IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A NEW APPROACH TO MAXIMUM FLOW OF MIN CUT THEOREM BY PROPOSED ALGORITHM

[1]Krithika.S, [2]Narmatha.S, [3]Murali Dharan.K

[1]PG Scholar, [2]Assistant Professor, [3]PG Scholar

[1]Department of Mathematics

[1]Sri Krishna Arts and Science College, Coimbatore, India

Abstract: The maximum number of flows that can be sent from the source to the sink is addressed in Maximum Flow Problem. The algorithm for Edmonds-Karp is a modified version of the algorithm for Ford-Fulkerson. The solution to this has also been demonstrated by the use of the proposed method to justify the utility of the approach proposed. Since certain parts of its protocol are left unspecified, Ford-Fulkerson is often called a process. On the other hand, Edmonds-Karp offers complete specifications. Most notably, it states that during the intermediate phases of the programme, breadth-first search should be used to find the shortest paths. Like Ford-Fulkerson, Edmonds-Karp also deals with max- flow min - cut problems. The maximum flow in Edmonds-Karp algorithm is a highly polynomial time algorithm. This algorithm uses BFS to find augmenting paths.

Keywords **- Minimum cut of maximum flow, Edmonds-Karp Algorithm, augmenting path, BFS**

I. INTRODUCTION

In this paper, by the proposed algorithm, we prove and estimate maximum network flow, the input is a directed graph G = (V, E) with two special vertices s and t, called the source and goal. The problem of maximum flow needs the maximum rate at which a resource can be transferred from s to t; the problem of minimum cut requires the minimum damage required to separate s from t. A special case of network flow problems (circulation problem) can be seen as the maximum flow problem. As shown in the max-flow min-cut theorem, the maximum value of the s-t flow i.e. the flow from source s to sink t) is equal to the minimum capacity of the s-t cut (i.e. the cut separating s from t) in the network.

Breadth-first search is an algorithm used for tree or graph data structures to traverse or search. It begins at the root of the tree (or any random graph node, often referred to as a 'search key') and examines all of the neighbouring nodes at the current depth before moving to the next depth level of the nodes.

Disjunctive constraints can increase the maximum flow problem: a negative disjunctive constraint means that a certain pair of edges cannot have a nonzero flow at the same time; a positive disjunctive constraint states that in a certain pair of edges at least one must have a nonzero flow. With negative constraints, even for simple networks, the issue becomes highly NP-hard. For positive constraints, if fractional flows are permitted, the problem is polynomial, but may be strongly NP-hard when the flows must be integral.

II. PRELIMINARIES

Definition 1

A **network** is a graph G = (V, E) with a vertex source s ∈ V and a vertex sink t ∈ V. Each edge e = (v, w) has a defined capacity, denoted by u(e) or u (v, w). Defining potential for any pair of vertices (v, w) ∉ E with u (v, w) = 0.

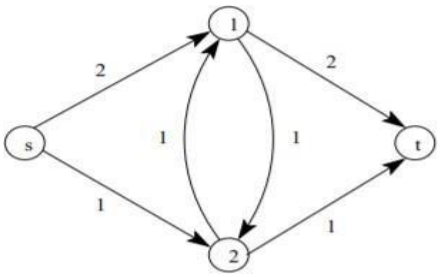

Fig.1

A network illustration with n = 4 vertices and m = 6 edges. The capacities of the edges are illustrated. We allocate a flow to each edge in a network flow problem. There are two ways in which a stream can be defined: raw (or gross) flow and net flow.

Definition 2

Raw flow is a r (v, w) function that satisfies the following properties:

- **Conservation:** For all vertices, the total flow entering v must equal the total flow leaving v. Other than s and t.

$\sum w \in V$

- **Capacity restriction:** The flow along every edge must be positive and less than that edge's capacity.

$0 \leq r(v, w) \leq u(v, w)$

Definition 3

Net flow is a function that meets the conditions below:

- **Symmetry of skew:** $f(v, w) = -f(w, v)$.
- **Conservation:** $\sum_{w \in V} f(v, w) = 0$, for all $v \in V \setminus \{s, t\}$.
- **Restriction of capacity:** $f(v, w) \leq u(v, w)$ for all $v, w \in V$.

