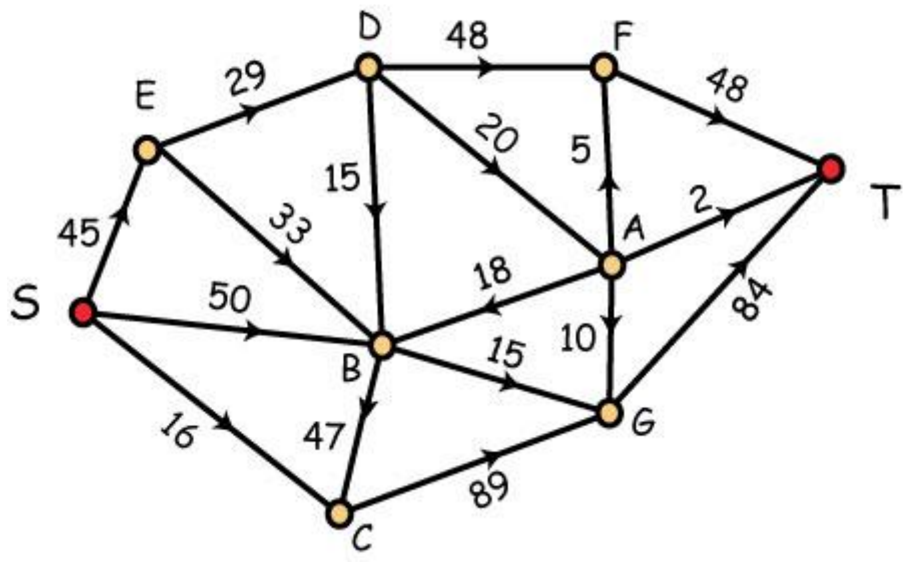
November 14, 2017

20 - Network Flows

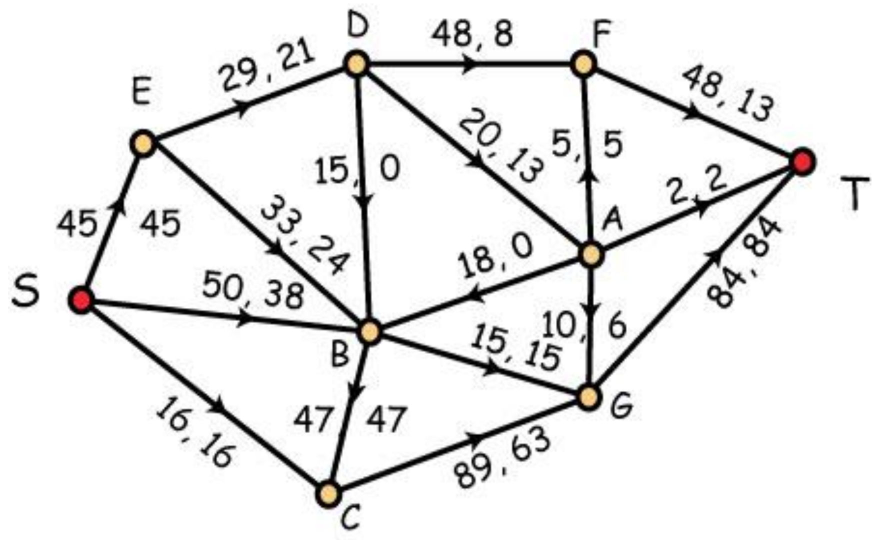
William T. Trotter
trotter@math.gatech.edu

