Generating an Orientation to Minimize Shared Links

Krushna Gopal Chukkayapally
40165076
Master of Applied Computer Science
Concordia University
Montreal, Canada

Vaishnavi Yendapally
40162741
Master of Applied Computer Science
Concordia University
Montreal, Canada

Sindu Chandra Kanth
40168381
Master of Applied Computer Science
Concordia University
Montreal, Canada

Aarthi Minde
40161231
Master of Applied Computer Science
Concordia University
Montreal, Canada

Nithin Reddy Indurthi
40181743
Master of Applied Computer Science
Concordia University
Montreal, Canada

***Abstract*—In order to follow the social distancing rules in hospitals, which include both Covid and Non-Covid patients, it is important to minimize the usage of same lanes for the movement of patients to reduce congestion and increase throughput. Motivated to solve the above problem, we worked on an algorithm which takes input as a graph G and two sets of node pairs named SD_1 and SD_2 to minimize the sharing of links. Since the number of node pairs in SD_2 are very limited, we performed a modified Breadth First Search(BFS) operation to reduce the usage of links for SD_2 orientation as much as possible and increase the cost of each edge. For SD_1 orientation, we find a path with least cost by using Depth First Search(DFS) and MinimumCostPath methods, which indeed will be a path consisting of minimal shared edges. If any edge has to be used by both Covid and Non-Covid paths in both directions, we convert that into an alternating edge.**

I. Introduction

In this paper, we discuss the problem of finding the accurate path to reach all the rooms of a hospital which has both Covid and Non-Covid patients. In the pandemic situation, it is important for the hospital authorities to focus on creating a risk free environment and also to improve the movement of patients. To solve this problem, we must assign a path for every Covid and Non-Covid patient separately, so that there is a very minimal contact between them. Since there exists multiple lanes between rooms in hospital corridors, we assume each lane as a link and links must be colored and given orientation to minimize the sharing of those links. Sharing of the lanes will decrease the number of patients going through that lane, So this paper is a work on algorithm to reduce the number of shared links and to increase throughput.

II. Problem Statement

Given two sets of node pairs named SD_1 and SD_2 which consists of pair of vertices indicating the Covid and Non-Covid sets and an undirected graph G = (V,E), in which the vertices represents the rooms in a hospital and edges are links between them. With the given inputs, we are intended to find the path from source node to destination node in each of the node pairs in SD_1 and SD_2 sets.

A Shared link is an edge/link which allows either the Covid or Non-Covid patients to share a link at a given time. An Alternating link is an edge/link which can be used in both directions by any patient but in an alternate manner at a given point of time. This means, if the path is being used by any Non-Covid patient at a certain time, then any Covid patient who has to use that link must wait in the Waiting room. These waiting rooms are at the end of the alternating links and the patients are supposed to wait until the path becomes free. Every node pair in SD_1 and SD_2 sets have path 'p' in between them. The objective is to design an algorithm to minimize both shared and alternating edges with respect to link 'l' in path 'p', so that number of links which are alternating or shared are minimum in between.

III. Literature review

- The paper [1] provided an intuition for solving the problem of giving orientation for graphs having certain connectivity constraints. We have got a clear understanding about essential edges and how to find them in a graph by studying this paper.
- The idea on how to find minimum shared edges in a directed graph has been obtained from the paper [2].
- Finding disjoint paths between source and destination can be done by giving a weight of 1 to every link and by running the Ford Fulkerson algorithm. Finding minimally disjoint paths will help us solve the problem of minimum shared links. Different approaches of solutions to find minimal disjoint paths has been studied from [3].
- From this article [4], we have got idea to find all possible paths from source to destination using Depth First Search(DFS).

- The article [5] helped us to determine a minimum cost path between source and destination among all the obtained paths.
- The article [6] has been used to keep track of the path it followed to search an element using BFS, with the help of a variable “prev”.

IV. Orientation of the Graph

Approach : In order to find the paths between every node pair present in the SD_1 and SD_2 by minimizing the shared edges, it is better to find the orientation of red edges first. This is because there are minimal number of nodes present in SD_2(Covid) set and we made sure to use as less number of red edges as possible.

Red Orientation : For each node pair present in SD_2 set, we perform Breadth First Search(BFS) operation to find the path from source to destination of that particular node pair. In this process, we increment the cost of each edge by 10. Once the path is found for all the node pairs in SD_2 set, we will get the orientation of red edges with a minimal usage of edges.

