IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

Obtaining a Feasible Path with Maximum Flow Rate in a Network

Shreekant Jere

Presidency University, Bangalore

Abstract: This research work proposes an algorithm to find a path which has the maximum allowed flow rate for data, between source and destination in a network. Unlike max-flow and min-cut theorem, algorithm is selecting single path for data transmission. To find a path in a network there are multiple techniques. Prim's technique is used recursively in our proposed algorithm to find different paths between source and destination. The maximum allowed flow rate for each of those paths is calculated and finally we take the maximum of those calculated flow rates.

Keywords: Maximum flow rate; max-flow and min-cut theorem; Prim's algorithm.

I. INTRODUCTION

In many transport networks, knowledge about finding a path with maximum flow rate, from source node to destination node is essential. Such information can be acquired through, study of weighted connected graphs in which vertices represent nodes and the edges represent the links of the network. In such graphs, the weight of an edge represents the capacity of the link; namely, the maximum amount of flow possible per unit of time. It will be assumed that there is no accumulation of data at any node along the path and that the node itself can handle as much flow as allowed through the links. It will further be assumed that the links are lossless. Our proposed algorithm is a maximum flow rate algorithm which mainly finds the path between source and destination such that selected path should have maximum allowed flow rate. Here, maximum allowed flow rate refers to a path with maximum allowable data rate that can flow between selected source and the destination. To find a path between source and destination we have used Prim's algorithm [1]. Prim's algorithm finds a minimum spanning tree in a given network [2]. Here, Prim's algorithm has been utilized to find different paths between source and destination. Our proposed algorithm finds maximum flow for each path selected from Prim's algorithm. Finally, we find maximum of maximum flow rate that can flow in a network from the set of calculated maximum flow rates.

II. PRIM'S ALGORITHM

Prim's algorithm constructs minimum spanning tree through a sequence of expanding subtrees. The initial subtree in such a sequence consists of a single vertex selected arbitrarily from the set *V* of the graph's vertices. On each iteration, we expand the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. The algorithm stops after all the graph's vertices have been included in the tree being constructed. Since the algorithm expands a tree by exactly one vertex on each of its iterations, the total number of such iterations is n-1, where n is the number of vertices in the graph. The tree generated by the algorithm is obtained as the set of edges used for the tree expansions.

ALGORITHM: *Prim* (*G*)
//Input: A weighted connected graph G = (V, E)
//Output: E_T, the set of edges composing a minimum spanning tree of G
1: $V_T \leftarrow \{v_0\}$ //set of tree vertices initialized with any vertex
2: $E_T \leftarrow \emptyset$
3: for i ← 1 to |V| – 1 do
4: find a minimum- weight edge $e^*=(v^*, u^*)$ among all
the edges (v, u) such that v is in V_T and u is in $V- V_T$
5: $V_T \leftarrow V_T \cup \{u^*\}$
6: $E_T \leftarrow E_T \cup \{e^*\}$
7: return E_T

III. MAXFLOW ALGORITHM

Maximum flow algorithm finds out the path between source and destination with maximum allowable flow rate. Unlike max-flow and min-cut theorem, we are selecting single path for data transmission [3-6]. In the initial network source node *s* and destination node *d* are selected from the set of nodes *V*. The weights on the link represents maximum allowed flow rate for a particular time period. Using Prim's algorithm, we find path between *s* and *d*. On each iteration after finding maximum flow the link with minimum flow rate is deleted. This method is done recursively until there is no path between *s* and *d*. Finally maximum of maximum allowed flow rate is calculated.