It is possible to transform a raw flow r (v, w) into a net flow through the formula $f(v, w) = r(v, w) - r(w, v)$. For instance, if we have 9 flow units from v to w and 5 flow units from w to v, then f (v, w) = 4 is the net flow from v to w.

Definition 4

An **augmentation path** is a simple path through the graph using only edges with positive capacity from the source to the sink - a path that does not include cycles.

An augmented path is a directed path in the residual network G_f from node s to node t.

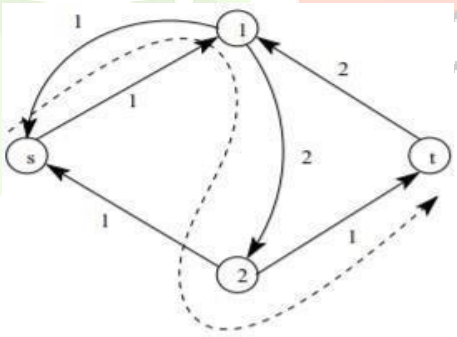

Fig.2

Note that if we have an augmenting path in G_f, then this means we can push more flow along such a path in the original network G. If we have an augmentation path (s, v_1, v_2, . . .v_k, t), to be more precise, the maximum flow we can push along that path is min {u_f (s, v_1), u_f (v_1, v_2), u_f (v_2, v_3), … u_f (v_{k-1}, v_k) and u_f (v_k, t)}. Therefore, if an augmentation path in G_f exists for a given network G and flow f, then the flow f is not a maximum flow. More generally, if f ′ in G_f is a feasible flow, then f + f′ in G is a feasible flow. Since flow conservation is linear, the flow f + f ′ still satisfies conservation. The flow f + f ′ is possible since the inequality f ′ (e) ≤ uf (e) = u(e) − f(e) can be rearranged to get f ′ (e) + f(e) ≤ u(e). If f ′, on the other hand, is a feasible flow in G, then the flow in G_f is a feasible f – f ′. We can state and prove the max-flow min-cut theorem using residual networks and augmenting paths.

III. MAX -FLOW MIN-CUT THEOREM

3.1 LEMMA

The following conditions are equivalent in a flow network G:

1. A flow F is a flow maximum.
2. There are no augmentation paths for the residual network G_f.
3. For some S. cuts, |f| = u(S)

These conditions imply that the maximum flow value is equal to the minimum cut value of s- t: max f |f|= min u(S), where F is a flow and S is a cut value of s-t.

Proof:

We show that the other two are implied by each condition.

1 ⇒ 2: If G_f is an increasing path, then we have previously argued that we can push extra flow along that path, so f was not a maximum flow. The counter positive of this statement is 1 ⇒ 2.

2 ⇒ 3: s and t must be disconnected if there are no augmenting paths in the residual network G_f. Let S = {vertices which can be reached from s in G_f}. Since, t is not reachable, an s-t cut is defined by the set S.

Network G_f has been disconnected. All nodes that can be reached from s are included in the set S. By nature, all edges (v, w) that cross the cut have a residual capacity of 0. This implies that these edges have f (v, w) = u in the original network G (v, w). Hence, |f| = f(S) = u (S).

3 ⇒ 1: If we know f needs to be a maximum flow for any cut S, |f| = u(S). Alternatively, we will have a g flow with |g| > u (S),

We know that, contrary to (1) and (3), the maximum flow cannot be less than the minimum cut value, since for some S, |f| = u(S) and u(S) are at least as high as the minimum cut value.

The maximum flow should not be greater than the minimum cut value, Lemma tells us. The maximum flow value and the minimum cut value, therefore, are the same.

3.2 THEOREM

In a minimum cut, the maximum volume of flow passing from the source to the sink in a flow network equals the total weight of the edges, i.e. the smallest total weight of the edges that will disconnect the source from the sink if removed.

Proof:

The theorem applies to two quantities: the maximum flow through the network and the minimum cutting power of the network i.e., the minimum flow capacity.

Each of these quantities must first be specified to state the theorem.

Let N = (V, E) be a directed graph where the set of vertices is denoted by V, and the set of edges is E. Let s ∈ V and t ∈ V respectively be the source and the sink of N.

