MARTIAN ROVER NAVIGATION SYSTEM

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE EVALUATION OF MID-SEMESTER COMPONENT, SEMESTER 4

OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Submitted By:

ABHILASHA BANSAL, 2K19/CO/014

ANSHUMAN RAJ CHAUHAN, 2K19/CO/067

Under the supervision of

Mr. Sanjay Kumar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

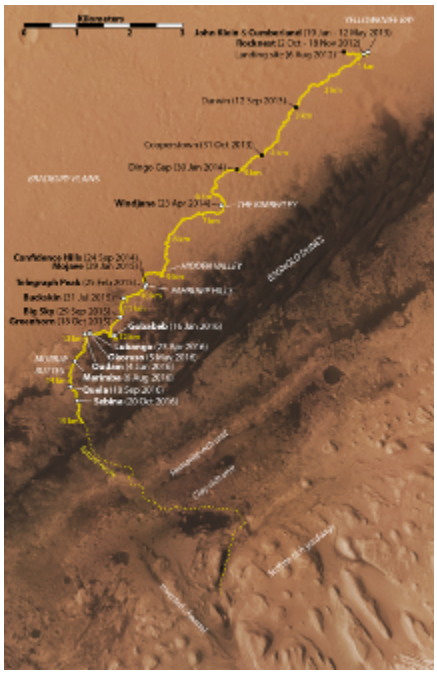
(Formerly Delhi College of Engineering)

Bawana Road, Delhi, 110042

May 2021

ABSTRACT

Humans have always been very curiosity driven. This drive makes us want to explore more, learn more and understand more about the world we have around us. This quest has not just been limited to the Earth now.



Curiosity's lifetime journey

Most recently space exploration has been at the forefront of this human endeavour, and Mars at the centre of it. Be it NASA, ISRO, SpaceX or any other space agency across the globe, everyone invariably has plans to land their rovers on Mars and understand the red planet in ways we don't yet do.

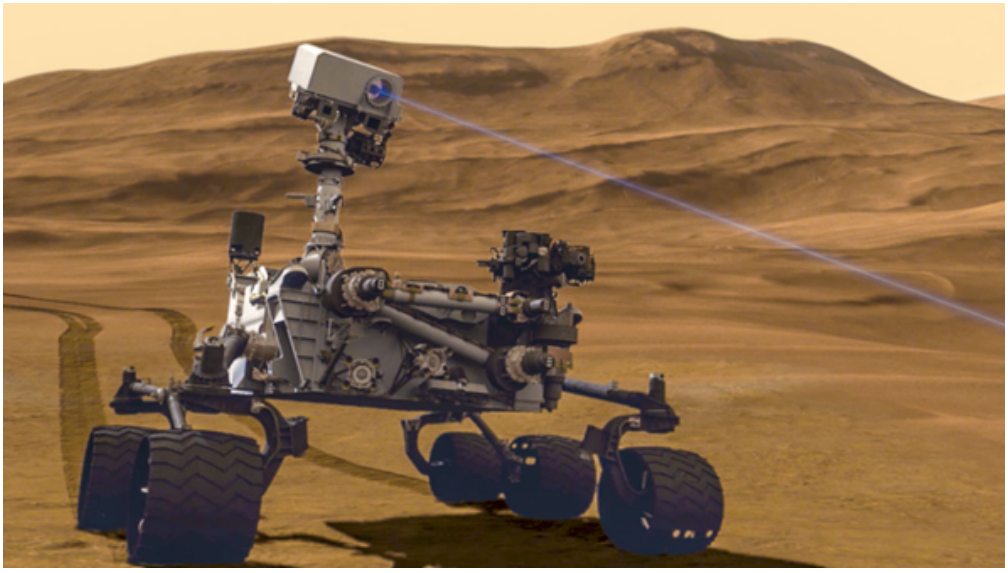
This project aims to provide a way to maximise the research done through these rovers. The system we have developed can find paths with maximum research potential (that can also be traversed within constrained time), given the values of research potentials for various points and the time taken to traverse between them, in the form of the vertices and edges of a graph.

TABLE OF CONTENTS

PROBLEM STATEMENT

Navigation on Martian surface for Martian Rovers is a challenging task. While a complete Martian day (called a sol) is about 24 hours and 40 minutes long (or 24 hours 37.5 minutes to be precise), the Sun can only provide enough power for driving during a four-hour window (around high noon). **This means the rovers have to be able to move quickly and effectively.**

Moving safely from rock to rock or location to location is a major challenge because of the communication time delay between Earth and Mars, which is about 20 minutes on average. Unlike a remote controlled car, the drivers of rovers on Mars cannot instantly see what is happening to a rover at any given moment and they cannot send quick commands to prevent the rover from running into a rock or falling off of a cliff.



Upon landing onto the Martian surface initially, the rover is going to need to communicate with the engineers on Earth to take a quick scan of the surrounding area, identify hazardous spots & create maps to guide its journey. It will also need to identify the spots with good research potentials. Once all that work is done, the rover can save time & energy by moving on the paths with maximum research potential & less hazards. The hazards & research points can be identified with the help of some Computer Vision & Deep Learning techniques.

