Algorithm for GMP1R with Single Hole

Jagrati Singh, Anmol Darak, Sanket S. Dash and Kalpesh Kapoor

***Abstract*-- In this paper, we are given an undirected, unweighted, connected simple graph with n vertices. We have a movable robot and a hole at two distinct vertices, and non-distinct obstacles at the rest of the vertices. Each vertex holds either a robot or an obstacle or is empty. The empty vertex is said to have a hole. In one step the robot or an obstacle is free to move along an edge to reside at an adjacent vacant vertex containing hole. We analyze the motion planning problem where a robot initially at some given vertex, is required to be taken to a particular destination vertex. We describe a configuration graph approach to solve this type of problem and present $O(n^3)$ algorithms with similar backgrounds.**

Index Terms-- ***Configuration graph, Robot, Hole, Breadth first search***

I. Introduction

CONSIDER a situation where we have a railway network with each station capable of holding a single train or no train. Among n stations, $(n-1)$ stations hold distinct trains with only one station empty and each route can hold only one train as can be seen in Figure 1. The network expects to move a particular train, say R, initially at station s, to a particular destination station, say t, by making other trains move to clear path for it in the shortest possible number of moves possible as shown below.

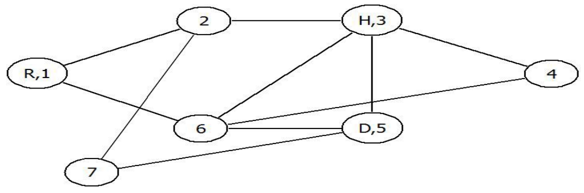

Fig. 1. Example of a typical train management situation

This situation is a particular type of railways management problem but in this report we study this type of problem but with all trains as similar (non-distinct), and train R expected to go from s to t. This problem is modeled in form of a graph with stations as vertices, routes as edges, train R as a robot R, empty station considered as a hole (H) and all the rest trains as obstacles. We are required to move R from initial vertex s to destination vertex t with the help of one hole and R, and obstacles are free to move one at a time across edges and each vertex holding a single entity (R, H or obstacle). We study this type of **robot motion planning** in detail as described ahead.

Given a connected, undirected, unweighted graph $G = (V, E)$ where V is the set of vertices and E is the set of edges connecting them in the graph G. Our objective is to move the robot R (initially at source vertex s) to destination vertex t with the smallest number of moves, where a move consists in moving an object (robot or obstacle) from one vertex to an adjacent vertex that contains a hole (if a vertex does not contain an object, then it is said to contain a hole; if an object is moved from vertex x to y, we can also say that a hole is moved from y to x). This problem was introduced by Papadimitrou et al. [2].

Before considering the optimization problem, the decision problem whether a given instance of the problem is feasible or not, should be considered first which is taken up inside Section II, which also deals with some properties related to our problem. In Section III, we present an algorithm that optimizes the total number of moves required to execute the problem in different approaches. In between we inspect the time complexity of the provided algorithms with insights into further modifications. Finally in Section IV, we present our conclusion and some related issues for further research.

II. Shortest Path Properties

A. Sub Additivity Property

If there exists a shortest path p from vertex u to v passing through w, then the sub-path p_1 of p from u to w is the shortest path from u to w. Similarly the sub-path p_2 of p from w to v is the shortest path from w to v. This is represented in Figure 2.

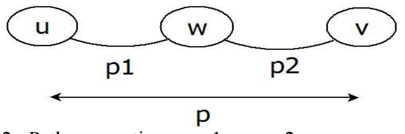

Fig. 2. Path connections: u p1 w, w p2 v, u p v

In our problem we have combined robot and hole movement in which the shortest path is determined by the number of robot moves plus the number of hole moves (from predecessor neighbour to successor neighbour).

We find that this property doesn't hold true for our case as is shown below through an example in Figure 3:

Jagrati Singh is with the Department of Computer Science, Banasthali Vidyapith, Allahabad-211004, India (e-mail: singh.jagriti5@gmail.com).

Anmol Darak and Sanket S. Das are with the Department of Mathematics, Indian Institute of Technology, Kharagpur, India (e-mail: anmolmathcs@gmail.com, sanketsagar.iitkgp@gmail.com).