An edge capability is a c: E → R^+ mapping denoted by c_{uv} or c (u, v) where u, v ∈ V. The maximum amount of flow that can move through an edge is represented.

Flows:

A flow is a f: E → R+ mapping, denoted by f_{uv} or f (u, v), subject to the following two limitations:

1. Capacity Constraint: In E, $f_{uv} \leq c_{uv}$ for any edge (u, v)
2. Conservation of Flows: The following equality holds for each vertex v apart from s and t (i.e. the source and sink, respectively)

$$\sum\{u:(u,v)\in E\} f_{uv} = \sum\{w:(v,w)\in E\} f_{vw}$$

A flow can be visualized across the network as a physical flow of a fluid, following the path of each boundary. The power constraint then states that the volume flowing through each edge per unit time is less than or equal to the edge's maximum capacity, and the conservation constraint states that, apart from the source and sink vertices, the amount flowing into each vertex equals the amount flowing out of each vertex.

A flow value is defined by,

$$|f| = \sum\{v:(s,v)\in E\} f_{sv} = \sum\{v:(v,t)\in E\} f_{vt}$$

Whereas above the source node is s and the sink node is t. In the fluid analogy, it represents the volume of fluid at the source node entering the network. This is the same as the sum of flow exiting the network at the sink node because of the conservation axiom for flows.

The problem of maximum flow asks for the largest flow on a given network.

Maximize | f |, that is, to route as much flow from s to t as possible.

Cuts:

A different element of a network corresponds to the other half of the max-flow min-cut theorem: the collection of cuts. An s-t cut C = (S, T) is a V partition that is s ∈ S and t ∈ T. That is, s-t cut is a division of the network vertices into two parts, with one part of the source and the other part of the sink. A cut C's cut-set XC is the set of edges connecting the source portion of the cut to the sink portion:

$$X_C = \{(u, v) \in E: u \in S, v \in T\} = (S \times T) \cap E$$

Then, if all the edges in the cut-set C are removed, then no positive flow is possible, because in the resulting graph there is no path from the source to the sink.

The total weight of its edges is the capacity of an s-t cut,

$$C(S, T) = \sum_{\{(u,v) \in X_C\}} C_{uv} = \sum_{(i,j) \in E\}} c_{ij}\, d_{ij}$$

Where we have as dij = 1 if i ∈ S and j ∈ T, 0 otherwise.

In a graph, there are usually several cuts, but cuts with smaller weights are often harder to find.

s-t minimum cut issue. minimize c (S, T), i.e. evaluate S and T in such a way that the S-T cut capacity is minimal.

The theorem therefore relates the maximum flow through a network to the minimum network cut.

Over all s-t cuts, the maximum value of the s-t flow is equal to the minimum power.

Example:

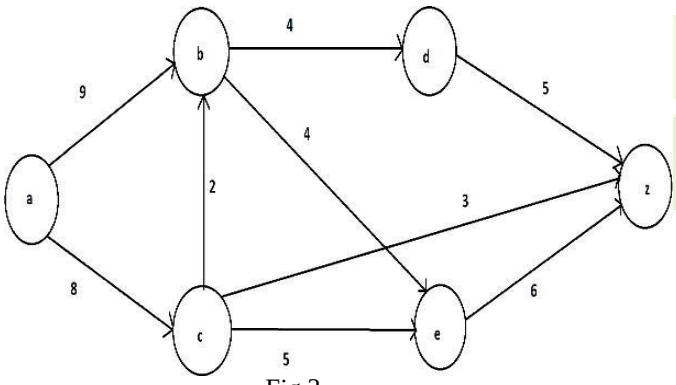

Fig.3

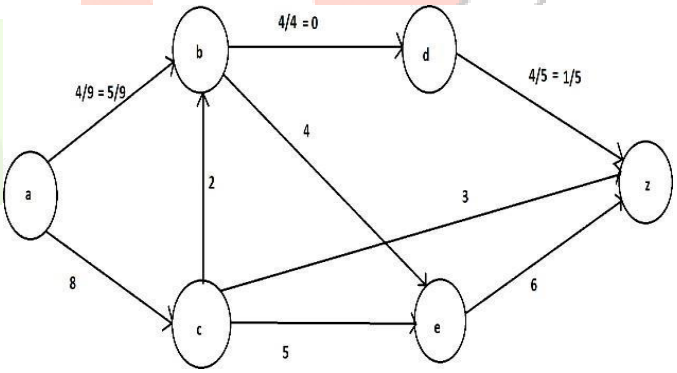

Fig.4

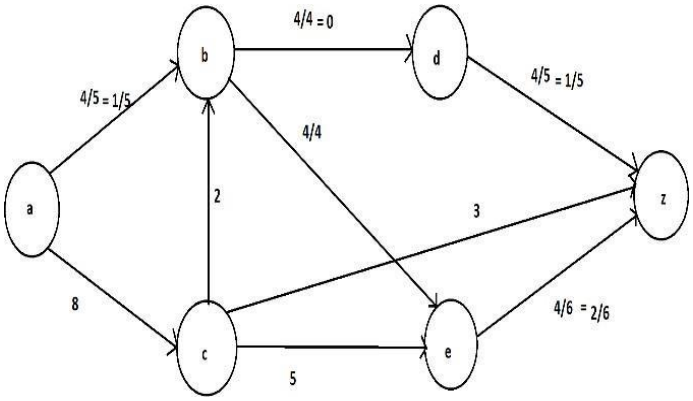

Fig.5

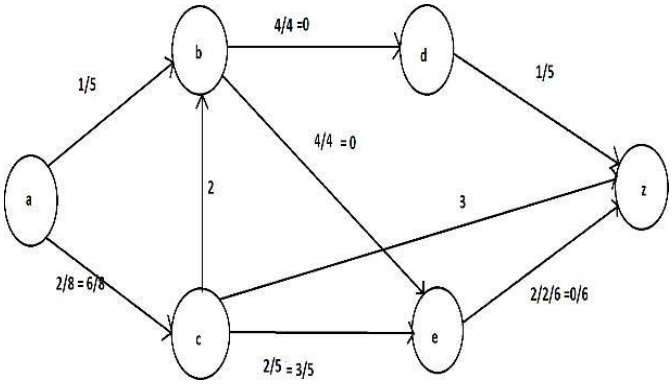

Fig.6

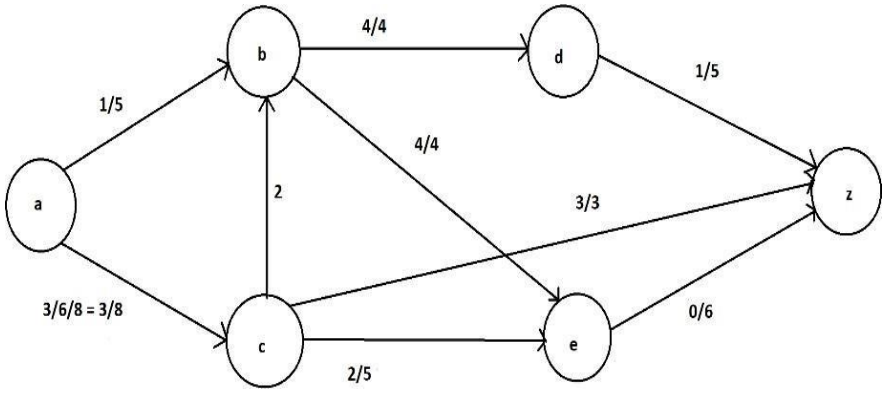

Fig. 7

Total number of augment path = 4 +4+2+3 = 13 is the maximum flow of the network. Cut:
Set of edges whose elimination splits the network into x and y halves: Where s ∈ x and t ∈ y.

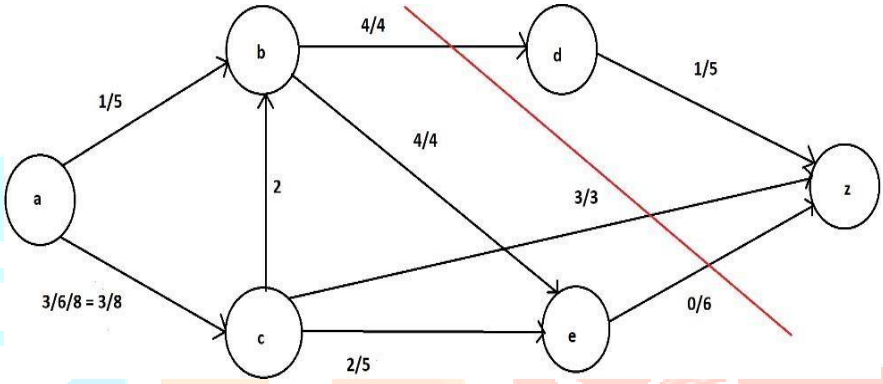

Fig.8

The red line given is to cut the graph from source to sink and we add the total capacities of the red line passed.