This project aims to develop an algorithmic system that can help to determine the paths with maximum research potentials that can also be traversed in constrained time, so as to help the rover maximize its research, and spend less time traversing the surface.

This would help engineers & scientists to make an informed decision on how to guide the rover further. Less time in traversing the surface would also potentially lead to less wear & tear for the rover.

INTRODUCTION

During surface operations on Mars, each rover receives a new set of instructions at the beginning of each sol (a Martian day). Sent from the scientists and engineers on Earth, the command sequence tells the rover what targets to go to and what science experiments to perform on Mars. The rover is expected to move over a given distance, precisely position itself with respect to a target, and deploy its instruments to take close-up pictures and analyze the minerals or elements of rocks and soil.

This guidance from the people on Earth is crucial & necessary for the safety of the rover. Initially after landing, the rover can make use of some machine learning & deep learning algorithms so as to locate the obstacles and the valuable research points of the Martian surface. Then the scientists & engineers back at Earth, can help to create approximate maps to help guide the rover. Once constructed, the map would help the rover select safe movements to the destination and prevent it from encountering obstacles already avoided during prior segments of the drive. The data collected during these initial drives can be sent back to Earth to create a master map / graph on which engineers can then run an algorithm to determine the paths with maximum research potential. We have presented one such algorithm in this project.

Also, if there comes a time when humans actually start inhabiting Mars, we can use this algorithm with even more ease to guide the quests of the rover, after an initial scan/ exploration drive to identify research points & obstacles

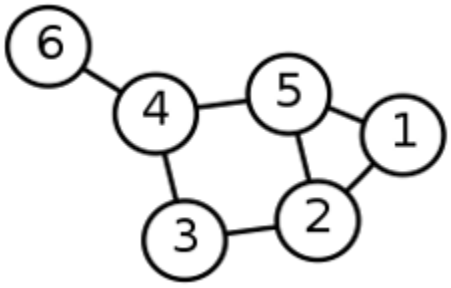
Using an algorithm to determine optimum paths to traverse for maximum research within given constraints of fuel & time would ensure more efficient exploration and could potentially make the rover autonomous after initial scan. It would also allow for less wear & tear of wheels, thereby increasing the rover life.

PRELIMINARY/ PREREQUISITE CONCEPTS

Our solution has been built on top of these following concepts.

1. Graphs

   In Graph Theory, a graph represents a structure that describes the relation between pairs of objects in a set. Each object in the graph is referred to as a vertex and each pair of related objects forms an edge.



2. Weighted Graph

   A weighted graph is a graph where edges are assigned values (weights). They may be used to describe the various parameters associated with each edge.

3. Path

   A path in a graph is a walk where none of the vertices and edges are repeated.

4. Recursion & Backtracking

   Recursion is a technique where we solve a problem through a function that calls itself, until it reaches a base case, for which we already know the solution. In backtracking, we use recursion in order to explore all the possibilities till we get the best result for the problem.

   Backtracking can be applied effectively for problems which allow the concept of a "partial candidate solution" and a relatively quick test of whether it can possibly be completed to a valid solution. When it is applicable, however, backtracking is often much faster than brute force enumeration of all complete candidates, since it can eliminate many candidates with a single test.

5. Floyd Warshall Algorithm

The Floyd-Warshall Algorithm is used to find the shortest path between all pairs of vertices in a weighted graph. It does so by comparing each path between each pair of vertices at a time. It has a time complexity of Θ (n^3), which isn't very good but once executed, query for any pair of nodes can be executed in O(1) time.