Kalpesh Kapoor is with the Department of Mathematics, Indian Institute of Technology, Guwahati, India (e-mail: kalpesh@iitg.ernet.in).

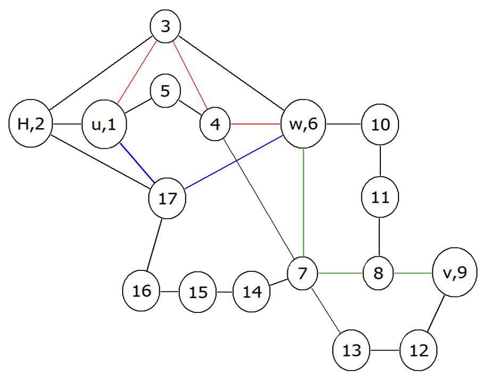

Fig. 3. Subadditivity property example

Let we have a shortest path p from vertex u to v passing through w using (l_1+d_1) number of moves from u to w (through red coloured path of vertices 1 − 3 − 4 − 6) and (l_2+d_2) number of moves from w to v (through green coloured path of vertices 6−7−8−9), where l_1 is the number of moves used by the robot in going from u to w (path p_1) and l_2 is the number of moves from w to v (path p_2), and d_1 is the number of moves used by the hole starting at u_1 (predecessor neighbour of u) and reaching w, and d_2 is the number of moves used by the hole starting at w_1 (predecessor neighbor of w) and reaching v. The cost incurred in the path is

$$P(C) = (l_1+d_1)+(l_2+d_2).$$

In order to disprove the above property, we propose a shortest path p_3 from u to w (blue coloured path of vertices 1 − 17 - 6) that uses (l_3+d_3) number of moves such that

$$(l_3+d_3) < (l_1+d_1)$$

and it uses (l_2+d_4) number of moves in going from w to v, predecessor neighbour on p_3) and reaching v.
Observe that d_4 is different where l_3 is the number of moves used by the robot in going from u to w (path p_3), d_3 is the number of moves used by the hole starting at u' and reaching w through p_3, d_4 is the number of moves used by the hole starting at w''.

Observe that d_4 is different from d_2 because after traversing path p_1 the hole lies on p_1 whereas in our proposed path p_3 after traversal the hole lies on p_3. Thus in the next transition from w to v, in the first case, the hole has to move from its current position on p_1 to its first desired position on p_2, and in the second case, the hole has to move from its current position on p_3 to its first desired position on p_2 thus proving d_4 different from d_2. But we show that the total cost in path,

$$P_3(C') = (l_3+d_3)+(l_2+d_4),$$

exceeds our cost of path $P(C)$. That is,

$$P_3(C') = (l_3+d_3)+(l_2+d_4) > (l_1+d_1)+(l_2+d_2).$$

Here in our example we have

$$l_1 = 3, l_3 = 2, d_1 = 4, d_3 = 3, l_2 = 3, d_4 = 10, d_2 = 7.$$

Thus
$$C = (l_1+d_1)+(l_2+d_2) = 17,$$
$$C' = (l_3+d_3)+(l_2+d_4) = 18.$$
$$C' > C$$

B. Triangle Inequality

Let $G(V,E)$ be an undirected, unweighted simple graph. Let us suppose that the source vertex of G is s, where there is atleast one path from s to every other vertex. Then for all edges $(u,v)\in E$, with $w(u,v)$ representing weight of edge from u to v, we have the following inequality:

$$\delta(s,v) \le \delta(s,u)+w(u,v).$$

Returning to our problem of robot's motion with a single hole, this property doesn't hold true for our case as shown below by an example in Figure 4, where we have d_1 as the distance of the shortest path from vertex u (vertex 2) to w (vertex 10), d_2 for shortest path from w to v (vertex 7) and d_3 for shortest path from u to v (not through w), then we have