Green Orientation : For each node pair present in SD_1 set, we perform Depth First Search(DFS) operation to find all possible paths from source to destination of that particular node pair. In this process, we get multiple paths with different costs for each node pair. With the help of a FindMinCostPath function , we find the path with least cost among all the previously obtained paths. The reason for selecting a path with least cost is to minimize the shared links. Since the edges with high cost indicates that they have already been used(shared) multiple times and cannot be used more. This improves the minimization of the shared edges and maximization of throughput.

A. Minimal Shared Edges

Algorithm 1 Minimal Shared Edges

procedure $MinimalSharedEdges$(G(V,E), SD_1, SD_2)
    $redDirected \leftarrow null$
    $greenDirected \leftarrow null$
    $blueEdges \leftarrow null$
    $RedOrientation(G, SD_2, redDirected)$
    $GreenOrientation(G, SD_1, redDirected, greenDirected, blueEdges)$

Input: The inputs for MinimalSharedEdges method are an undirected graph G=(V,E), where V represents the vertices and E represents the edges and two sets of nodes pairs named SD_1(Non-Covid) and SD_2(Covid).

Parameters: The list ”redDirected” will store the red directed edges in the graph for SD_2 node pairs. The list ”greenDirected” will store the green directed edges in the graph for SD_1 node pairs. The list ”blueEdges” will store essential edges with its start node as waiting room.

Approach : We call ”RedOrientation” algorithm to arrange red path, then we call ”GreenOrientation” algorithm to arrange green path. The red path is for Covid set and green path is for Non-Covid set.

While running Red, we try to use as minimum edges as possible, so that while running green, it will be easy to avoid such edges which are already used in red path (shared edges).

B. Red Orientation using BFS.

Algorithm 2 ModifiedBFS

procedure $ModifiedBFS(G(V,E), source, destination)$
    for all $v \in$V[G] **do**
        $visited[v] \leftarrow$false
        $v.prev \leftarrow$null
    end for
    $Q :=$Empty
    ENQUEUE(Q, source)
    while not Empty(Q) **do**
        $u \leftarrow Dequeue()$
        if not visited[u] **then**
            $visited[u] := true$
            for all $w \in$Adj[u] **do**
                if not visited[w] **then**
                    ENQUEUE(Q,W)
                    $w.prev \leftarrow u$
                    if w = destination **then**
                        $visited[w] := true$
                        Q.Empty()
                        break
                    end if
                end if
            end for
        end if
    end while

Algorithm 3 Red Orientation

procedure $RedOrientation$(G(V,E), SD_2, $redDirected$)
    for (source, destination) in SD_2 **do**
        $ModifiedBFS(G, source, destination)$
        $tempNode \leftarrow destination$
        while $tempNode \neq$source **do**
            if (tempnode.prev,tempnode) not in RedDirected **then**
                $redDirected.add((tempNode.prev, tempNode))$
                $cost(tempNode.prev, tempNode) \leftarrow cost(tempNode.prev, tempNode) + 10$
            end if
        end while
        Delete (source,destination) from SD_2
    end for

1) Modified BFS: Input: The algorithm takes input as Graph, source, destination.

Parameters: "prev" variable which keep tracks of BFS.

Approach: The only difference between a normal Breadth First Search(BFS) and "ModifiedBFS" algorithm is that, "ModifiedBFS" stops once the destination is found. Also, it is going to keep track of a variable of data type vertex named "prev" which stores the previous node that it used to reach the present node using BFS. We initialize prev node for every vertex to null at the start of BFS. $w.prev \leftarrow u$ (Updating prev variable, to keep track of how BFS is done to reach that node.)

Eg.1—2—3—4--5--6--7--8

Explanation for "prev" variable in "ModifiedBFS" : After finding the node 8 in above example, using BFS, it will have a variable that stores 7 as previous, and 7 stores 6 as previous, so this allows us to give directions to the edges making them as 6 $\rightarrow$ 7$\rightarrow$ 8.

The complexity of this algorithm is same as that of BFS but we are keeping track of prev variable. Hence the time complexity is $O(V + E)$.

2) RedOrientation Algorithm: Input: The algorithm "RedOrientation" takes graph G, node pair set SD_2 and set of red directed edges "redDirected" as input.

Parameters: redDirected.add((source,destination)) will add a directed red edge from source, destination.

function "cost(s,d)": The cost/weight of edge between s,d.

Approach: We are calling ModifiedBFS in line 3 for every source and destination pair in SD_2. From the lines 4 to 12, using prev node, we are backtracking the path of BFS to reach destination and adding the directed edges into redDirected and incrementing its edge cost by 10.