A Network Flow Problem



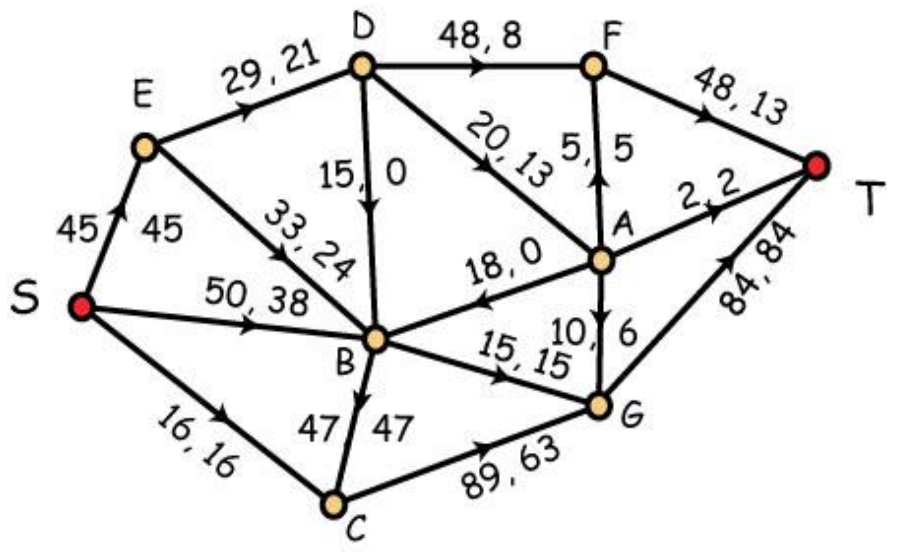
Setup We are given a digraph with positive weights on edges. There are two distinguished vertices, a source S and a sink T. All edges incident with S are oriented away from S and all edges incident with T are oriented towards T. The weights represent capacities and we will denote the capacity of edge e as c_e.

A Flow in a Network



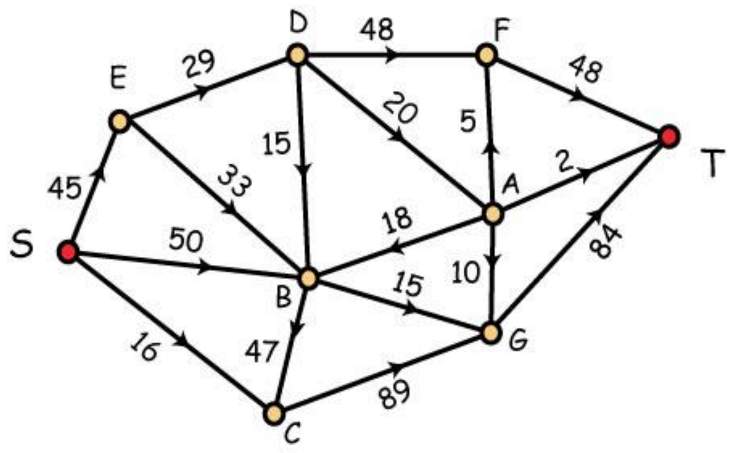
Definition A flow is an assignment of a non-negative value x_e to every edge e of the digraph subject to the following conditions: (1) $x_e \le c_e$, i.e., the flow on edge e does not exceed its capacity; (2) the total amount leaving the source = the total amount arriving at the sink; and (3) for all other vertices v, amount into v = amount leaving v.

Challenge: Find a Maximum Flow



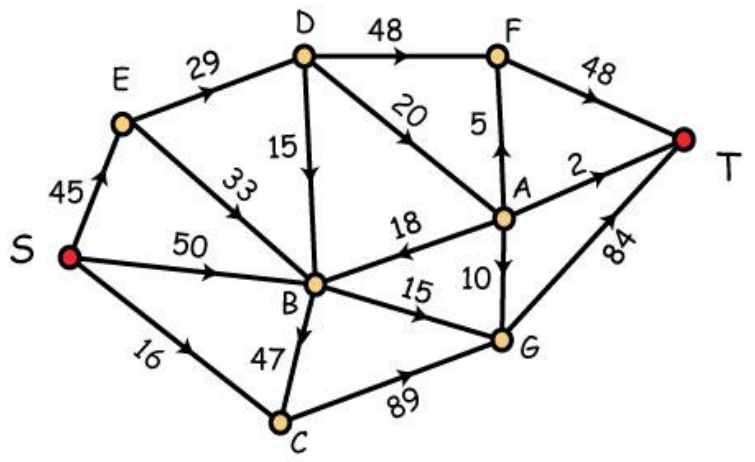
Definition The value of a flow is the amount leaving the source, which is exactly the same as the amount arriving at the sink. This flow has value 45 + 38 + 16 = 99. Our challenge then is to find a maximum flow, i.e., a flow of maximum value. Can you tell by inspection whether this flow is maximum?

