** The agent can move up, down, left, or right within the grid. Diagonal movements are allowed but penalized. Movement is constrained by the presence of buildings and roadblocks.
  + ***Steering* System (Motor/Actuator):**

**Example:** The car's steering system, controlled by an electric motor or hydraulic actuator, which turns the wheels to change the direction based on the agent's decisions.

**Function in Agent:** When the GPS agent decides on a direction (e.g., turn left, go straight), the steering system executes that command.

* + ***Throttle* and Brake Systems:**

**Example:** The car’s acceleration (throttle) and deceleration (brake) systems are controlled by actuators.

**Function in Agent:** The GPS agent might adjust the speed of the vehicle based on traffic conditions, proximity to hazards, or the need for precision in navigation.

* **Sensor(S):** The sensor perceives the state of the environment, including the location of buildings, roads, roadblocks, the starting point, and the destination. The agent can sense the current position and the status (open path, building, blocked path) of adjacent cells.
  + ***GPS* Receiver:**

**Example**: A GPS receiver that provides the vehicle’s current location, speed, and direction.

**Function in Agent:** It constantly updates the agent about the vehicle's real-time location, which is used to calculate the path and make navigation decisions.

* + **Cameras and LIDAR:**

**Example**: Cameras and LIDAR (Light Detection and Ranging) systems mounted on the vehicle that detect obstacles, road conditions, and the layout of the city grid (e.g., buildings, roadblocks, and open paths).

**Function** in Agent: These sensors help in identifying obstacles and road conditions, allowing the agent to avoid buildings, roadblocks, and detect adjacent hazards.

* + **Proximity Sensors:**

**Example**: Ultrasonic or infrared sensors that detect the distance to nearby objects.

**Function** in Agent: These sensors detect if the vehicle is close to buildings or roadblocks, which would add penalties to the path’s cost. They can also detect if a straight path is blocked, necessitating a diagonal move.

To solve this problem, we need to apply the given algorithms to find the shortest and safest path from the office to home on a grid with some blocked paths. Let's break down the steps for each algorithm:

**Uniform Cost Search (UCS) – Algorithm**

Uniform Cost Search (UCS) is a search algorithm which finds the least-cost path in a graph or grid where different paths may have different costs. UCS is optimal and complete, meaning it will always find the least-cost path if a path exists, and it will eventually explore all nodes if required.

**Algorithm -**

* **Initialize**: Start at the office location. Use a priority queue to keep track of nodes to explore, prioritized by safe and optimal path having minimum number of squares and maximum points.
* **Expand Node:** Dequeue the node with the minimum number of squares and maximum points.
* **Goal Test:** If the dequeued node is the goal (home), return the path.
* **Generate Successors:** For the current node, generate all possible successors (neighbouring nodes that are not blocked).
* **Update Costs:** For each successor, calculate the path cost and enqueue it in the priority queue if it has not been visited or if an optimal (with minimum number of squares and maximum points) path to it is found.
* **Repeat:** Continue until the goal is reached or the priority queue is empty.

Consider the problem grid as below:

S is the start (office), G is the goal (home), X represents roadblock, B represents building.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | O |  | X | X |  |  |
| 1 | B |  | X |  |  |  |
| 2 |  |  |  |  | X |  |
| 3 | B |  | X |  |  | B |
| 4 |  |  |  |  | B |  |
| 5 |  | B |  | X |  | H |

Start office (0 0); goal home (5 5)

| **Itertion** | **Open List/Frontiers/ Fringes** | **Closed List** | **Goal Test** |
| --- | --- | --- | --- |
| 1 | (0 0: 0) |  |  |
| 2 | (0 1: 1, -3) | (0 0) | Fail on (0 0) |
| 3 | (1 1: 2, -1) | (0 0) (0 1) | Fail on (0 1) |
| 4 | (2 1: 3, -1) | (0 0) (0 1) (1 1) | Fail on (1 1) |
| 5 | (2 0: 4, 9) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) | Fail on (2 1) |
| 6 | (3 1: 5, 8) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) | Fail on (2 0) |
| 7 | (4 1: 6, 13) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) | Fail on (3 1) |
| 8 | (4 0: 7, 18) (4 2: 7, 10) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) | Fail on (4 1) |
| 9 | (5 0: 8, 23) (4 2: 7, 10) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) | Fail on (4 0) |
| 10 | (4 2: 7, 10) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) | Fail on (5 0) |
| 11 | (4 3: 8, 12) (5 2: 8, 12) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) (4 2) | Fail on (4 2) |
| 12 | (5 2: 8, 12) (3 3: 9, 9) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) (4 2) (4 3) | Fail on (4 3) |
| 13 | (3 3: 9, 9) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) (4 2) (4 3) (5 2) | Fail on (5 2) |
| 14 | (3 4: 10, 16) (2 3: 10, 6) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) (4 2) (4 3) (5 2) (3 3) | Fail on (3 3) |
| 15 | (4 5: 11, 23) (2 5: 11, 18) (2 3: 10, 6) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) (4 2) (4 3) (5 2) (3 3) (3 4) | Fail on (3 4) |
| 16 | (5 5: 12, 23) (2 5: 11, 18) (2 3: 10, 6) (2 2: 4, -7) | (0 0) (0 1) (1 1) (2 1) (2 0) (3 1) (4 1) (4 0) (5 0) (4 2) (4 3) (5 2) (3 3) (3 4) (4 5) | Fail on (4 5) |
|  |  |  | Pass on (5 5) |