(i. e): 4 +3+6 = 13.

Therefore, the maximum flow is 13 and minimum s-t cut is 13.

IV. IMPORTANCE OF THE MAX-FLOW MIN-CUT THEOREM

Theorem:

The max-flow is the flow returned by Ford-Fulkerson f *.

Proof:

We have $|f| \leq c(S, T)$ for any flow f and $s - t$ cut (S, T).

The flow f* is such that the max-flow is $|f^*| = c(S^*, T^*)$, for some $s - t$ cut $(S^*, T^*) \Rightarrow f^*$. The value of the max $s - t$ flow for any (G, s, t, c) is therefore equal to the ability of the minimum $s - t$ cut.

V. EDMONDS-KARP ALGORITHM

A particular implementation of the Ford-Fulkerson algorithm is the Edmonds-Karp Algorithm. Edmonds-Karp is also an algorithm that deals with the max-flow min-cut problem, like Ford- Fulkerson. For the max-flow problem, it is a polynomial time algorithm.

Since certain parts of its protocol are left unspecified, Ford-Fulkerson is often called a process. A complete specification is given by Edmonds-Karp, on the other hand. Most notably, it states that during the intermediate phases of the program, breadth-first search should be used to find the shortest paths.

Edmonds-Karp increases Ford-run Fulkerson's time, which is O ($|E|. f^*$), to O ($|V|. |E|^2$).

For example:

Augmenting routes are essentially any route that can currently take further flow from the source to the sink. Flow is monotonically augmented throughout the course of the algorithm. So, there are periods when more flow will take place on a path from the source to the sink, and that is an increasing path. As we let edges have unit length, this can be found via a breadth- first search. The running time of O ($|V|. |E|^2$) is discovered by demonstrating that each augmentation path can be discovered in O(E) time, that each time at least one of the E edges is saturated, that the distance from the saturated edge to the source along the augmentation path must be longer than the last time it was saturated, and that the maximum length isV.

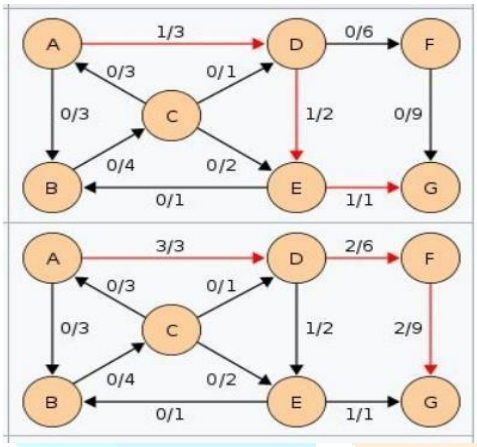

Fig.9

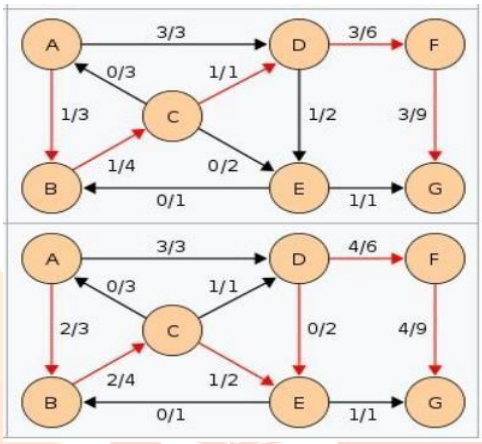

Fig.10

Notice how the length of the algorithm's augmentation path (in red) never decreases. The routes listed are the shortest possible. In the graph separating the source and the sink, the flow found is equal to the capacity over the minimum cut. In this graph, there is only one minimal cut, partitioning the nodes with the potential into the sets {A, B, C, E} and {D, F, G}.