MaxFlowRate = 0 //global variable, initialized to 0

ALGORITHM: Maxflow (*G*, *s*, *d*)

```
1: visited[]={0}
2: Copy graph G to graph Rate
3: visited[s]=1
4: ne=1
5: while ne < |V| and visited[d]!=1
6:      for i←1 to |V| do
7:          min←∞
8:          for j←1 to |V| do
9:              if Rate(i, j) < min
10:                 if visited[i]!=0
11:                     min←Rate(i, j)
12:                     u←i
13:                     v←j
14:     if visited[u]==0 or visited[v]==0
15:             ne++
16:             edge_minrate = edge_minrate + min

17:             visited[v] ←1
18:         Rate(u, v) ← Rate(v, u) ← ∞
19:         flag ← 0
20:         if there is a link from at least one visited node to the
            unvisited node
21:             flag ← 1
22:         if flag == 0
23:             go to step 25
24: end while
25: if flag == 0 and visited[d] == 0
26:     Terminate //There is no further path to the destination
27: else
28:     MinFlowRateEdge ← ∞
29:     find the edge with minimum rate
30:     MinimumFlowRateEdge ← minimum rate of the edge
31:     if MinimumFlowRateEdge > MaxFlowRate
32:         if MinimumFlowRateEdge != ∞
33:             MaxFlowRate = MinimumFlowRateEdge
34:     assign the edge in G with minimum flow rate in a path
        from s to d to ∞
35:     Maxflow(G, s, d)
```

The *maxflowrate* is the global variable which is initially set to zero. Line 1 set all the elements of array *visited[]* to zero. Line 2 copy the graph(network) *G* to graph *Rate*. For further calculations we use the graph *Rate* and the operations carried out on the graph *Rate* doesn't reflect on the original graph *G*. Line 3 sets the *visited[s]* to 1. Line 4 sets variable *ne* to 1, which is the initialization condition for while loop. The while loop of lines 5-24 finds the path between source and destination. If there is no path exists from source to destination, variable *flag* is set to zero and the control goes to step 25. Line 25 checks whether the *flag* is set to zero. Also, it checks whether *d* is not visited. If both the conditions are true then the Maxflow function terminates. Else, it continues with the step 27. Line 28 sets the *MaxFlowRate* to ∞. Line 29 finds the edge in the path with minimum rate. Line 30 sets the *MinimumFlowRateEdge* to the calculated minimum rate of the edge. Line 31 checks whether *MinimumFlowRateEdge* is greater than *MaxFlowRate* and Line 32 checks whether *MinimumFlowRateEdge* is not equal to ∞. If both are true then *MaxFlowRate* is set to *MinimumFlowRateEdge*. Line 34 assigns the edge in *G* with minimum flow rate in a path from *s* to *d*, to ∞. Line 35 calls the function Maxflow(*G*, *s*, *d*) recursively.

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified

Vol. 7, Issue 6, June 2018

IV. DETAILED EXAMPLE

Consider the network graph as shown in fig.1:

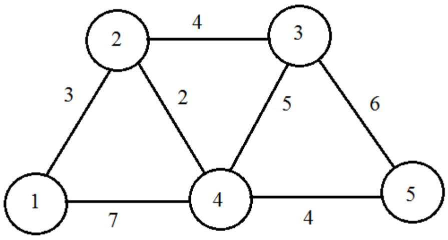

Fig. 1. Initial graph before 1st iteration

The abutment matrix of this example is like this:

$$\begin{bmatrix} 0 & 3 & \infty & 7 & \infty \\ 3 & 0 & 4 & 2 & \infty \\ \infty & 4 & 0 & 5 & 6 \\ 7 & 2 & 5 & 0 & 4 \\ \infty & \infty & 6 & 4 & 0 \end{bmatrix}$$

Iteration 1:

Step 1: Path from source 1 to destination 5 is 1 → 2 → 4 → 5. Here maximum allowed flow rate of individual links 1 → 2, 2 → 4 and 4 → 5 are 3, 2 and 4 respectively.

From the above flow rates of the individual links 3, 2, and 4, minimum allowed flow rate is 2. Therefore here, the maximum allowed flow rate from source node 1 to destination node 5 is **2**.

Step 2: Now delete the link 2 → 4 which has the minimum allowed flow rate among all the individual links in the considered path 1 → 2 → 4 → 5.