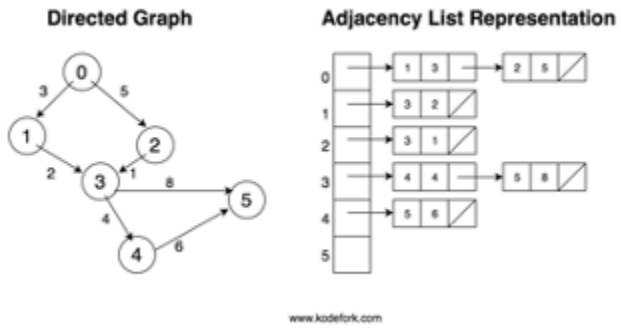
Pseudocode:

```
//pseudoCode(0.1)

let dist be a |V| × |V| array of minimum distances initialised to ∞
(infinity)
for each edge (u, v) do
      dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
      dist[v][v] ← 0
for k from 1 to |V|
      for i from 1 to |V|
      for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                  dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

6. Adjacency List

An adjacency list is a collection of lists used to represent a finite graph. Each list corresponds to one vertex, and contains a number of ordered pairs storing the neighbouring vertex and the edge weight between those two.



7. Bit Masking

   Bit-masking is a way of representing a set and allows O(1) removal and addition of elements into the set.

   For any integer n, it can be represented as a sequence of bits (0 and 1). Hence, for a given array, we can say the i'th element is in the set if set(i) is 1. Else, the element does not belong to the set.

   Hence, this representation can be used to represent sets from a given array, and since setting and unsetting bits takes O(1) time, elements can be added or removed in O(1) time.

The next section discusses our proposed algorithm, which is built on these concepts in detail.

PROPOSED SOLUTION

To solve the discussed problem statement, we require an algorithm for profit maximization, within given constraints, in a graph where both vertices and edges have weights. Graphs where vertices also have weights have not been explored in much detail, and there is no standard algorithm to do this task. Hence, we have built the solution using recursion & backtracking, which are generic logical concepts that can be applied to any problem whose solution can be found by finding the solution to one or more sub problems.

ALGORITHM TO DETERMINE MAXIMUM RESEARCH PATH

Assumptions

Our solution makes the following **assumptions**.

1) The research potential of various points will be given as input to the system, and these points will form the weighted vertices of a graph. These values will have to be estimated with help of Computer Vision and/ or Deep Learning techniques.
2) The time taken to travel between these vertices will also be given as input to the system.
3) The time taken to travel between vertices will be calculated according to the terrain of that particular area, and will be directly proportional to the fuel consumption for the same path. This means that if there is a hill or bump on the surface between two vertices, the time input corresponding to that edge would be more, even though the absolute geographical distance between them may be more.

System Input

- Research points as vertices and their estimated research potentials as vertex weights
- Time taken to traverse between vertices as edge weights
- Source vertex

- Destination vertex
- Time (or fuel) constraint

Algorithm

Let's assume the source vertex to be ‘A’, destination vertex to be ‘B’ & time constraint to be ‘T’.
In simplest words, our algorithm uses recursion along with backtracking to explore all possible paths from A to B and finds the most optimum path. We define the most optimum path as one that facilitates maximum research possible within given constraints of time/ fuel, source & destination vertex.

We define a recursive function.
To find the optimum path from ‘A’ to ‘B’, ‘A’ will ask its adjacent vertices for the path with maximum research from them to ‘B’ and one that does not exceed the time constraints.

This means, for each neighbouring vertex in the adjacency list of ‘A’ that is not already present in the current path, we will recursively apply this algorithm to find an optimal path from that vertex to the destination vertex, and build our answer as the best of all possible paths through these vertices. The base case would be when we reach the destination vertex itself.

The path with maximum research among all paths would be chosen and returned as the output. This is similar to a standard Depth First Search in a graph, but has more conditions & checks to ensure selection of optimal path.

A boolean array/ bit-mask can be used to ensure that none of the vertices get repeated in the process and a state variable ‘spareTime’ has to be passed in every successive call to backtrack as soon as remaining time becomes less than 0 (ie, time constraint gets violated). This way we can reduce some of the function calls and hence effective running time by identifying paths that are not feasible under given conditions.

Pseudocode:

```
//pseudoCode

Let optPaths be an array that stores optimal paths to reach the final
vertex 'B' from current vertex, ie, optPaths[x] stores <researchDone,
spareTime, ListOfVertices in optimal path from 'x' to 'B'>

1 Initialise optPath for each vertex to have spareTime = -2, researchDone
= -2 and listOfVertices = [].

2 Initialise a bit-mask 'mask' as 0(indicating no vertices have been
visited yet in the current path).

Call the traverse function and pass initial vertex as 'A', time
constraint, final vertex as 'B', 'mask', 'optPaths' as function
parameters.

This means:
3 Set variable 'totalResearch' = traverse('A', timeConstraint, 'B', mask,
optPaths), where traverse is the recursive function that finds the optimal
path.

Description of traverse function-
It returns the researchDone in traversal from currentVertex to finalVertex
with following fn parameters & fn body:

researchDone traverse(currentVertex, spareTime, finalVertex, mask,
&optPaths){
      Set 'curr' <- currentVertex;
      If spareTime < 0 (ie, time constraint exceeded):
            // ie no research possible
             return -2;
      If curr == 'B':
            // destination has been reached
            Set optPaths[curr].researchDone <- researchPotential[b]
            Set optPaths[curr].spareTime <- spareTime
            Set optPaths[curr].listOfVertices <- [B]
            return optPaths[curr].researchDone

      Set 'maxResearchDone' <- -1;
      Set 'maxSpareTime' <- INT_MAX;
      Set 'dummy' <- -1;

      For every adjacent vertex 'nbr' in adjacency list of curr:
            Set 'edge' <- edge weight b/w 'curr' and 'nbr';
```

```
            If mask & (1 << curr) == 0:
                  // 'nbr' not in current path/ not visited
                  // Call traverse function recursively

                  Set dummy <- traverse(nbr, spareTime - edge, finalVertex
'B',  mask | (1 << pos), optPaths)

                  If (dummy > maxResearchDone) or (dummy == maxResearchDone
and maxSpareTime < optPaths[nbr].spareTime):
                        // either the research done is greater than all
prev results, or it is equal to prev maximum result, but with more spare
time at hand
                        Set maxResearchDone <- dummy
                        Set maxSpareTime <- optPaths[nbr].spareTime
                        Set optPaths[curr] <- optPaths[nbr]
                        Append 'curr' to optPaths[curr].listOfVertices
                        Set optPaths[curr].researchDone =
optPaths[nbr].researchDone + researchPotential[curr]

      If maxResearchDone == -1:    (
            // no path possible to 'B' from current vertex in given
constraints
            return -2;
      Else
            return optPaths[curr].researchDone

 }

Since optPaths was passed by reference, we can get access to the optimum
path from 'A' to 'B' by 'optPaths[A]'.