OVERVIEW

The Ford-Fulkerson algorithm was first resolved to find the maximum flow for a network. A network is also represented abstractly as a graph, G, which has a set of vertices, V, linked by a set of edges, E. There is a source, s, and a drain, t, which represents where and where the flow comes from. It was solved through the max-flow min-cut theorem to find the maximum flow through a network, which was then used to construct the Ford-Fulkerson algorithm.

Except for one very significant feature, Edmonds-Karp is similar to Ford-Fulkerson. The search order of the paths to augment is well established. Augmenting routes, along with residual graphs, are the two important concepts to understand when seeking the full flow of a network as a refresher from the Ford-Fulkerson.

Augmenting routes are essentially any route that can currently take further flow from the source to the sink. Flow is monotonically augmented throughout the course of the algorithm. So, there are periods when more flow will take place on a path from the source to the sink, and that is an increasing path.

From now on, when in a network we refer to a "shortest path," we mean a path that uses the fewest edges, and the number of edges is the "length" of a path. The length of the shortest path from s to the vertex is the "distance" of a vertex from s.

BFS can be implemented in O (|V + E|) = O(|E|) time, and so we need to find an upper bound to the number of possible iterations to complete our study of the algorithm.

The architecture of BFS algorithm:

CONCEPT DIAGRAM

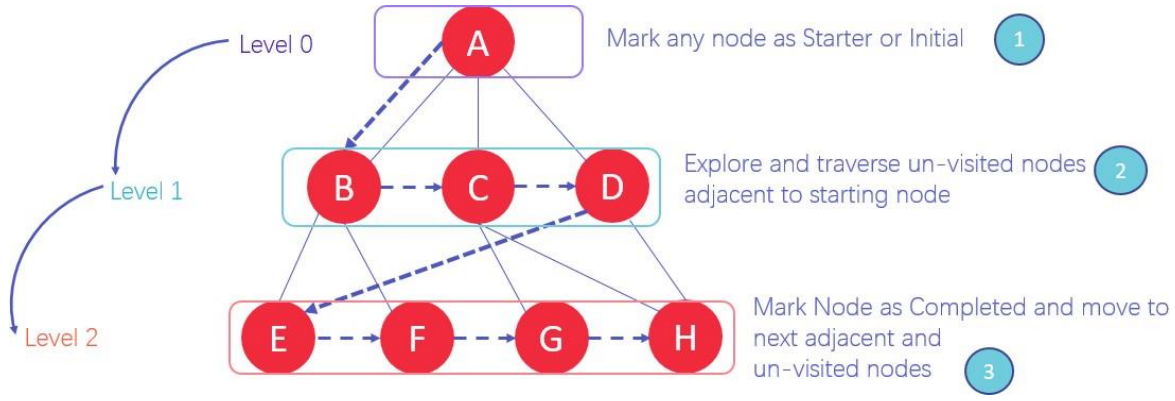

Fig.11

The following theorem states that for each |E| iterations, the length of the shortest path from s to t in the residual network can only increase over the different iterations, and it increases at the rate of at least one extra edge in the shortest path.

5.1 THEOREM

If the length of the shortest path from s to t in the residual network at a certain iteration is *l*, then it is ≥ *l* at each subsequent iteration. In addition, the distance becomes ≥ *l* + 1. after a maximum of |E| iterations.

Clearly, as long as there is a path from s to t, the distance from s to t is |V|-1 at most, and so this theorem tells us that s and t must be disconnected in the residual network after |E|·(|V|-1) iterations at most at which point the algorithm stops. O(|E|) takes time for each iteration, so the cumulative runtime is O (|V|. $|E|^2$). Let us prove the theorem now.