Cuts in Network Flow Problems



Definitions A cut in a flow is a partition of the vertices into two subsets **L** and **R** with S in **L** and T in **R**. If the network has n vertices, there are 2^{n-2} cuts. The capacity of a cut (**L**, **R**) is the sum of the capacities of all edges from **L** to **R**. Note that we do not include the capacities of edges from **R** to **L** in this sum. Here the capacity of the cut where **L** = (S, E, B, C) and the remaining vertices are in **R** is 29 + 15 + 89 = 123.

The Max Flow/Min Cut Theorem

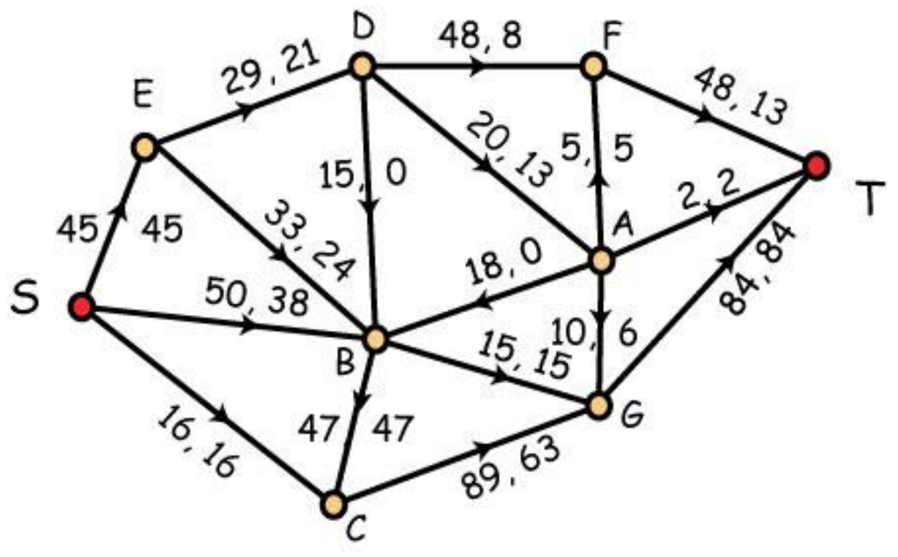


Theorem The maximum value of a flow is equal to the minimum capacity of a cut.

Observation The fact that the value of any flow is at most the capacity of any cut is an immediate (and elementary) consequence of the conservations laws and the definitions of flows and cuts.

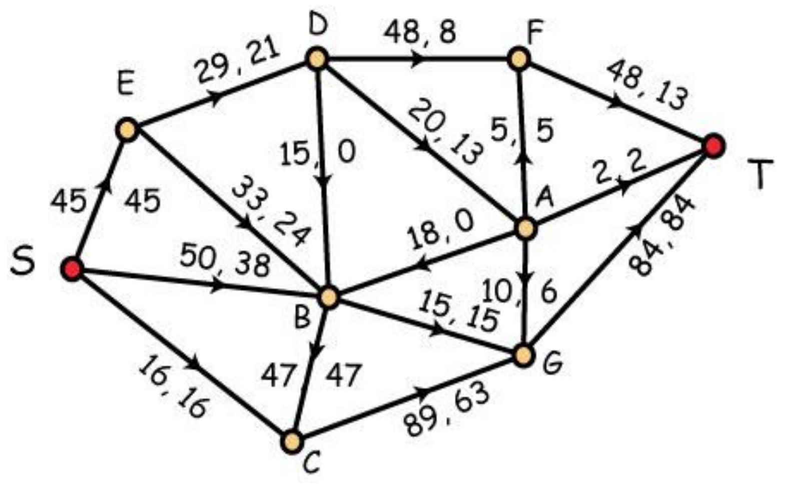
Remark So the challenge in the theorem is finding the maximum flow and the minimum cut, and this is what we will do next!