Complexity: The for loop in line 2 runs for $O(V^2)$ times in worst case scenario. The complexity of modifiedBFS which was called in line 3 is $O(V+E)$. The while loop in line 5 runs for O(E) times which indicates the total number of edges in the worst case. Hence the total complexity is $O(V^2 * (V + 2E))$, that is $O(E * V^2)$.

C. Green Orientation Using Minimum Cost

1) Finding the Minimum Cost Path: Input: Graph, source, destination, greenDirected list.

Parameters: cost(e): The cost/weight of particular edge,

$|E|$: Number of edges,

mincost: minimum cost of all the paths.

Approach: FindMinCostPath function will find all the paths from source to destination and it will compute the cost of each path, as already used links will have cost of 10, shared edges will have cost of 20, and essential edges will have cost of 1.

If the edge already has green color and in the same direction as expected, decrement the cost by 1 so as to encourage to take that path instead of taking an unused path. Out of all paths, find the best path with minimum cost, that will be the path with minimum shared edges and minimum essential edges. Complexity: The complexity for line 2 will be $O(V^V)$ from [4] as we are running DFS for V times from each vertex. The complexity for lines 5 to 18 will be $O(|N| * E)$, where N is

Algorithm 4 Minimum Cost Path

1: **procedure** $FindMinCostPath$(G, $source$, $destination$, $greenDirected$)
2: Find all possible paths from source to destination using Depth First Search (DFS)
3: $mincost \leftarrow |E| * 20$
4: $minpath \leftarrow$null
5: **for** $every\ path$ **do**
6: $cost \leftarrow$0
7: **for** $edge \in path$ **do**
8: **if** $edge \notin greenDirected$ **then**
9: $cost \leftarrow cost + cost(edge)$
10: **else**
11: $cost \leftarrow cost - 1$
12: **end if**
13: **end for**
14: **if** $cost < mincost$ **then**
15: $mincost \leftarrow$cost
16: $minpath \leftarrow$path
17: **end if**
18: **end for**
19: $return\ minpath$

Algorithm 5 Green Orientation

1: **procedure** $GreenOrientation(G(V, E), SD_1,$ redDirected, greenDirected, blueEdges)
2: $minSharedPath \leftarrow$null
3: **for** (source, destination) in SD_1 **do**
4: $minsharedPath \leftarrow$FindMinCostPath(G,source, destination, greenDirected)
5: **for all** $e \in$minsharedPath **do**
6: **if** cost(e) == 0 **then**
7: $greenDirected.add(edge(e.startNode, e.endNode))$
8: $cost(e) \leftarrow cost(e) + 10$
9: **else if** cost(e) == 10 and !greenDirected.contains(e.startNode,e.endNode) **then**
10: $greenDirected.add(e.startNode, e.endNode)$
11: $cost(e) \leftarrow cost(e) + 10$
12: **else if** cost(e) == 20 **then**
13: $blueEdges.add(e.startNode, e.endNode)$
14: $type(e.startNode) \leftarrow$Waiting area
15: $redDirected.remove(e)$
16: $greenDirected.remove(e)$
17: $cost(e) \leftarrow$1
18: **end if**
19: **end for**
20: $Delete(source, destination)$ in SD_1
21: **end for**

the number of paths found in DFS. Hence the total complexity is $O((V^V) + (|N| * E))$, that is $O(V^V)$.

2) GreenOrientation Algorithm: Input: The algorithm GreenOrientation takes graph G, node pair set SD_1 and List of "redDirected", "greenDirected", "blueEdges" as input.

Initialization: Initialize mincostPath to null. parameters: greenDirected.add(source,destination) will add a directed green edge from source, destination. Approach: In lines 3 to 20, we take each node pair present in the SD_1 set and first we will find the minSharedPath using FindMinCostPath, then for each edge in path, we do the following:

- If cost of that edge is 0, it means that, the edge has not been used by any node pair before. And by choosing it, the shared edges can be minimized. We will add this directed edge into the "greenDirected" list and increment its cost by 10.
- Else, If the cost of that edge is 10, it means that, the edge has been used once for finding a path of some node pair.
  - If that edge has been used by some green path and in same direction, meaning that, the required orientation has already been given to that edge.
  - Else, If that edge has been used by some other path, then add that edge into the "greenDirected" list and increment its cost by 10. This will be considered as a shared edge from now on.
- Else, If the cost of that edge is 20 and that edge is not present in the list of green directed edges, then it will be considered as alternating edge from now. Add that edge to the Blue edges list and remove it from Green, Red directed edges list. Update the start node of that edge as "waiting room". The cost of that edge will be updated to 1. It is updated to 1 so as to encourage the use of same alternating edge instead of creating some other shared or alternating edges.

