CPSC 500 Lecture
by Will Evans on Linear Programming

Scribe : Anil Mahmud

November 4, 2013

1 Overview

In this lecture we were introduced to Linear Programming, which can be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. The constraints may be equalities or inequalities.

Apart from this the date and time of the final exam was also decided by voting.

The exam will be on:

December* 5^{th} *from 10:00 AM to 12:30 PM.

2 Introduction to Linear Programming

Linear programming can be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. The constraints may be equalities or inequalities.

Or in other words, we are given a set of variables, and we want to assign real values to them so that:

1. They satisfy some given linear equations or inequalities.
2. They maximize a given linear objective function.

2.1 “Chocolate Maker” Example

Let us look at the simple ”Chocolate Maker” example:

Background:

A chocolate maker can make two types of chocolate boxes, each with different amount of profit and demand. The data is represented in the table below.

	Profit in Dollars	Demand in Box(es) per Day
Box 1	1	≤ 200
Box 2	6	≤ 300

Constraint:

And we also have the constraint that the total number of boxes the factory can produce per day is less than or equal to 400 boxes per day.

Objective Function:

Now the question we should ask ourselves is how many boxes shold the chocolate maker produce of each type to maximize his profit. Drawing analogy to our definition this profit here is the linear objective function. And in this case we want ot maximize it.

Solution:

If we denote the number of chocolate boxes of type 1 and type 2 as X_1 and X_1 respectively, then the profit/objective function is:

$$X_1 + 6X_2$$

The constraints are :

$X_1 \leq 200$	
$X_2 \leq 300$	
$X_1 + X_2 \leq 400$	Constraint due to maximum production ability.
$X_1 \geq 0$	Number of boxes produced cannot be negative.
$X_2 \geq 0$	Number of boxes produced cannot be negative.

Using the inequalities we can draw the diagram shown in Figure 1.

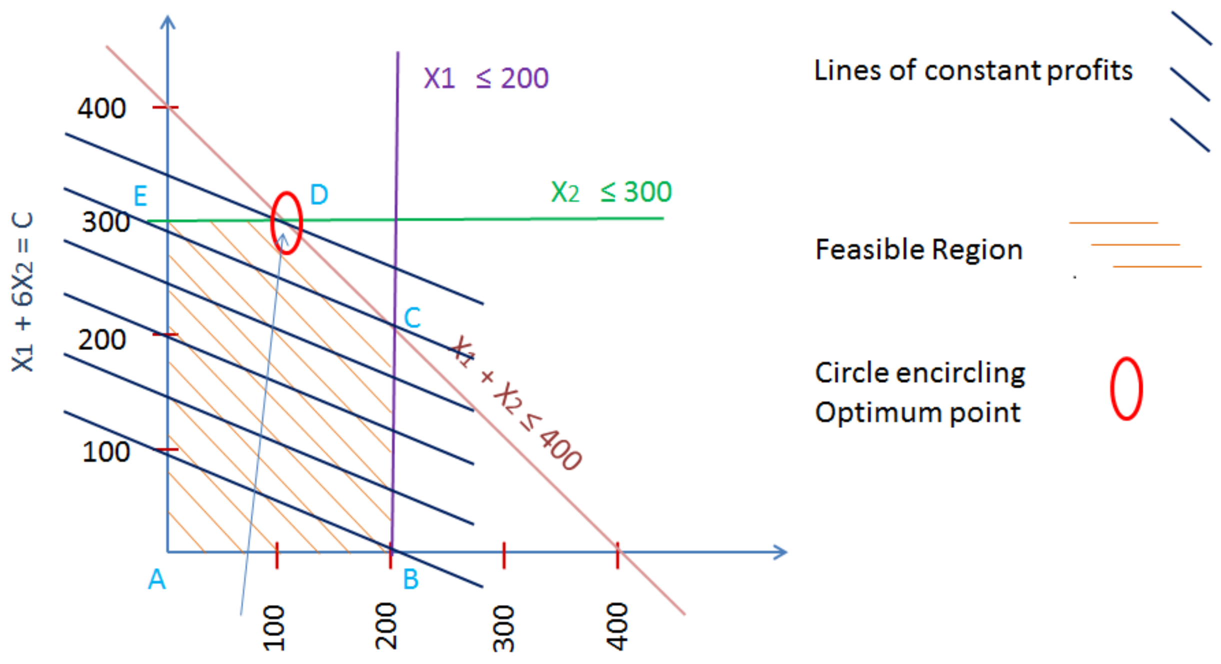


Figure 1: Diagram the Feasible Region and The Optimal Point for the "Chocolate Maker" Problem