Full end Empty Edges



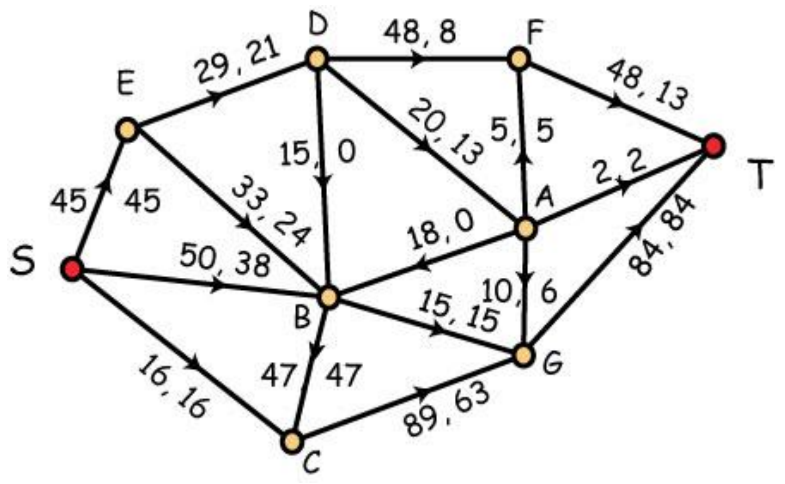
Definitions An edge is full when the flow on the edge is equal to the capacity of the edge. An edge is empty when the flow on the edge is 0. Here edges (S, E), (S, C), (B, G), (A, F), (A, T) and (G, T) are full while (D, B) and (A, B) are empty.

Augmenting Paths



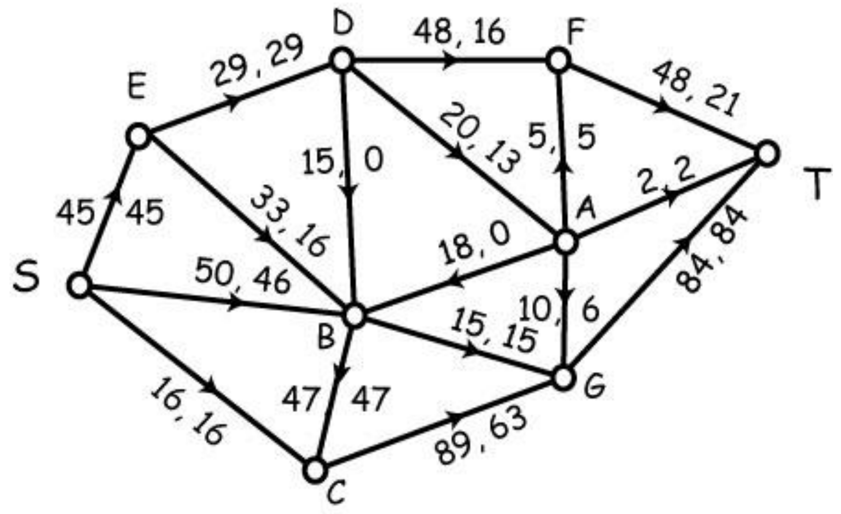
Definitions Allowing the ability to walk on an edge in the network in either direction, an ordinary path from S to T traverses some edges in a forward manner and others in a backwards manner. The first and last are always forward. The path is called an augmenting path when the forward edges are not full and the backwards edges are not empty. Here, the path (S, B, E, D, F, T) is an augmenting path.

Augmenting Paths (2)



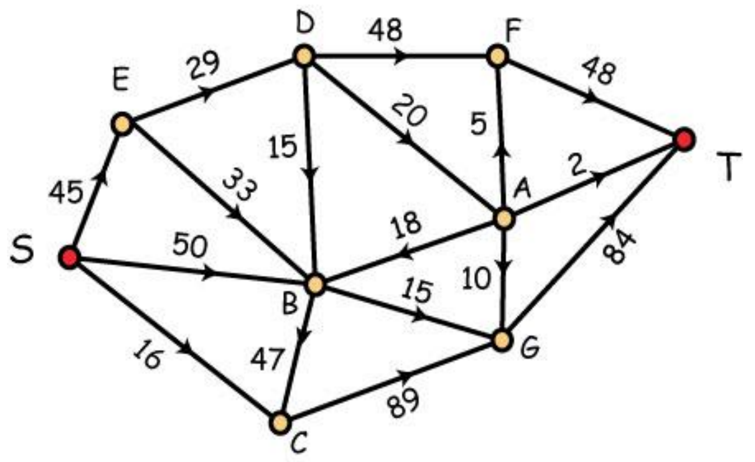
Observations A forward edge on an augmenting path has spare capacity and a backwards edge has spare flow. Let v be the minimum value among these quantities. Update the flow by increasing the flow on the forward edges by v and decreasing the flow on the backwards edges by v. For the augmenting path (S, B, E, D, F, T), the quantity v is 8 which is the spare capacity on edge (E, D).