Finally, Delete the (source, destination) pair from SD_1 as path between them is already set.

Complexity: The for loop in line 3 will run for $O(V^2)$ times. The function FindMinCostPath will run for $O(V^V)$ times which is in line 4. The for loop in line 5 will run for E times in the worst case scenario. This makes the complexity of line 4 to 18 be $O(E + V^V)$. Hence the total complexity of GreenOrientation algorithm becomes $O(V^2 * (E + V^V))$, that is $O(V^V)$.

D. Complexity Analysis

Complexity of our algorithm i.e., MinimimalSharedEdges is the $O(RedOrientation + GreenOrientation)$, that is $O((E * V^2) + V^V)$. Hence the total complexity becomes $O(V^V)$. As the complexity of our algorithm is exponential, it might be an NP-Hard problem, which is similar to the one mentioned in [2]

E. Diagramatic Explanation

Let us consider the following graph differentiated by Covid ward and Non Covid wards.

SD_1 - {(A,L), (L,A), (H,G), (G,A), (H,I), (I,H), (I,J), (J,A),(A,D), (D,A), (A,K), (K,A) }

SD_2 - {(C,F), (A,C),(C,L), (F,A), (L,A),(A,L)}

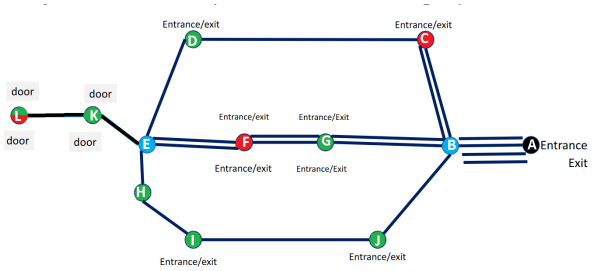

Fig. 1. Input graph with red and green nodes.

In the graph example given in Fig. 1., nodes C,F,L are considered as Covid nodes where L is both in covid and Non covid set. The RedOrientation algorithm will run for all the node pairs present in SD_2. The first node pair is (C,F). First, Breadth First Search (BFS) operation will be run on (C,F). A path will be found between them, which is C→D→E→F and its red orientation will be given. In this process, the cost of every visited edge will be incremented by 10. Similarly, path is found for node pairs, (A,C), (C,L), (F,A). Inorder to find the path between node pair (L,A), After doing BFS we will find path L→K→E→F→G→B→A, we give red orientation from B→A, G→B, F→G and we already found path from L→F. Path between (A,L) is already found. After giving the orientation for all nodes in SD_2, the graph will be represented as in Fig. 2.

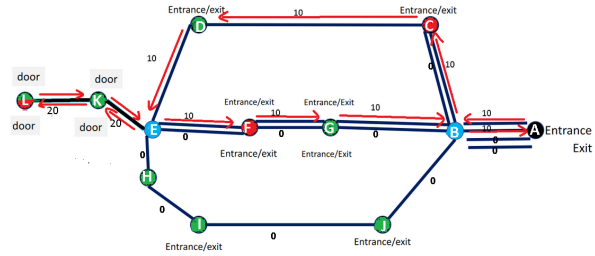

Fig. 2. Output at the end of RedOrientation algorithm.

Now let us continue green orientation for the SD_1 set - {(A,L), (L,A), (H,G), (G,A), (H,I), (I,H), (I,J), (J,A),(A,D), (D,A), (A,K), (K,A) }

After the red orientation, we are left with SD_1 node pairs which belongs to Non-Covid nodes. We consider each node pair and find the path in between them which will produce least cost. In our example, first node pair is (A,L). The possible paths from A to L are (i) A→B→J→I→H→E→K→L with cost = 40, (ii)A→ B→C→D→E→K→L with cost = 70, (iii) A→B→G→F→E→K→L with cost = 40. In the path A→B→G→F→E→K→L, there are two possible paths between B to E as there are two edges between each of the

nodes. The cost of these two possible paths will be 40 and 70 (70 being the one which is already used in red orientation). The FindMinCostPath algorithm will consider the path with least cost, i.e., (i)A→B→J→I→H→E→K→L with cost 40.

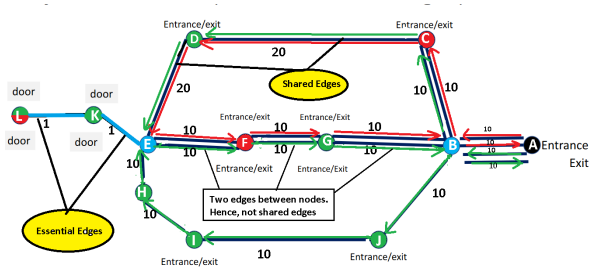

Fig. 3. Output at the end of GreenOrientation algorithm.