$$d_3 > d_1 + d_2$$

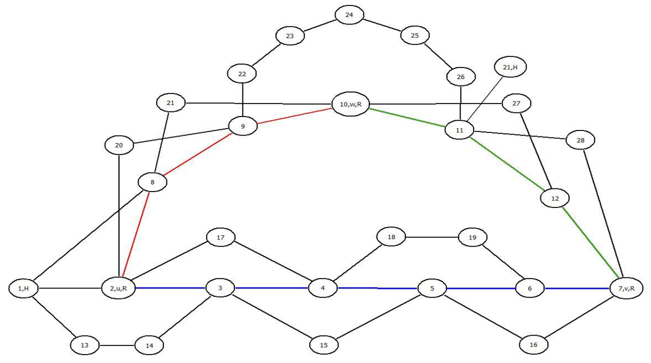

Fig. 4. Triangular property example

Here the red edges represent robot motion from u to w, the green ones from w to v, and the blue edges from u to v. Thus, the necessary condition for triangle inequality $d_3 \le d_1 + d_2$ is not valid in our example.

Proposition 1 (Robot motion path). *In order for the robot (R) to travel from starting point s to reach destination t in a graph G with the shortest number of moves possible, it always need not follow the actual shortest path from s to t in G.*

To show it we present an example in Figure 5 where the robot moves from s to t in G with the shortest possible number of moves but that path is not the actual shortest path from s to t.

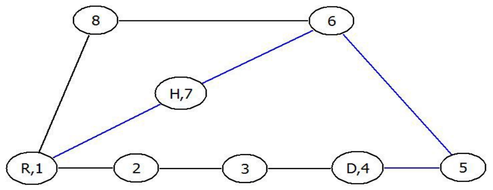

Fig. 5. Robot follows longer path to shorten its number of moves

III. Meta Search Algorithm

After looking at the properties in Section II, we now move on to the in-depth study of our algorithm. We have an undirected, unweighted graph $G(V,E)$ with $|V|=n$, $|E|=m,$ as the number of vertices and edges respectively. There is a mobile robot (R) at a starting vertex s that should reach a specified destination vertex t with only one hole (H) to aid its motion and the rest $(n-2)$ vertices containing obstacles. We can visualize this initial condition as a particular arrangement of R and H among all possible arrangements of R and H with obstacles. One can see that there are ${}^{n}P_2$ such arrangements possible for different positions of R and H.
Moving on, in order to simplify our problem, we form a Meta Graph (a type of configuration graph) $G^*=(V^*,E^*),$ with each vertex $v\in V^*$ as a particular arrangement of R and H in our graph G. We have two nodes V_i and V_j connected by an edge if both have the hole (H) in adjacent positions in G, i.e. both V_i and V_j differ in their arrangements by only a single hole movement to any of its adjacent vertex in G. We assign for each $v\in V^*$, $V.R$ as the position of R in V and $V.H$ as the position of H in V.
We have now reached a situation where we need to identify the node V^* with R at position t and H at a position adjacent to t, say t_{ad} ($t_{ad}\in adj[t],$ in G), that is, $V^*.R=t$ and $V^*.H=t_{ad}$, and that the node V^* is at shortest distance from initial vertex V_s (with R and H at position given in G) among all other such vertices. The number of vertices in our meta graph $G^* = {}^{n}P_2 = n(n-1).$

The algorithm starts with computation of a shortest path tree rooted at V_s that provides us with the shortest distance from V_s to each vertex $v\in G^*$ using Breadth first search algorithm. We have now got the shortest distance from V_s to the final vertex, thus the task left is to identify the vertex V^*. For each $v\in G^*$, we assign it, a parameter $d[r][h]$ which is the shortest distance from V_s to v in the shortest path tree.
We add a checking condition where we check the positions of R and H in each v, if they are found at t and t_{ad} respectively, then compare the distance of each such node with the optimal value which initially will be infinity, if distance is less than the optimal value, then update the optimal value and update that vertex in a new variable V^* each time for all such vertices. Next we find the path from source vertex V_s to V^* from the shortest path tree by the path finding algorithm in BFS.
Before looking at the algorithm we present ahead an example of Meta graph in Figure 6 and then we look at few properties related to the configuration graph.