Augmenting Paths (3)



Remark In the figure, we show the updated flow. Now the value of the flow is 99 + 8 = 107. In this new flow, do you see any augmenting paths?

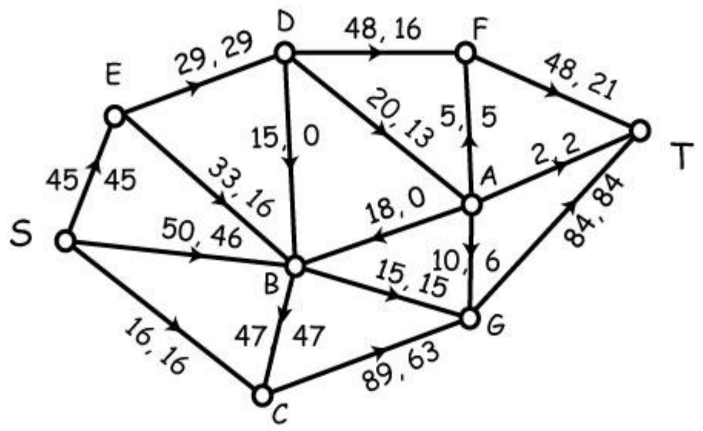
Finding the Minimum Capacity of a Cut



Theorem The maximum value of a flow is equal to the minimum capacity of a cut.

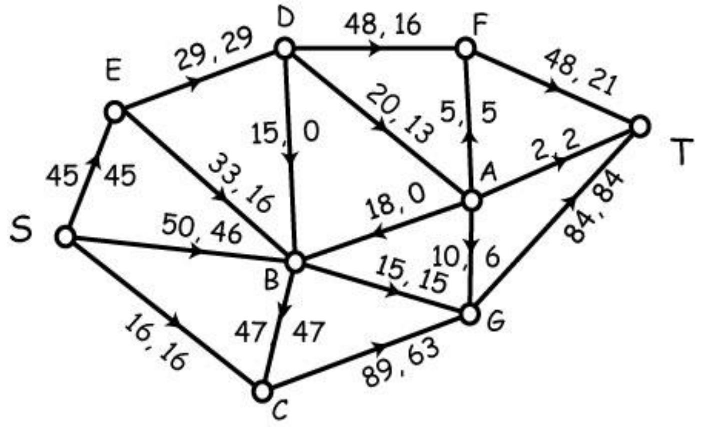
Observation If there are n vertices, there are 2^{n-2} cuts so even if we could evaluate them all to find the minimum capacity of a cut, we would gain no information about the details of a maximum flow. Remarkably, there is an efficient algorithm for finding a maximum flow and a minimum cut at the same time!

It's All About Augmenting Paths!!



Observation Let **L** denote the set of all vertices X for which there is an augmenting path from S to X, and let **R** be the remaining vertices. Then if e is an edge from L to R, it follows that e is full. On the other hand, if f is an edge from R to L, then f is empty. In this example, **L** = (S, B, E) while the remaining vertices are in **R**. Notice that the edges (E, D), (B, G), (S, C) and (B, C) are full while (B, D) and (B, A) are empty.