A linear equation in X_1 and X_2 defines a line in the two-dimensional (2D) plane, and a linear inequality designates a half-space, the region on one side of the line. Thus the set of all feasible solutions of this linear program, that is, the points (X_1, X_2) which satisfy all constraints, is the intersection of five half-spaces, each corresponding to the five inequalities or constraints given above. It is a convex polygon, shown in Figure 1. The chocolate maker needs to find the point in this polygon at which the objective function - the profit - is maximized. The points with a profit of C dollars lie on the line $X_1 + 6X_2$= C , which has a slope of − 1/6 and is shown in Figure 1, for selected values of C. As C increases, this "profit line" moves parallel to itself, up and to the right. Since the goal is to maximize C, we must move the line as far up as possible, while still touching the feasible region. The optimum solution will be the very last feasible point that the profit line sees and must therefore be a vertex of the polygon, as shown in the figure. If the slope of the profit line were different, then its last contact with the polygon could be an entire edge rather than a single vertex. In that case, the optimum solution would not be unique but given there will be vertices at the end points of the edge which the profit line co-incides with just before leaving the feasible region, those vertices can be considered optimal vertices.

In this example of the "Chocolate Maker" problem shown in class we see that X_1 is 100 and X_2 is 300 and our profit is thus

$$1 \times 100 + 6 \times 300 = 1900$$

2.2 Existence of Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region. The only exceptions are cases in which there is no optimum; this can happen in two ways:

(1) The linear program is infeasible; that is, the constraints are so tight that it is impossible to satisfy all of them. For instance,

$$x \leq 1, x \geq 2.$$

(2) The constraints are so loose that the feasible region is unbounded, and it is possible to achieve arbitrarily high or low objective values. For instance,

$$\begin{aligned} \text{Objective Function:} \quad & x1 + x2 \\ & x1, x2 \geq 0 \end{aligned}$$

In this case we could arbitrarily increase $x1$ and $x2$ to get higher values for the objective function.

3 Simplex Method for solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947. This algorithm starts at a vertex, which could be the origin too, and repeatedly looks for an adjacent vertex (connected by an edge of the feasible region) of better objective value. In this way it does hill-climbing on the vertices of the polygon, walking from neighbor to neighbor so as to steadily increase the objective function value along the way. Upon reaching a vertex that has no better neighbor, simplex declares it to be optimal and halts.

The algorithmm can be described as:

- Start from any vertex v of the feasible set.
- While there is a neighbor v' of v with better objective value, then go to the vertex v' and repeat.
- Otherwise, output vertex v.

Here is a possible trajectory of the Simplex method for our "Chocolate Maker" problem, trying to maximize the profit:

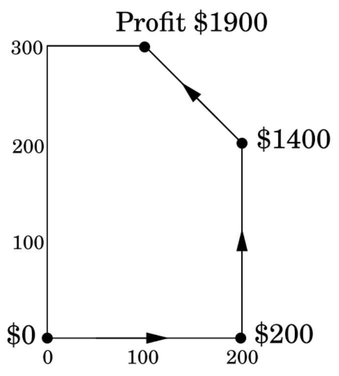


Figure 2: A sample execution of the Simplex method on the Chocolate Maker problem.

We might ask why does this local test imply global optimality?

The answer is given by using concepts of simple geometry. Let us think about the profit line passing through the vertex that has been declared optimal by the algorithm. Since all the vertex′s neighbors lie below the line, the rest of the feasible polygon must also lie below this line as the polygon depicting the feasible region is a convex polygon and for a vertex to lie above the optimal point declared by the Simplex method, the polygon has to be concave as we are using only linear continuos functions to describe the constraints.

3.1 "Bandwidth Allocation" Example

Next we looked at a miniature version of the problem of allocating the optimum bandwidth for a network, often faced by network service providers.

Suppose we are managing a network whose lines have the bandwidths shown in Figure 3. and we need to establish three connections: between users A and B , between B and C , and between A and C . Each connection requires at least two units of bandwidth, but can be assigned more. Connection A-B pays $3 per unit of bandwidth, and connections B-C and A-C pay $2 and $4, respectively.

Connection	Pay in $ / unit
A–B	3
B–C	2
A–C	4

Each connection can be routed in two ways, a long path and a short path, or by a combination: for instance, two units of bandwidth via the short route, one via the long route.