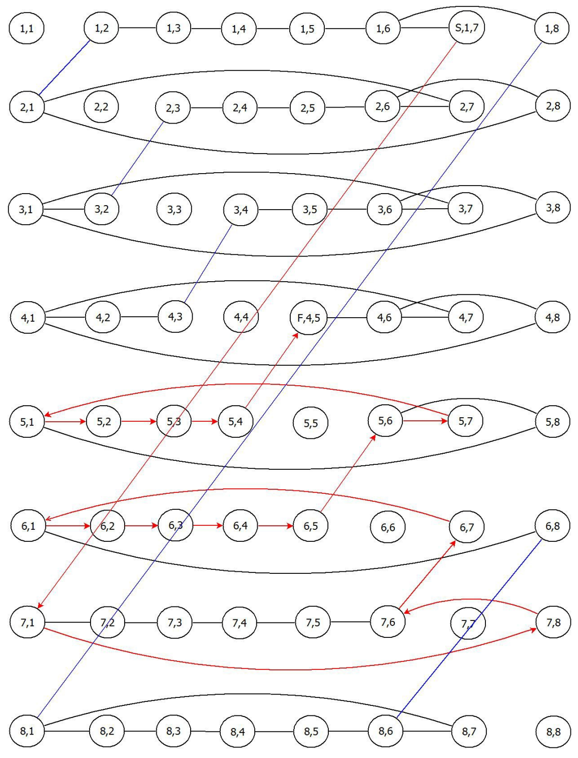

Fig. 6. Configuration graph for graph in Figure 5

A. Properties of Meta Graph

The meta graph introduced earlier concerns the various situations possible during the robot's traversal. It is interesting to look at some of its properties.

Theorem 1. *The number of edges in the configuration graph* G^* *of our original graph* G *is bounded by* $O(n^3)$.
Proof. Consider a vertex in the configuration graph denoted by (a,b), where a is the position of the robot (R) in G^* and b is the position of the hole (H) in G^* for that particular vertex of G^*. Assuming the number of vertices in the original graph to be n, we have $(n-1)(\_-b)$ combinations in the configuration graph.
Observe that (b,b) cannot be a vertex in G^* (as robot and hole cannot reside at same positions in a graph).
Now looking at a particular node (a,b), the degree of this node in G^* is the degree of the vertex b in G. Thus we have the number of edges in G^* as:

$$= (n-1)(\deg(v_1)+\ldots+deg(v_n))-X,$$
$$= (n-1)(2e)-X,$$
$$= (n-1)O(n^2)-X,$$
$$= O(n^3).$$

where X is number of such edges that get repeated and will be of order $O(n^2)$.

Recall that a vertex (a,b) is connected to (c,d) in the configuration graph if and only if

1) $a=c$ and b is adjacent to d in the original graph.
2) $a=d$ and $b=c$ and a is adjacent to b.

And this is the visualizing concept for the number of edges in the configuration graph being of order $O(n^3)$.

Theorem 2. *The length of the shortest path in* G *is bounded by* $O(n^2)$.

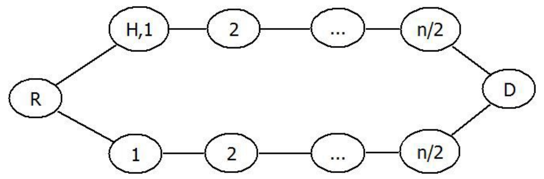

Fig. 7. Example of cycle depicting path length restriction

Proof. In order to show it, we consider an example in Figure 7 of a cycle of length n involving robot R (at s) and destination t, comprising of two chains of same length from s to t. We can see that the shortest distance between s and t is $n/2$. We can see that the robot takes either of these chains (depending on hole position) as its shortest path to t. It's easy to see that the number of edges equals the number of vertices in the cycle is n.

Let the position of the hole be just adjacent to the robot in the original setup, thus the first robot motion step requires one move, thus one edge from $n/2$ is covered, for the next edge, the hole has to move from predecessor position of R to successor position of R thereby traversing $(n-2)$ edges. Then we have the length of the shortest path is

$$((n/2-1)\times(n-2))+n/2=O(n^2).$$

Corollary1. *The shortest path length is inversely proportional to the number of edges in the configuration graph.*

We can see its applicability with few examples as shown in Figure 8, here we show the relation between the shortest path length and number of edges in general graph G and as edges in configuration graph is directly proportional to edges in general graph $(\mathrm{E}(\mathrm{G}^*)=\mathrm{E}(\mathrm{G})^2-\mathrm{E}(\mathrm{G}))$, these examples illustrate the concept.