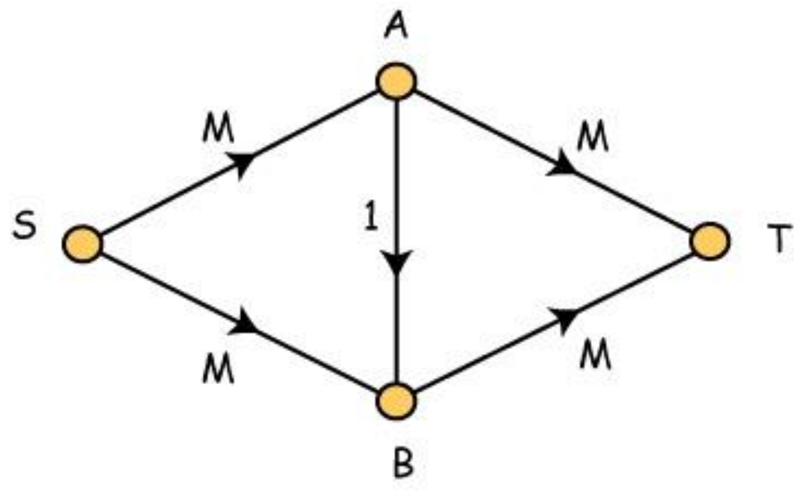
It's All About Augmenting Paths!! (2)



Observation Simple algebra now implies that when all the edges from **L** to R are full and all the edges from R to L are empty, then the value of the current flow is equal to the capacity of the cut (L, R).

Observation So all we have to do is keep finding augmenting paths and increase the flow each time we find one. Eventually, there will be no augmenting paths and we are done.

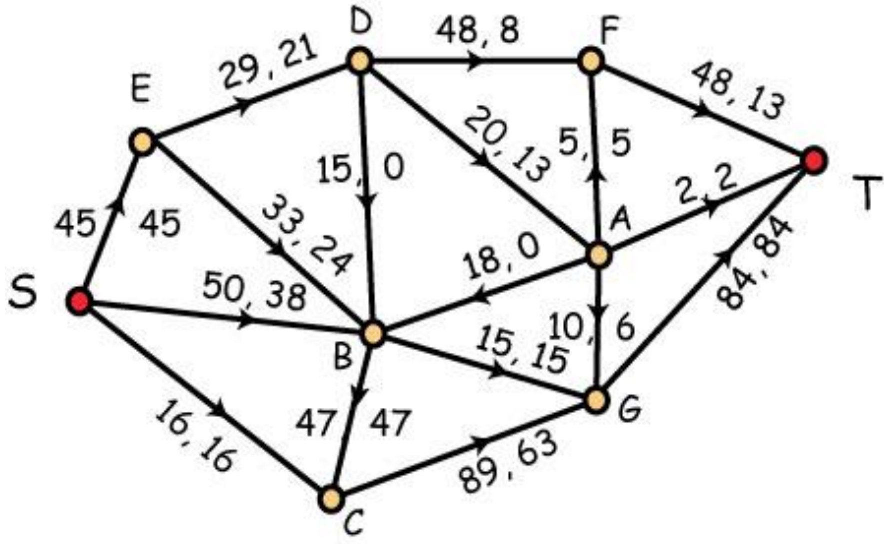
Augmenting Paths – Be Careful



Remark In the figure, suppose M = 1,000,000,000 and we advance from the zero flow using the longest augmenting path we can find. Do you see that it takes 2M steps to find the maximum flow? Furthermore, if you take shortest augmenting paths, you get to the final answer in 2 steps.

Remark This example suggests that we should focus on finding augmenting paths using the minimum number of edges.

Primal/Dual with Dijkstra as the Dual



Strategy Use Dijkstra to find an augmenting path using the minimum number of edges. Normally, this is done by carrying out the Ford-Fulkerson labelling algorithm. This algorithm assigns triples to vertices of the network, starting with a labelling of the source S. If the sink T is labelled, then we will have found an augmenting path using the minimum number of edges.