Here, total number of expanded nodes are 16.

**Iteration 1:**

Starting from (0 0) with zero number of squares and 0 points goal test fails.

**Iteration 2:**

Next possible square is (0 1) with one square and -3 points as adjacent to roadblock. Goal test fails.

**Iteration 3:**

Next possible square is (1 1) with one square (total = 1+1 = 2) and 5 points as adjacent to building; -3 points as adjacent to roadblock (total = -3+5-3 = -1). Goal test fails.

**Iteration 4:**

Next possible square is (2 1) with one square (total = 2+1 = 3) and 0 points (total = -1+0=-1). Goal test fails.

**Iteration 5:**

Next possible squares are (2 0) and (2 2). (2 0) with one square (total = 3+1 = 4) and 10 points as adjacent to two buildings (total = -1+10 = 9). (2 2) with one square (total = 3+1 = 4) and -6 points as adjacent to two blocks (total = -1-6 = -7). Since minimum number of squares and maximum points need to be selected, (2 0) is considered but goal test fails.

**Iteration 6:**

Next possible square is diagonal (3 1) with one square (total = 4+1 = 5) and 5 points as adjacent to building, -3 points as adjacent to roadblock, -3 points for diagonal move (total = 9+5-3-3 = 8). Goal test fails.

**Iteration 7:**

Next possible square is (4 1) with one square (total = 5+1 = 6) and 5 points as adjacent to building (total = 8+5 = 13). Goal test fails.

**Iteration 8:**

Next possible squares are (4 0) and (4 2). (4 0) with one square (total = 6+1 = 7) and 5 points as adjacent to building (total = 13+5 = 18). (4 2) with one square (total = 6+1 = 7) and -3 points as adjacent to roadblock (total = 13-3 = 10). (4 0) with maximum points is considered but goal test fails.

**Iteration 9:**

Next possible square is (5 0) with one square (total = 7+1 = 8) and 5 points as adjacent to building (total = 18+5 = 23). Goal test fails.

**Iteration 10:**

Next possible squares are (4 0) and (4 1). But both are visited so next is (4 2) with one square (total = 6+1 = 7) and -3 points as adjacent to roadblock (total = 13-3 = 10). But goal test fails at (4 2).

**Iteration 11:**

Next possible squares are (4 3) and (5 2). (4 3) with one square (total = 7+1 = 8) and 5 points as adjacent to building and -3 points as adjacent to roadblock (total = 10+5-3 = 12); similar is for (5 2). Both have same number of squares and points. Let us consider (4 3) but goal test fails at (4 3).

**Iteration 12:**

Next possible square is (3 3). With one square (total = 8+1 = 9) and-3 points as adjacent to roadblock (total = 12-3 = 9). As there is already a block (5 2) in the queue with higher cost than (3 3); it will be considered but goal test fails at (5 2).

**Iteration 13:**

Next possible square is (4 2) but its already visited. So next block in the queue (3 3) will be considered but goal test fails at (3 3).

**Iteration 14:**

Next possible squares are (3 4) and (2 3). (3 4) with one square (total = 9+1 = 10) and 10 points as adjacent to two buildings; -3 points as adjacent to roadblock (total = 9+10-3 = 16). (2 3) with one square (total = 9+1 = 10) and -3 points as adjacent to roadblock (total = 9-3 = 6). Block with higher cost will be considered but goal test fails at (3 4).

**Iteration 15:**

Next possible square with straight move is only (3 3) but its already visited. Possible diagonal moves are (4 5) and (2 5). (4 5) with one square (total = 10+1 = 11) and 10 points as adjacent to two buildings, -3 points for diagonal movement (total = 16+10-3 = 23). (2 5) with one square (total = 10+1 = 11) and 5 points as adjacent to building, -3 points as adjacent to roadblock (total = 16+5-3 = 18). Block with higher cost will be considered but goal test fails at (4 5).

**Iteration 16:**

Next possible square is (5 5) which home with one square (total = 11+1 = 12) and 0 points (total = 23+0 =23). Goal test passed.