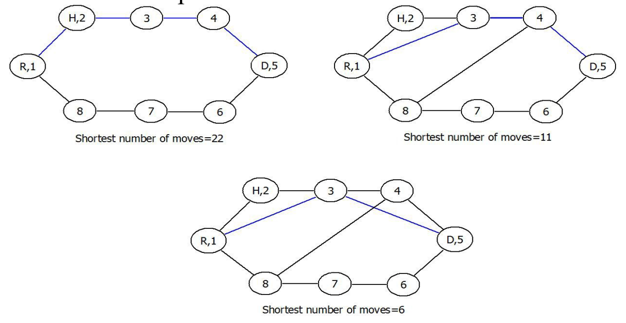
H,2
3
4
R,1
D,5
8
7
6
Shortest number of moves=22
Shortest number of moves=11
Shortest number of moves=6

Fig. 8. As number of edges increases, shortest path length decreases

In the next part we present the proposed algorithm.

A. Meta Graph Search Algorithm

Algorithm 1 : Meta Graph Search Algorithm

```
1:  G* = Make configuration graph (G, R, H), where
    R is the position of robot and H is the position of
    hole in the given graph G
2:  $V_s \leftarrow V[R][H]$
3:  Run BFS to store distance of each vertex from source
    vertex $V_s$ in $d[r][h]$ array
4:  $V^*$ = Search success node $G^*$
5:  If ($V^*$ is empty) then
6:  print " robot cannot reach to destination position"
7:  else
8:  print path from source $V_s$ to $V^*$ node by path finding
    algorithm in BFS
Make configuration graph (G, R, H)
1:  From graph $G=(V^*,E^*)$ with vertex $V^* \in G^*$ for each
    possible arrangement of $R$ and $H$ among $n$
    vertices in G, thus $|V^*| = {}^nP_2 = n(n-1)$. We store all
    such in two dimensional array (size $n(n-1)$) with
    $v[r][h]$ representing $V.R=r$ and $V.H=h$
2:  Initially edge set $E^*$ is empty set
3:  for $h=1$ to $n$ do
4:      for $r=1$ to $n$ do
5:          for all $h_{ad} \in adj[h]$ do
6:              if $(r=h_{ad})$ then
7:                  Form edge $e_1$ between $v[r][h]$ and
                    $v[h][r]$
8:              else
9:                  if $(h \neq r)$ and $(h_{ad} \neq r)$ then
10:                     Form edge $e_1$ between $v[r][h]$ and
                        $v[r][h_{ad}]$
11:                 end if
12:             end if
13:             if $e_1$ is not present in set $E^*$ then
14:                 $E^* \leftarrow E^* \cup e_1$
15:             end if
16:         end for
17:     end for
18: end for
Search success node ($G^*$)
19: optimal = $\infty$
20: for $h=1$ to $n$ do
21:     for $r=1$ to $n$ do
22:         for each vertex $t_{ad} \in adj[t]$ do
23:             if $(h \neq r)$ then
24:                 if $v[r][h] == v[t][t_{ad}]$ && $d[r][h] <$
                    optimal then
25:                     optimal $\leftarrow d[r][h]$
26:                     $V^* \leftarrow v[r][h]$
27:                 end if
28:             end if
29:         end for
```

```
30:      end for
31:  end for
```

B. Analysis and Time Complexity

- If the Configuration Graph G^* is disconnected and the shortest path tree rooted at V_s doesn't contain any node with $v[r][h] == v[t][t_{ad}]$, it implies that it is not possible to reach such a node from V_s, making the problem infeasible.
- The basic idea of this algorithm resides on the fact that this meta graph G^* consists of all the possibilities for robot and hole movements and without much thought one can see that this graph in reaching from V_s to V_f in shortest path indeed is equivalent to robot moving from s to t in G in shortest path as the situations arising in the shortest path in G is being enumerated in different arrangements of *R* and H in G^*.
- Our algorithm for robot motion planning takes the following as input, the original graph G, the initial position of robot (*r*) and hole (*h*) in the graph G and destination position (*t*) where robot has to finally reach.