Details on the Labelling Scheme

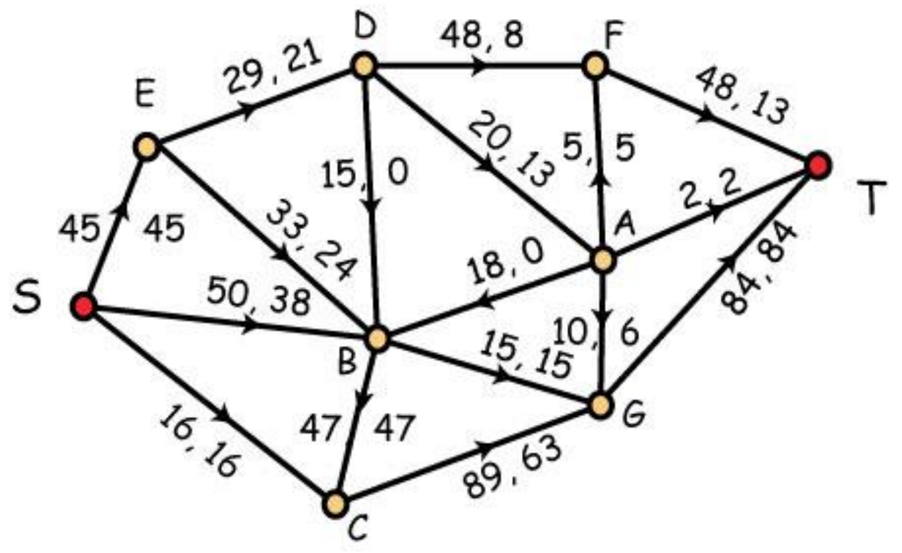
Initialize Label the source S with the triple (*, +, ∞).

Scan Using the rule: first labelled, first scanned, let X be a labelled vertex and let a(X) be the positive amount on X. Scan the neighbors on X in pseudo-alphabetic order (S, T, A, B, C, D, E, F, …). If Y is scanned an the edge is (X, Y), i.e., the edge is oriented from X to Y, and it is not full, let a(X, Y) be the spare capacity. Label Y as (X, + , min { a(X), a(X, Y)).

If the edge is (Y, X), i.e., oriented from Y to X, and the edge is not empty, let a(Y, X) denote the flow on this edge. Then label Y as (X, -, min{ a(x), a(Y, X) }).

Halt if the sink T is labeled, as backtracking will specify an augmenting path using the minimum number of edges.

Primal/Dual with Dijkstra as the Dual



Ford-Fulkerson

S (*, +, ∞)

B (S, +, 12)

E (B, -, 9)

D (E, +, 8)

A (F, +, 7)

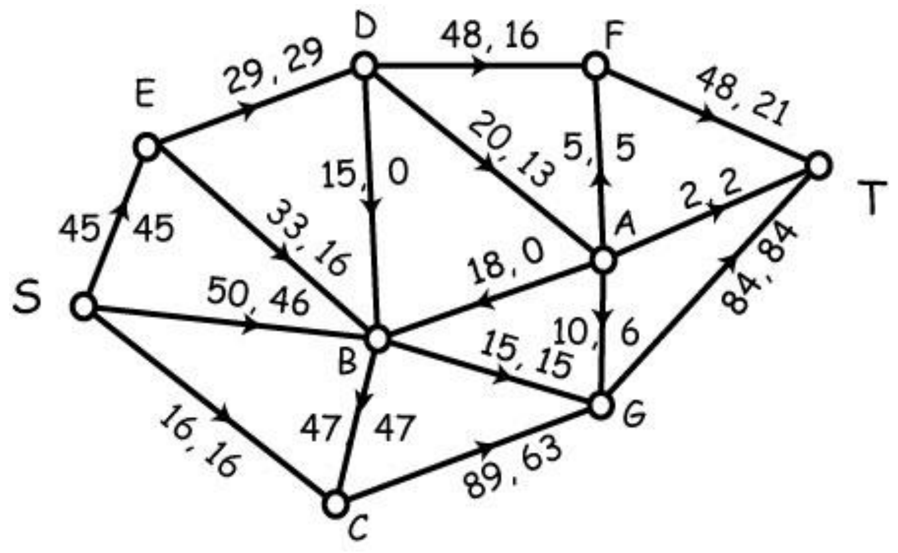
F (D, +, 8)

G (A, +, 4)

T (F, +, 8)

Augmenting Path (S, B, E, D, F, T) with value 8.