The resultant graph after iteration 1 is shown in fig.2:

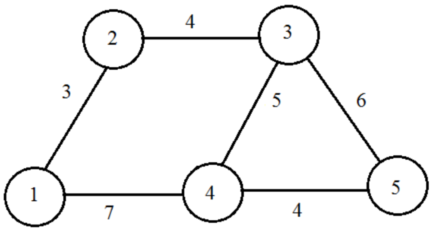

Fig. 2. Residual network after 1st iteration

The corresponding matrix for the above graph is:

$$\begin{bmatrix} 0 & 3 & \infty & 7 & \infty \\ 3 & 0 & 4 & \infty & \infty \\ \infty & 4 & 0 & 5 & 6 \\ 7 & \infty & 5 & 0 & 4 \\ \infty & \infty & 6 & 4 & 0 \end{bmatrix}$$

Iteration 2:

Step 1: From the above resultant graph after iteration 1, path from source 1 to destination 5 is 1 → 2 → 3 → 4 → 5. Here maximum allowed flow rate of individual links 1 → 2, 2 → 3, 3 → 4 and 4 → 5 are 3, 4, 5 and 4 respectively.

From the above flow rates of the individual links 3, 4, 5 and 4, minimum allowed flow rate is 3. Here, the maximum allowed flow rate from source node 1 to destination node 5 is **3**.

Step 2: Now delete the link 1 → 2 which has the minimum allowed flow rate among all the individual links in the considered path 1 → 2 → 3 → 4 → 5.

The resultant graph after iteration 2 is shown in fig.3:

DOI 10.17148/IJARCCE.2018.7611

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified

Vol. 7, Issue 6, June 2018

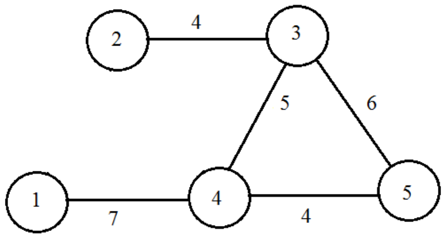

Fig. 3. Residual network after 2nd iteration

The corresponding matrix for the above graph is:

$$\begin{bmatrix} 0 & \infty & \infty & 7 & \infty \\ \infty & 0 & 4 & \infty & \infty \\ \infty & 4 & 0 & 5 & 6 \\ 7 & \infty & 5 & 0 & 4 \\ \infty & \infty & 6 & 4 & 0 \end{bmatrix}$$

Iteration 3:

Step 1: From the above resultant graph after iteration 2, path from source 1 to destination 5 is 1 → 4 → 5. Here, maximum allowed flow rate of individual links 1 → 4, and 4 → 5 are 7 and 4 respectively.

From the above flow rates of the individual links 7 and 4, minimum allowed flow rate is 4. Here, the maximum allowed flow rate from source node 1 to destination node 5 is **4**.

Step 2: Now delete the link 4 → 5 which has the minimum allowed flow rate among all the individual links in the considered path 1 → 4 → 5.

The resultant graph after iteration 3 is shown in fig.4:

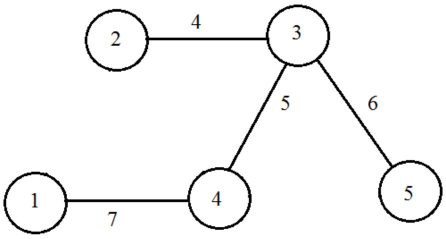

Fig. 4. Residual network after 3rd iteration

The corresponding matrix for the above graph is:

$$\begin{bmatrix} 0 & \infty & \infty & 7 & \infty \\ \infty & 0 & 4 & \infty & \infty \\ \infty & 4 & 0 & 5 & 6 \\ 7 & \infty & 5 & 0 & \infty \\ \infty & \infty & 6 & \infty & 0 \end{bmatrix}$$

Iteration 4:

Step 1: From the above resultant graph after iteration 3, path from source 1 to destination 5 is 1 → 4 → 3 → 5. Here maximum allowed flow rate of individual links 1 → 4, 4 → 3 and 3 → 5 are 7, 5 and 6 respectively.