  Step 1. The time complexity of the Make Configuration Graph subroutine is O(n^3) because time complexity to make a graph is of order $(V^* + E^*)$.

  Step 2. In this step, we compute the shortest path tree rooted at s, thus providing us with the shortest distance $d[r][h]$ for each vertex $v \in G^*$ from node V_s. We use BFS to find the shortest path whose time complexity is $O(V^* + E^*)$ i.e $O(n^3)$ because there is $O(n^3)$ number of edges in the configuration graph.

  Step 3. In this subroutine we search the success (destination) node where robot has to reach in optimal number of moves. The time complexity of this step is $O(n^3)$ because we run the loop for each combination of hole and robot and further we check for the number of adjacent positions of the hole whose complexity will be of $O(n)$ which all together gives $O(n^3)$ complexity.

  Step 4. The path is printed from source to V^* node which we have found to be optimal in our problem in $O(n^2)$ time because maximum number of vertices in any path can be n^2.
- Hence the overall time complexity of our algorithm is $O(n^3)$.

C. Algorithm Demonstration Through Examples

Example 1.

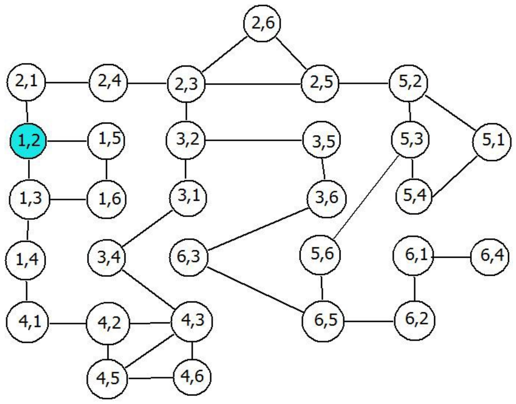

Fig.9. Original graph *G* where initial position of the robot is vertex 1, hole position is 2 and destination position is vertex 5.

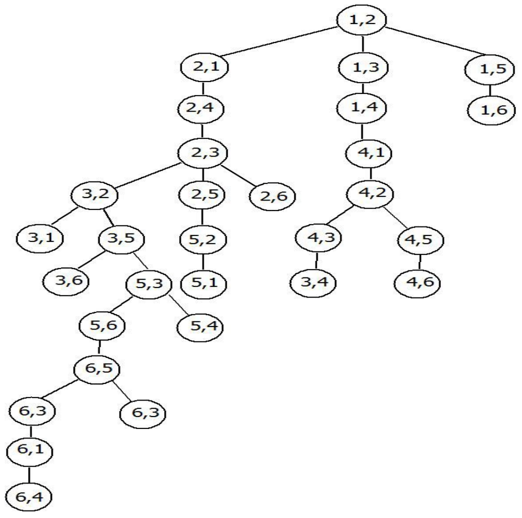

Fig. 10. This is the configuration graph *G** that is the 1st step of our algorithm. Here v[1][2] vertex is the source vertex where 1 is the position of robot and 2 is the position of hole in original graph *G*.

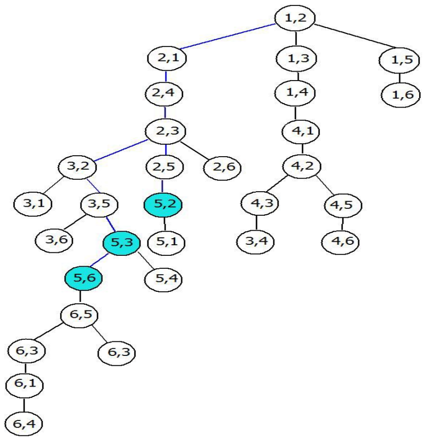

Fig. 11. Shortest Path Tree of Configuration Graph G* i.e 2nd step of our algorithm.