Proof:

Suppose that we have the residual network R = (V, E'), s, t, c ') after some number of iterations T and that the length L of the shortest path from s to t in the residual network is l. Create a BFS tree starting at s, and call V_1, V_2, ..., V_k, ..., the vertices on the tree's first, second, k-th sheet, i.e., the vertices whose distance from s is 1, 2, ... etc. Notice that any edge (u, v) in the network is such that nodes can go from higher-numbered layer to lower-numbered layer if u ∈ V_i and v∈ V_j then j ≤ i + 1, or remain in the same layer, or advance by at most one layer. If, for any i, u ∈ V_i and v ∈ V_{i+1}, let's call an edge (u, v) a forward edge. Then, at each step, the shortest path from s to t must use a forward edge and, equivalently, A route that uses a non-forward edge at any point might not be the shortest path between s and t. What happens in the next T +1 iteration? We select one of the paths of length-*l* p from s to t, and we drive flow through it. At least one of the edges in p disappears in the next residual network because it has been saturated, and we see edges moving in the reverse direction for each edge of p. Now it is still true that for every edge (u, v) of the residual network at the next stage T +1, if u ∈ V_i and v ∈ V_j, then j ≤ i+1 (where V1, ... are the layers of the network's BFS tree at iteration T) since all the edges we have added actually go from higher-numbered layers to lower-numbered ones. This implies that the distance of t from s is still at least *l* at the iteration T + 1 since t ∈ V_l and we can advance at most by one layer at each step on a path.

Note: We have proved that if at one iteration the distance from t to s is l, then at the next iteration it is at least l. By induction, this is enough to conclude that if, in all subsequent iterations, there will always be at least l.

In addition, if there is a length- *l* path from s to t in the residual network at iteration T + 1, then only edges that were already present in the residual network at iteration T and were "forward edges" at iteration T must be used for the path. This also suggests that it is so because there is a length-*l* path made entirely of edges that were forward edges at iteration T, in all the subsequent iterations, s to t distance remains *l*. However at least one of those edges is saturated at each iteration and is absent in subsequent steps from the residual network, so there can be at most |E| iterations during which the distance from s to t remains *l*.

5.2 EDMONDS-KARP ALGORITHM FOR MAXIMUM FLOW

We define the order in which the paths must be taken, just a tiny tweak to the Ford- Fulkerson algorithm. Second, go from the source to the sink by using the shortest paths. This is similar to a BFS finding with all weight as 1, the path from source to sink.

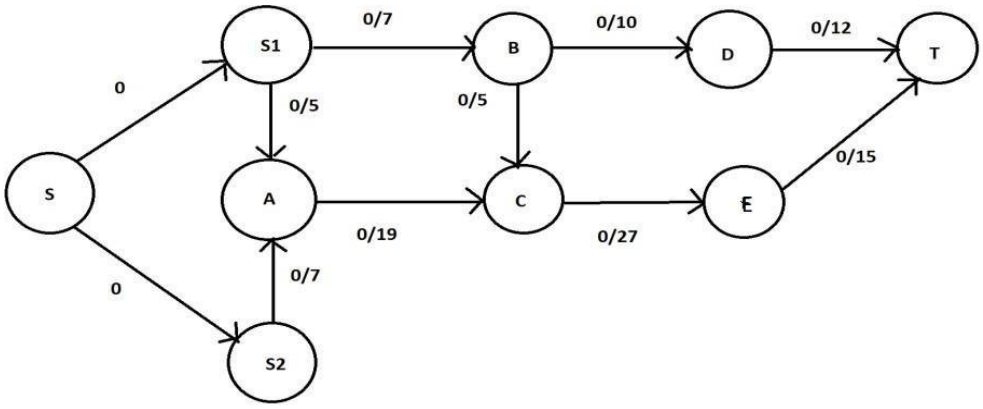

Fig.12

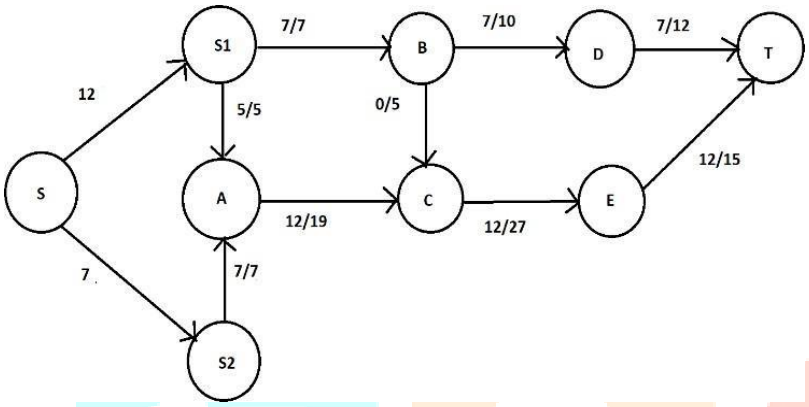

Fig.13

Augmentation Path	Flow
S S_1 B D T	7
S S_1 A C E T	5
S S_2 A C E T	7
Total	19

Fig.14

Key points:

- Path lengths for augmentation never decrease.
- One more edge is critical after each iteration.
- It takes |E| time for each augmentation.