From the above flow rates of the individual links 7, 5 and 6, minimum allowed flow rate is 5. Here, the maximum allowed flow rate from source node 1 to destination node 5 is **5**.

Step 2: Now delete the link 4 → 3 which has the minimum allowed flow rate among all the individual links in the considered path 1 → 4 → 3 → 5.

The resultant graph after iteration 4 is shown in fig.5:

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

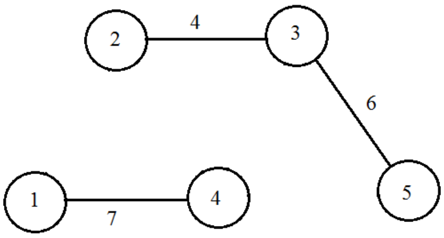

Fig. 5. Residual network after 4th iteration

The corresponding matrix for the above graph is:

$$\begin{bmatrix} 0 & \infty & \infty & 7 & \infty \\ \infty & 0 & 4 & \infty & \infty \\ \infty & 4 & 0 & \infty & 6 \\ 7 & \infty & \infty & 0 & \infty \\ \infty & \infty & 6 & \infty & 0 \end{bmatrix}$$

Iteration 5:
From the above resultant graph after iteration 4, we can observe that there exists no path from source 1 to destination 5. Therefore, algorithm terminates.

Maximum flow rates in iteration 1, 2, 3 and 4 are 2, 3, 4 and 5 respectively. Among all the maximum flow rates obtained from iteration 1, 2, 3 and 4, the maximum allowed flow rate is 5. Hence, it shows that the maximum allowed flow rate from source 1 to destination 5 is 5. The corresponding path for this is 1 → 4 → 3 → 5.

V. TIME COMPLEXITY

The efficiency of the proposed algorithm depends on the data structures chosen for the graph itself and for the priority queue of the set $V - V_T$, whose vertex priorities are the maximum allowed data rate to the nearest tree nodes. If a graph is represented by its adjacency linked lists and the priority queue is implemented as a min-heap, the running time of our algorithm is in $\mathbf{O(|E|^2 \log |V|)}$ approximately.

VI. CONCLUSION

Maximum flow rate algorithm is based on obtaining a path with maximum flow rate. Proposed algorithm uses classical Prim's algorithm to find a path from source to destination in a given graph. Here assumption is the data sent from source to destination without buffering. The maximum allowed flow rate is computed in all the paths and finally maximum of maximum allowed flow rate is selected. We have analyzed the algorithm for different number of nodes in the graph and obtained the time complexity.

REFERENCES

[1] Prim, R.C. Shortest connection networks and some generalization. *Bell System Technical Journal*, vol. 36, no 1, 1957, 1389-1401.
[2] D. Cheriton and R. E. Tarjan: Finding minimum spanning trees. generalization. In: SIAM Journal on Computing, 5 (Dec. 1976), pp. 724–741.
[3] Eugene Lawler (2001). "4.5. Combinatorial Implications of Max-Flow Min-Cut Theorem, 4.6. Linear Programming Interpretation of Max-Flow Min-Cut Theorem". *Combinatorial Optimization: Networks and Matroids*. Dover. pp. 117–120. ISBN 0-486-41453-1.
[4] Christos H. Papadimitriou, Kenneth Steiglitz (1998). "6.1 The Max-Flow, Min-Cut Theorem". *Combinatorial Optimization: Algorithms and Complexity*. Dover. pp. 120–128. ISBN 0-486-40258-4.
[5] L.Ford and D.Fulkerson. "Maximal flow through a network", Canadian Journal of Mathematics,1956. [CrossRef]
[6] A.V. Goldberg and R.E. Tarjan. "A new approach to the maximum flow problem", STOC, 1986.