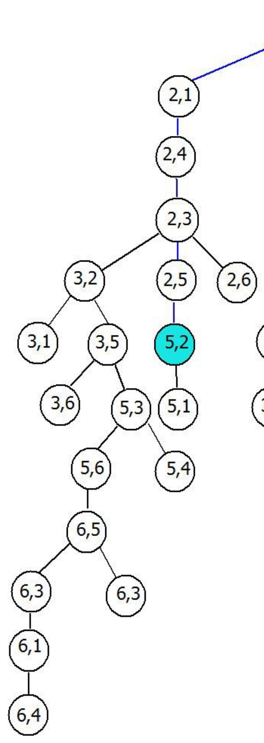

Fig. 12. This is the final step of our algorithm. Coloured node represents success node and coloured line showing the shortest path that is followed by the robot to reach the destination.

Example 2.

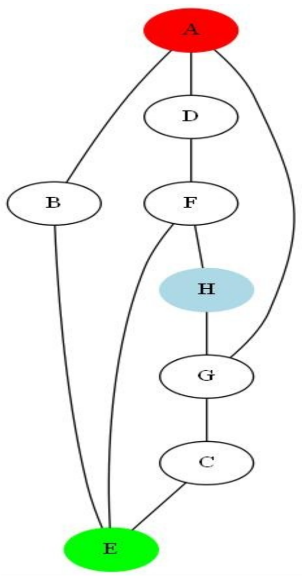

Fig. 13. This is the Original Graph. Red colored node represents source node and green colored node showing the hole and blue colored node showing the destination node where robot has to finally reach.

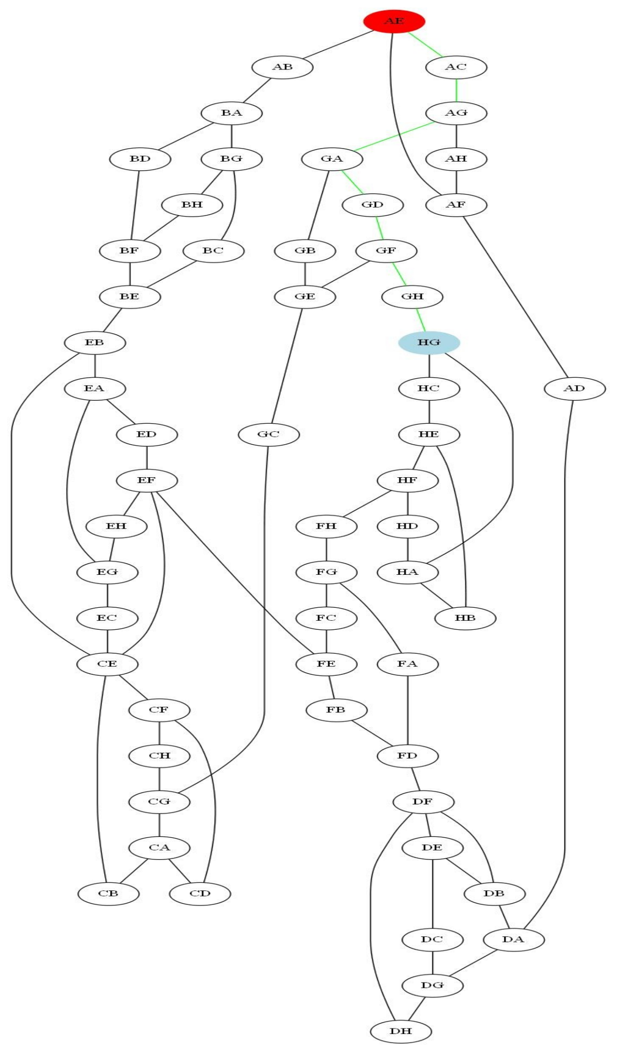

Fig. 14. This is the Configuration Graph of our example. First position of node is Robot position in original graph and second position is hole position in original graph. Blue colored node represents success node and green colored line showing the shortest path that is followed by the robot from red colored node that is source node of configuration graph to reach the destination.

IV. Conclusion and Further Work

The robot motion planning problem discussed here makes use of concepts of shortest path discussed in [1]. Here we study the GMP1R problem with single hole making use of Breadth first search shortest path algorithm and present an O(n^3) algorithm in two different ways. Further study into the replacement path concept involving all pairs can create inroads into a better and faster approach.

V. References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms.* Prentice Hall of India Private Ltd, 2001.

[2] Christos H. Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki, "Motion planning on a graph", *Foundations of Computer*, 1994, pp 511-520.