**Time and Space Complexity:**

Time and space complexity of uniform cost search does not depend upon branching or depth of the queue as it is more concerned about the cost. Given by: O(b(1+C/ε)).

b: The branching factor, representing the maximum number of successors for any node.

C: The cost of the optimal solution.

ε: The minimum cost between any two states.

Output:

A screenshot of a computer

Description automatically generated



**Reverse Best First Search (RBFS) Algorithm:**

Recursive Best-First Search (RBFS) is a memory-efficient variant of the A\* search algorithm. It combines the depth-first search (DFS) approach with heuristic-guided search, similar to A\*. RBFS operates recursively, maintaining a linear space complexity by storing only the path being explored and a small amount of additional information for backtracking.

**Algorithm:**

* **Initialize**: Start at the office location.
* **Heuristic Function**: Define a heuristic function h(n) that estimates the cost from node n to the goal (home). This could be the Manhattan distance if we assume each move has a uniform cost.
* **Recursive Search**:
  + Expand the current node.
  + Sort the successors by their estimated total cost f(n)=g(n)+h(n), where g(n) is the cost to reach node n from the start.
  + Recursively apply RBFS to the best successor.
  + If the best successor's cost exceeds the current best alternative, backtrack and explore the next best alternative.
* **Termination**: The search terminates when the goal (home) is reached or all paths are explored.

In pathfinding algorithms, the choice of heuristic can significantly impact the efficiency and accuracy of finding the optimal path. The two commonly used heuristics are the *Manhattan distance* and the *Euclidean distance*.

**Key Differences between Manhattan and Euclidean distance:**

|  |  |  |
| --- | --- | --- |
|  | Manhattan Distance | Euclidean Distance |
| Definition | The Manhattan distance between two points (x1, y1) and (x2, y2) is calculated as |x1 - x2| + |y1 - y2|. It represents the sum of the absolute differences in the horizontal and vertical directions. | The Euclidean distance between two points (x1, y1) and (x2, y2) is calculated as sqrt((x2 - x1)^2 + (y2 - y1)^2). It represents the straight-line distance between the two points. |
| Applicability | The Manhattan distance is best suited for grids where movement is restricted to orthogonal (straight) paths—i.e., only horizontal and vertical movements are allowed. | The Euclidean distance is more appropriate for grids where diagonal movements are allowed, as it directly measures the shortest path between two points. |
| Movement Types | Assumes only horizontal and vertical movement is allowed. It does not account for diagonal paths, which makes it less effective in scenarios where diagonal movement is possible. | Takes into account diagonal movement by calculating the direct straight-line distance, making it more versatile for grids allowing both straight and diagonal movements. |
| Pathfinding Accuracy | In scenarios where diagonal movement is allowed, using Manhattan distance can lead to suboptimal paths because the heuristic underestimates the true distance when diagonal paths are available. | Provides a more accurate estimation of the true path cost, especially when diagonal movements are involved, leading to more efficient and accurate pathfinding. |
| Heuristic Admissibility: | Always admissible (it never overestimates the true cost) in grids where only straight movements are allowed. | Also admissible and consistent in scenarios where diagonal movement is allowed because it never overestimates the true cost of the path. |

**Conclusion:** For this GPS Navigation Agent problem in a city grid where both straight and diagonal movements are allowed, the *Euclidean distance heuristic is preferred*. It more accurately represents the possible paths the agent can take, leading to more efficient and reliable pathfinding compared to the Manhattan distance.

**Advantages and Disadvantages:**

* **Advantages**:
* **Memory Efficiency**: RBFS has a space complexity of O(bd), where b is the branching factor and d is the depth of the solution. This is much lower than A\*'s O(b^d).
* **Heuristic Guidance**: Like A\*, RBFS uses heuristics to guide the search, potentially reducing the number of nodes expanded compared to uninformed search strategies.
* **Disadvantages**:
* **Revisiting Nodes**: Since RBFS uses backtracking, it may revisit nodes multiple times, leading to potentially higher time complexity than A\*.
* **Optimality**: While RBFS is complete and optimal if the heuristic is admissible and consistent, it might not perform as well as A\* in terms of speed due to the need to backtrack.

**Time Complexity:**

* **Worst-Case Time Complexity**: The time complexity of RBFS is similar to that of A\* but with some improvements due to its depth-first nature: O(b^d)

However, since RBFS uses backtracking and might need to re-explore branches due to its limited memory usage, the actual number of nodes expanded could be higher than A\* in some cases.

**Space Complexity:**

* **Worst-Case Space Complexity**: The key advantage of RBFS over A\* is its lower space complexity. RBFS only needs to store a linear number of nodes relative to the depth of the search: O(bd) where:
  + b is the branching factor.
  + d is the maximum depth of the search.

**Output**:

A screen shot of a computer

Description automatically generated