The Example Updated



Ford-Fulkerson

S (*, +, ∞)

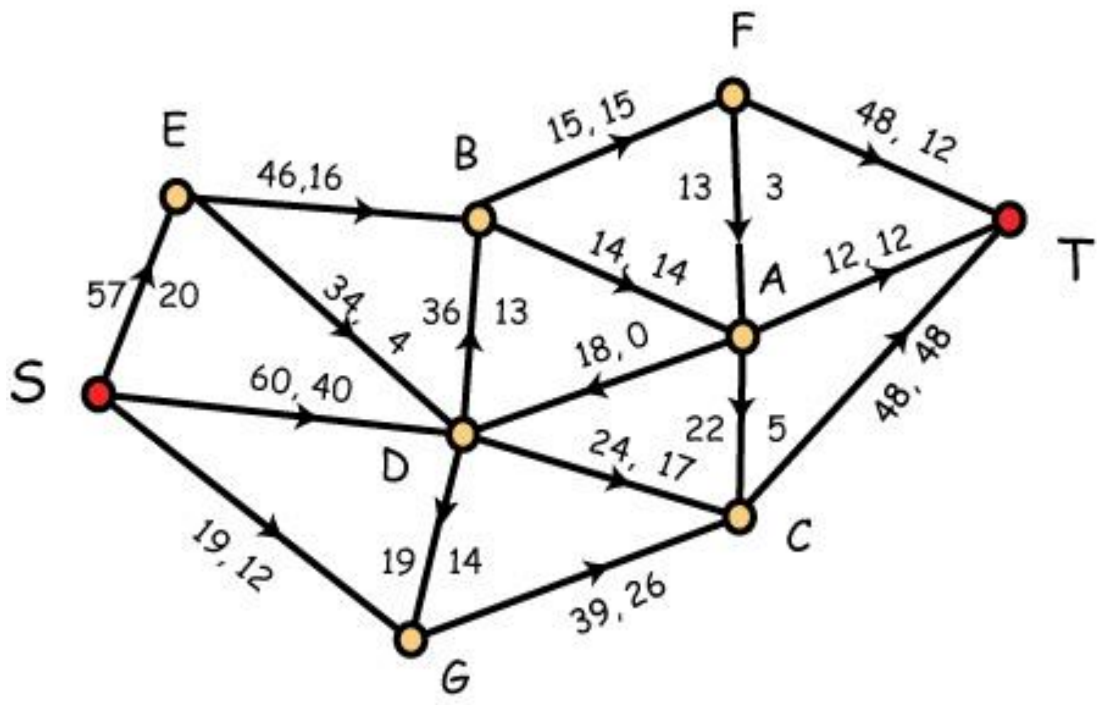
B (S, +, 4)

E (B, -, 4)

The algorithm halts with **S** = { S, B, E }. The remaining vertices belong to **R**.

Conclusion The current flow is maximum and the cut (**L**, **R**) is minimum. Done!

A Second Example



Exercise Carry out the Ford-Fulkerson labelling algorithm on the network flow.

Network Flows and Linear Programming

Definition A problem in n variables $x_1, x_2, \ldots, x_n$ is called a linear programming (LP) problem when it has the form:

Maximize $c_1x_1 + c_2x_2 + \ldots c_nx_n$

subject to m constraints of the form:

$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n \le b_i$

With all variables $x_1, x_2, \ldots, x_n$ non-negative.

Example Maximize $5x_1 + 7x_2$

subject to $14x_1 + 3x_2 \le 42$

$5x_1 + 9x_2 \le 45$

$x_1, x_2 \ge 0$

Some Observations on LP problems

Fact There are many equivalent formulations of the class of LP problems. In the set of constraints, you can allow equations and inequalities in the opposite direction. Also you call allow some or all of the variables to be negative.

Fact The solution space to the set of constraints forms a convex body in $\mathbf{R}^n$ with a bounded number of extreme (corner points). The maximum value always occurs at an extreme point. The number of extreme points can be exponentially large in terms of the number $n + m$ of variables and constraints.

Fact LP problems posed with integer constraints have solutions in the rational number system, but in general they do not have solutions with all variables being integers.

A Key Detail on Network Flow Problems

Fact A network flow problem posed with integer capacities on edges always has a maximum flow in which the flow on every edge is an integer. The proof of this fact is an immediate consequence of the fact that the Ford-Fulkerson labelling algorithm uses only addition, subtraction and minimum as its three operations.

Remark It is an important general problem to determine when optimization posed with integer constraints have integer valued solutions. Network flows are just one example.