4 Print the following:
'Research Done' = optPaths[A].researchDone
'Time taken for traversal' = timeConstraint - optPaths[A].spareTime
'Optimum Path' = optPaths[A].listOfVertices
(list should be printed in correct order as per code)
```

A small optimization:

A small optimization can be done to this existing recursive algorithm by using the Floyd Warshall algorithm initially on the entire graph. Using this algorithm, we can predetermine the minimum amount of time taken to traverse from one vertex to another, for every pair of vertices.

This way, whenever a query for a source vertex ‘A’, destination vertex ‘B’, and time constraint ‘T’ comes, we can determine in constant (O(1)) time, if it's at all possible to go from ‘A’ to ‘B’ within given time constraint. If it's not possible, we can straightaway output “No path possible” without running the algorithm. However this would not improve the overall upper bound of the solution’s time complexity.

Standard Concepts Used

- Recursion & backtracking
- Floyd Warshall Algorithm

System Output

- System outputs the path from initial to final vertex that facilitates maximum research within a given constraint of time/ fuel.
- If such a path is not possible due to absence of edges from initial to final vertex or due to lack of sufficient time, then the system gives a “No path possible” prompt.

Time Complexity

- The time complexity will highly depend on the time constraint as less time constraint will render the system to reach at the non promising nodes of the state space tree sooner.
- Floyd Warshall algorithm: O(V^3)
- Queries where no path is possible: O(1)
- The worst case complexity will be exponential. Although, in a system like Mars, the number of vertices will be fairly limited, therefore making the algorithm effective.

Space Complexity

The overall space complexity of our solution is O(V^2).

Limitations

- This is a recursive algorithm, hence its efficient execution on large graphs may depend on the allowed depth of the recursive call stack of device & programming language.
- This solution can potentially be improved in the future with the help of Dynamic Programming.

Points to Note

1) This algorithm can be used to **determine the path** to maximize the research done through a rover and minimize time spent travelling between these points, and wear & tear of rover.