Run time of this algorithm:

- At least one edge is critical for each magnification.
- Set a vector, n, so that the nth vertex from the source is the vertex before this edge.
- Thus, the n+1th vertex is the next vertex.

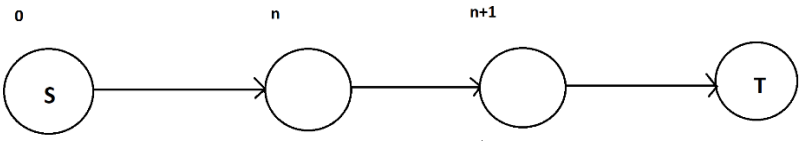

Fig.15

- It must be in the opposite direction, a decline, in order for the edge to be crossed again.
- At least the n + 1st vertex in the new augmentation direction must be the vertex after the edge.

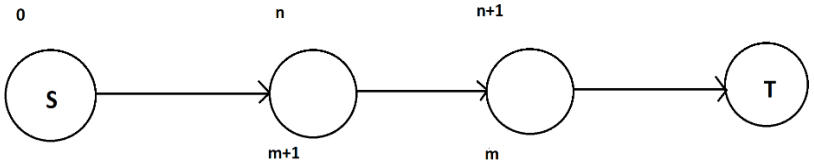

Fig.16

where m ≥ n+1

- Therefore, the first vertex must be further away from the source than n+2.
- From the source, every vertex can be |V|-1 at most.
- If the vertex is on the augmentation path before the edge, it must be |V|-2 from the source, at most.
- An edge can therefore be at most (|V|-2)/2 times the critical edge on the path.
- |E| edges are available, each of which can only be the critical edge (|V|-2)/2.
- A critical edge is formed by each enlargement.
- Only |E|(|V|-2)/2 augmentations can be made.
- As each edge is traversed at most one, each augmentation takes O(|E|) times.
- Therefore, the complexity of the Edmonds-Karp algorithm is shown to be:

$$O(|V|.|E|^2)$$

VI. CONCLUSION

In this paper, we conclude that maximum flow of the network is equal to minimum s-t cut. We used an algorithm called Edmonds-Karp algorithm to find the maximum flow of the network. It is the implementation of Ford-Fulkerson algorithm. The algorithm uses Breadth-first search (BFS) to find the shortest path of the given network first. The Breadth First Search (BFS) algorithm that traverses a graph in the motion of a big ward and uses a queue to remember when a dead end occurs in any iteration in order to start a search with the next vertex.

VII. REFERENCES

[1] A.V. Goldberg, R.E. Tarjan A new approach to the maximum flow problem J. Assoc. Comput. Mach., 35 (4) (1988), pp. 921-940.

[2] 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.

[3] F.T. Leighton and S. Rao (1988), An approximate max- ow min-cut theorem for uniform multicommodity ow problems with application to approximation algorithms", Proceedings, 29th Symposium on Foundations of Computer Science (1988), pp. 422- 431.

[4] L. R Ford, D. R Fulkerson, Maximal Flow through a Network, Research Memorandum RM-1400, The RAND Corporation, Santa Monica, California, 19 November 1954 published in Canadian Journal of Mathematics 8 (1956) 399–404.

[5] Picard J-C, Queyranne M (1980) On the structure of all minimum cuts in a network and applications. Math Program Stud 13:8-16.

[6] V.M. Malhotra, M. Pramodh Kumar, S.N. Maheshwari an $O(V^3)$ algorithm for finding maximum flows in networks Inf. Process. Lett., 7 (6) (1978), pp. 277-278.

[7] http://srmanikandasriram.github.io/files/DSA/Term-Paper.pdf.

[8] http://www.stanford.edu/class/cs97si/08-network-flow-problems.pdf.