After finding the path from A to L, the set SD_1 will be {(L,A), (H,G), (G,A), (H,I), (I,H), (I,J), (J,A), (A,D), (D,A), (A,K), (K,A)}. In the similar way, using the FindMinCostPath function, path for all the SD_1 node pairs will be found. The result of the GreenOrientation algorithm is shown in the figure Fig. 3 where L-K , K-E ended up as Essential edges and C-A, D-E are Shared edges.

Example-2: We have given another example where the node pair sets are SD_1 = {(A,I), (I,E), (A,J), (J,A), (E,F), (F,G), (E,A), (A,E), (A,G), (G,A), (F,A), (A,F)}

SD_2 = {(A,K,), (K,A), (C,D), (A,C), (D,A)}

The input graph has C, D, K as Covid nodes and there are two edges between (B,G), (G,F), (F,E) and (B,J).

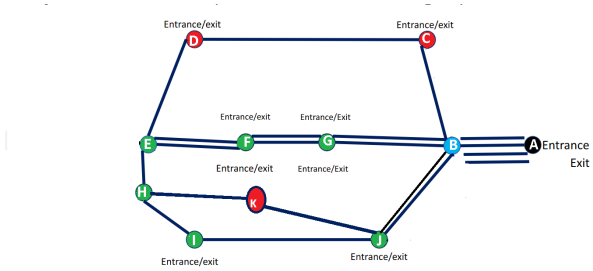

Fig. 4. Input graph for Example-2

The output of the Example-2 is shown in Fig. 5., where we have 1 Shared edge between H,E and no Essential edges.

F. Deductions drawn from the Algorithm

As per our algorithm, it is necessary to distinguish between two types of sharing, which are sharing the link with same color or different color. We have to discourage both types of sharing in order to increase throughput. But, it is less important to discourage sharing of path in same direction by same color, but more important to discourage the sharing of path with different directions or different colors.

This can be explained from the figure Fig. 6. The diagram picture-2 is not discouraging the usage of existing green

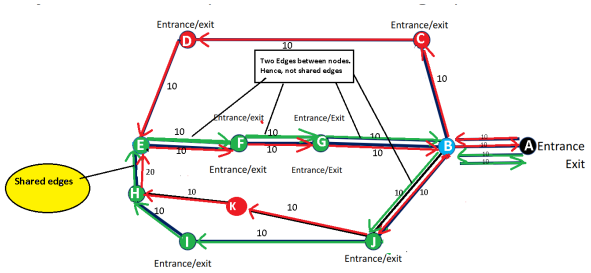

Fig. 5. Output after performing RedOrientation and GreenOrientation.

path of same direction from node 'a' to 'c' and picture-1 is discouraging the same. In picture-1, we ended up having 2 shared links of different colors(red and green) and zero shared links of different colors in picture-2. So, in our view, it is better not to discourage the sharing of paths of same color and same direction. We feel that minimization of shared links will help us to maximize the throughput as number of people travelling in a lane will reduce and they can reach destinations faster.

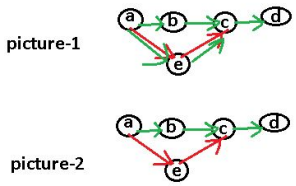

Fig. 6. Explanation of shared paths

CONCLUSION

In this paper, we have discussed an algorithm for giving the orientation and coloring of the edges in an undirected graph. We achieved this with a little modification to the Breadth First Search operation and finding the minimum cost path among all the paths obtained from Depth First Search operation. The main constraints of the algorithm were to minimize the number of shared edges while finding a path from every node to every other node and also to maximize the throughput of each edge. This approach will help the hospitals to organize their corridors to allow the Covid and Non-Covid patients.

REFERENCES

[1] Arkin, E.M., Hassin, R. (2002). *A note on orientations of mixed graphs. Discret. Appl. Math., 116, 271-278.*

[2] M.T. Omran, J. Sack, and H. Zarrabi-Zadeh. Finding paths with minimum shared edges. J. Comb. Optim., 26(4):709–722, Nov. 2013.

[3] Iqbal, Farabi. (2015). Disjoint paths in networks. Wiley Encyclopedia of Electrical and Electronics Engineering.

[4] https://www.geeksforgeeks.org/find-paths-given-source-destination/

[5] https://stackoverflow.com/questions/56660553/finding-minimum-cost-in-graph-using-dfs

[6] https://stackoverflow.com/questions/8922060/how-to-trace-the-path-in-a-breadth-first-search