2) Some other standard path planning algorithms such as the ‘A Star’ algorithm can be used for actually travelling between research points in the path determined by our algorithm.

3) Also, ‘A Star’ algorithm can be used along with hazard avoidance software while taking initial scans to form approximate maps of the surface, on which our algorithm can then be run.

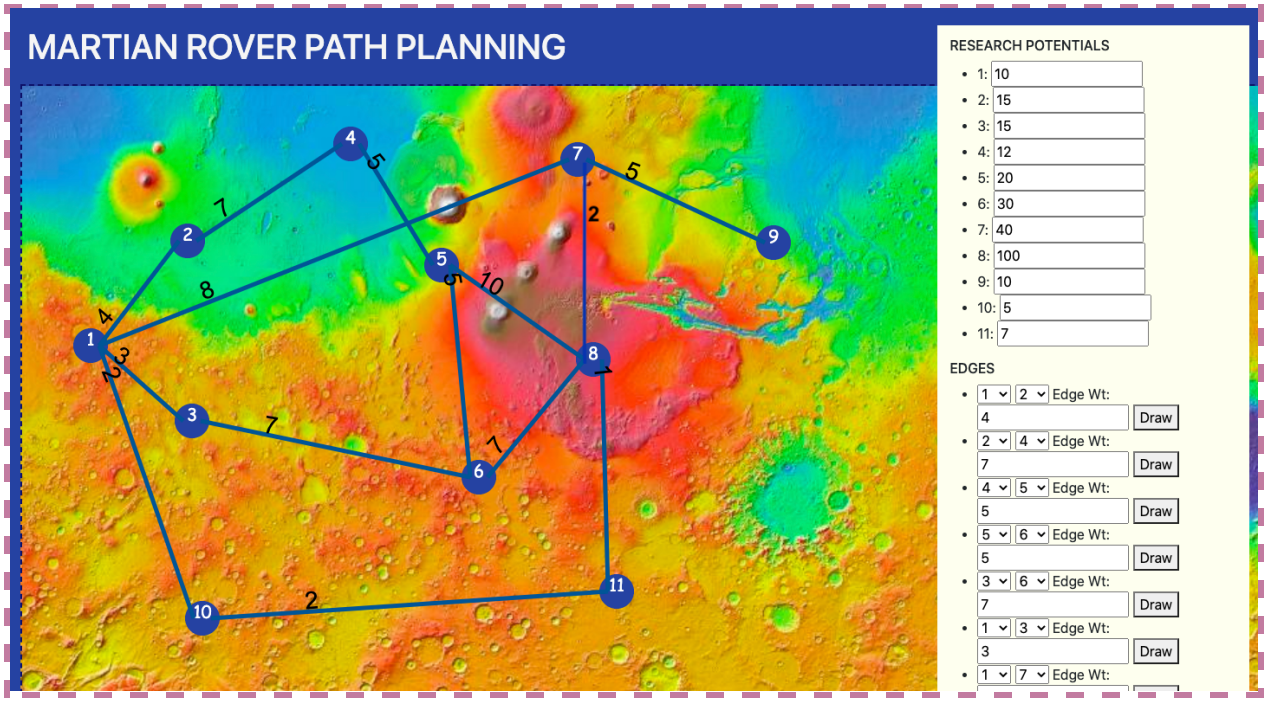
4) All of this can together help to make the rover autonomous to some extent, once initial work has been done. The rover would be able to function with minimal help from humans.

5) We have simulated both our algorithm as well as the A Star algorithm in our project.

SIMULATION & TESTING

We have implemented this algorithm in C++. We have also implemented it in JavaScript and created a static webpage to facilitate the simulation & testing of the system.

This is the simulation of the input graph on our website.



Test 1

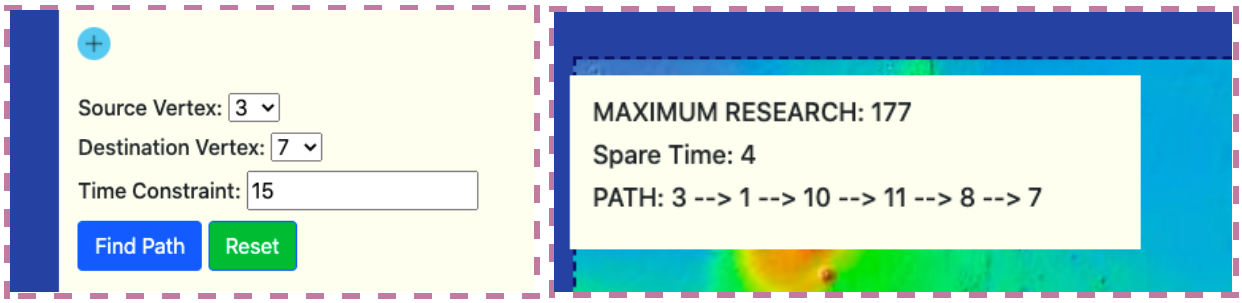
Input:

➢ src = 3, dest = 7, time constraint = 15

Output obtained:

➢ max research = 177,

➢ path to be followed = 3 -> 1 -> 10 -> 11 -> 8 -> 7



Test 2

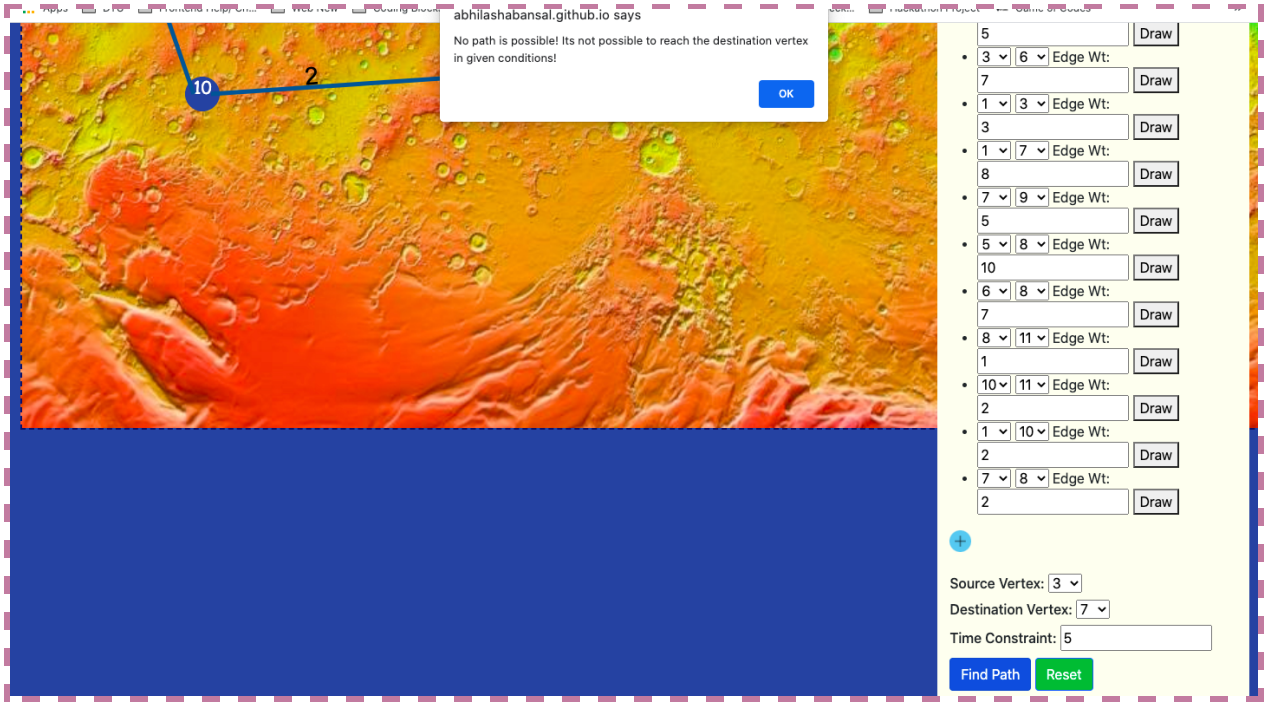
Input:

➢ src = 3, dest = 7, time constraint = 5

Output obtained:

➢ No path possible

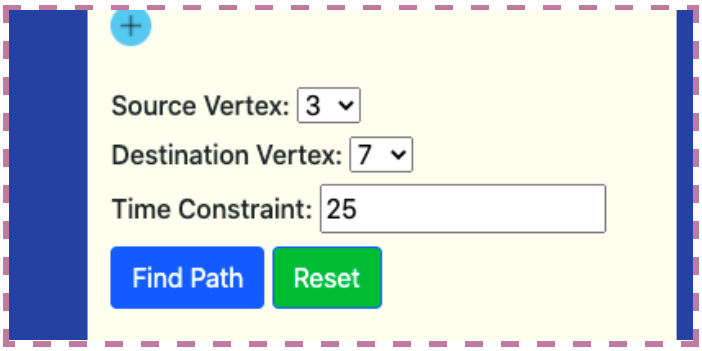
(The webpage gives a prompt that a path from source to destination vertex is not possible within given constraints.)



Test 3

Input:

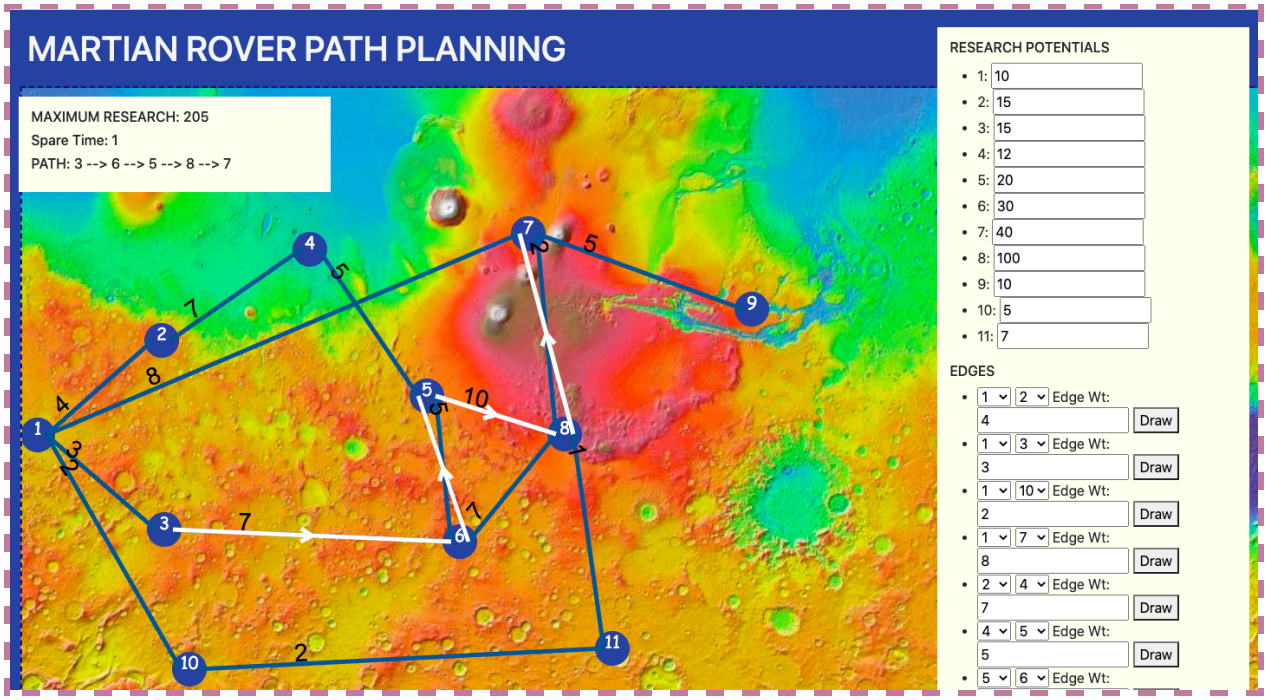
➢ src = 3, dest = 7, time constraint = 25



Output obtained:

- ➢ max research = 205
- ➢ path to be followed = 3 -> 6 -> 5 -> 8 -> 7

  The path to be followed has been displayed in white, as visible in the image below.

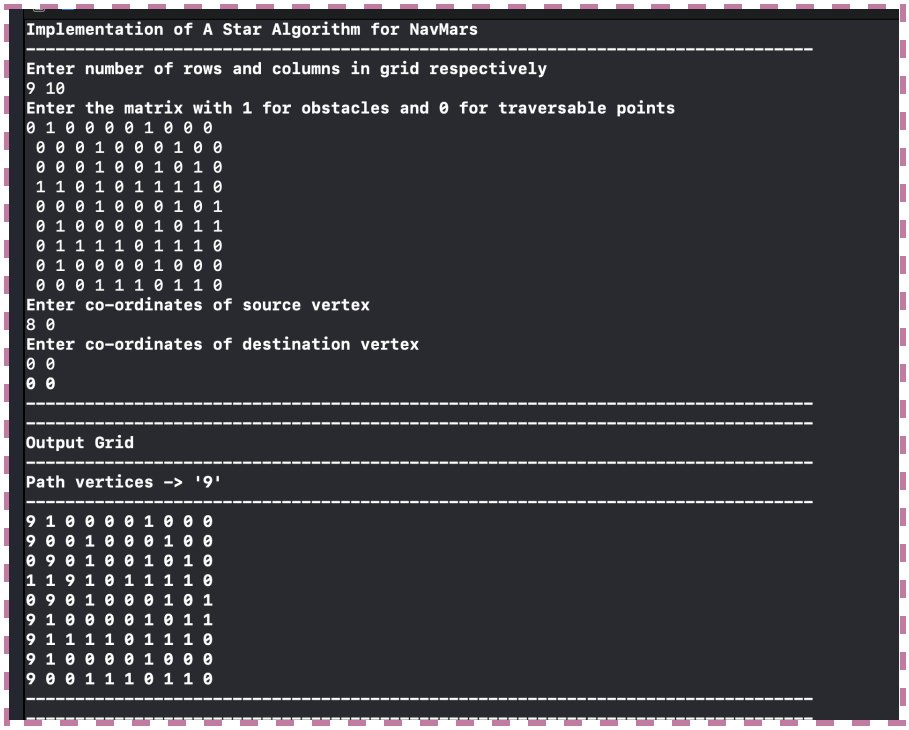


All the tests ran successfully and the results matched the results derived by us manually.

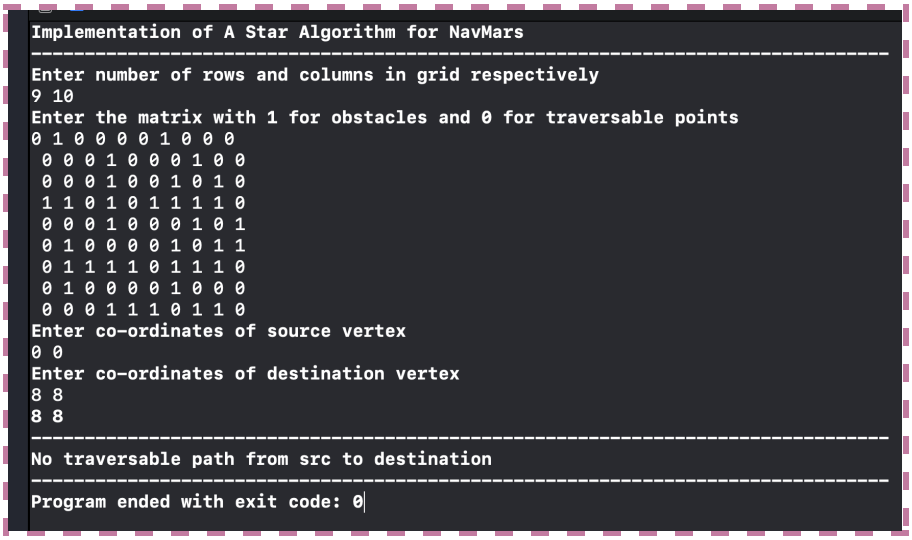
The webpage has been deployed on GitHub Pages through the following repository on GitHub.
https://github.com/AbhilashaBansal/ada_project

We have also coded the A Star algorithm and simulated it in C++. Following are the screenshots of the same.

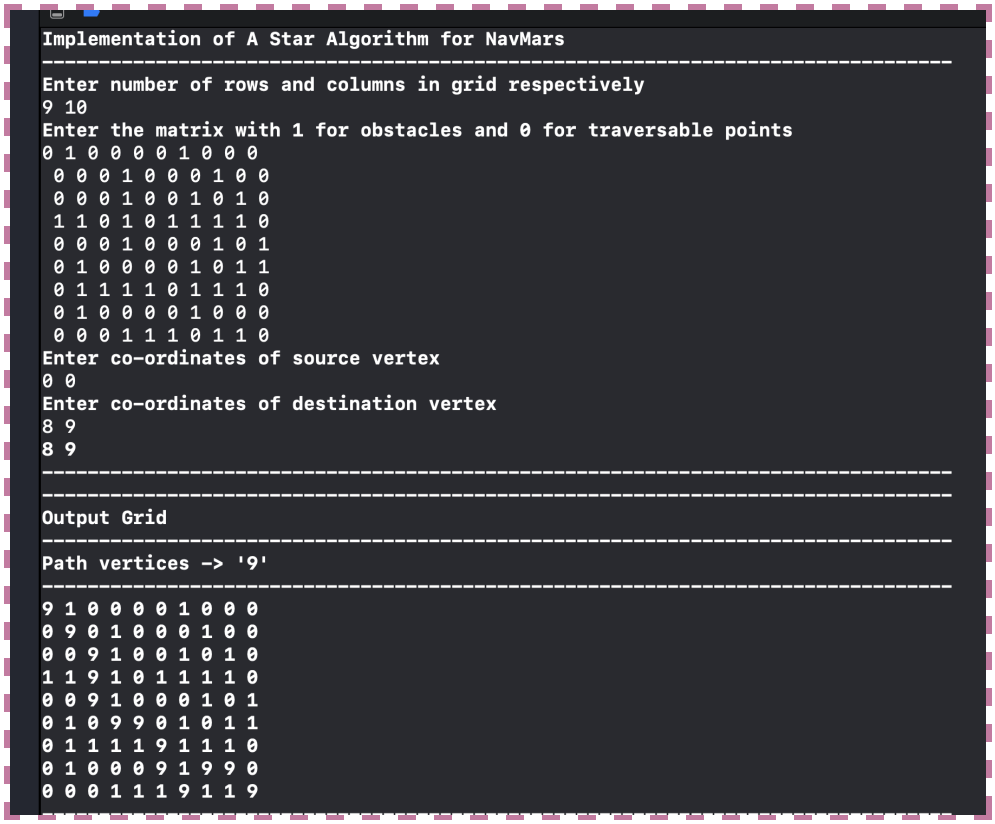
Test 1



Test 2



Test 3



```
Implementation of A Star Algorithm for NavMars
-------------------------------------------------------------------------------
Enter number of rows and columns in grid respectively
9 10
Enter the matrix with 1 for obstacles and 0 for traversable points
0 1 0 0 0 0 1 0 0 0
 0 0 0 1 0 0 0 1 0 0
 0 0 0 1 0 0 1 0 1 0
 1 1 0 1 0 1 1 1 1 0
 0 0 0 1 0 0 0 1 0 1
 0 1 0 0 0 0 1 0 1 1
 0 1 1 1 1 0 1 1 1 0
 0 1 0 0 0 0 1 0 0 0
 0 0 0 1 1 1 0 1 1 0
Enter co-ordinates of source vertex
0 0
Enter co-ordinates of destination vertex
8 9
8 9
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Output Grid
-------------------------------------------------------------------------------
Path vertices -> '9'
-------------------------------------------------------------------------------
9 1 0 0 0 0 1 0 0 0
0 9 0 1 0 0 0 1 0 0
0 0 9 1 0 0 1 0 1 0
1 1 9 1 0 1 1 1 1 0
0 0 9 1 0 0 0 1 0 1
0 1 0 9 9 0 1 0 1 1
0 1 1 1 1 9 1 1 1 0
0 1 0 0 0 9 1 9 9 0
0 0 0 1 1 1 9 1 1 9
```

Source Code:

The source code for the algorithm & all the simulations can be found in this GitHub repository.
https://github.com/its7ARC/martianRoverNavigationSystem

RESULT & CONCLUSION

The algorithm was simulated and tested on the webpage and the results were found to be accurate. Hence, this system can be implemented at scale and can be used to help the rovers move quickly & effectively.

The algorithm works under a set of conditions/ assumptions, and uses recursion & backtracking fundamentally, to arrive at the optimal solution. The time complexity for the same is exponential, however, it can be improved by use of Dynamic Programming.

Such a system could potentially help engineers & scientists to make an informed decision on how to guide the rover further. Using such an algorithm to determine optimum paths to traverse for maximum research within given constraints of fuel & time would ensure more efficient exploration and could potentially make the rover autonomous after an initial scan. It would also allow for less wear & tear of wheels, thereby increasing the rover life.

REFERENCES

1. Geeksforgeeks - Floyd Warshall Algorithm
   https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
2. Wikipedia - Graphs
   https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
3. Geeksforgeeks - Graph & Its Representations
   https://www.geeksforgeeks.org/graph-and-its-representations/
4. Exploring Vertex and Edge Weighted Graphs
   http://bert.stuy.edu/pbrooks/spring2009_old/materials/Intel-Papers/KennyYu1.pdf
5. https://stackoverflow.com/questions/10453053/graph-shortest-path-with-vertex-weight
6. https://www.w3schools.com/js/
7. https://developer.mozilla.org/en-US/docs/Web/JavaScript
8. Canvas Element
   https://www.w3schools.com/html/html5_canvas.asp