Example routes between A and B :

Long Path (x'_{AB}): $A \to a \to c \to b \to B$
Short Path (x_{AB}): $A \to a \to b \to B$

So the amount of bandwidth between A and B is $x'_{AB} + x_{AB}$

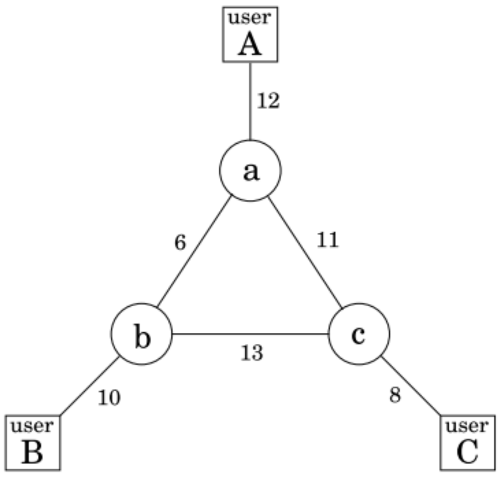


Figure 3: A communications network between three users A, B , and C . Bandwidths are shown.

Now the main question is how do we route these connections to maximize our network′s revenue?

This is a linear program. We have variables for each connection and each path (long or short); for example, x_{AB} is the short-path bandwidth allocated to the connection between A and B , and x'_{AB} the long-path bandwidth for this same connection. We demand that no edge's bandwidth is exceeded and that each connection gets a bandwidth of at least 2 units.

Objective function:

The revenue can be expressed as :

$$3x_{AB} + 3x'_{AB} + 2x_{BC} + 2x'_{BC} + 4x_{AC} + 4x'_{AC}$$

And we want to maximize our objective functions.

Constraints:

The constraints can be listed as:

$$
\begin{aligned}
&x_{AB} + x'_{AB} + x_{BC} + x'_{BC} \leq 10 && [edge(b, B)]\\
&x_{AB} + x'_{AB} + x_{AC} + x'_{AC} \leq 12 && [edge(a, A)]\\
&x_{BC} + x'_{BC} + x_{AC} + x'_{AC} \leq 8 && [edge(c, C)]\\
&x_{AB} + x'_{BC} + x'_{AC} \leq 6 && [edge(a, b)]\\
&x'_{AB} + x_{BC} + x'_{AC} \leq 13 && [edge(b, c)]\\
&x'_{AB} + x'_{BC} + x_{AC} \leq 11 && [edge(a, c)]\\
&x_{AB} + x'_{AB} \geq 2\\
&x_{BC} + x'_{BC} \geq 2\\
&x_{AC} + x'_{AC} \geq 2\\
&x_{AB}, x'_{AB}, x_{BC}, x'_{BC}, x_{AC}, x'_{AC} \geq 0
\end{aligned}
$$

Using the Simplex method we will be able to solve this problem and the solution that we will get is:

$$
\begin{aligned}
&x_{AB} = 0\\
&x'_{AB} = 7\\
&x_{BC} = x'_{BC} = 1.5\\
&x_{AC} = 0.5\\
&x'_{AC} = 4.5
\end{aligned}
$$

4 Network Flow

The next example we looked at was that of "Network Flow".

The networks we generally deal with, when considering the problem of "Network Flow" consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, which are, respectively, a source and sink of G ; and capacities $c_e > 0$ on the edges.

Generally the network is used to model traffic in a road system, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

For example such a network might be modeling a network for shipping oil from a source s to destination t.

A particular shipping scheme is called a flow and consists of a variable f_e for each edge e of the network, satisfying the following two properties:

1. It doesn't violate edge capacities:

$$0 \leq f_e \leq c_e \tag{1}$$

   for all e in E .

2. For all nodes u except s and t , the amount of flow entering u equals the amount leaving u :

$$\sum_{(w,u)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz} \tag{2}$$

   In other words, flow is conserved.

The size of a flow is the total quantity sent from s to t and, by the conservation principle, is equal to the quantity leaving s :

$$size(f) = \sum_{(s,u)\in E} f_{su} \tag{3}$$

An example flow network is shown below.

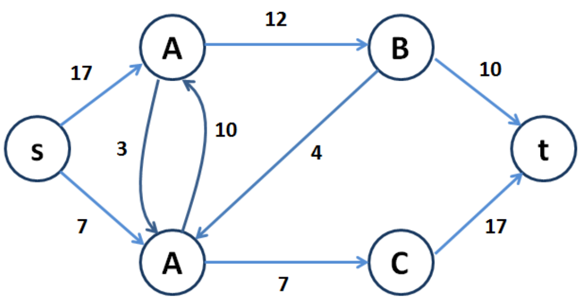


Figure 4: An example flow network.

In the maximizing flow problem, we would like to maximize the flow through the network that is to assign values to $\{f_e : e \in E\}$ that will satisfy a set of linear constraints and maximize a linear objective function.

This maximization in case of the oil shipping scenario would translate to the problem of shipping maximum amount of oil from the source s to the sink t . Each pipeline has a maximum capacity it can handle, and there are no opportunities for storing oil en route. In class a short comparison was drawn with the problem of plumbing, where the pipes have specific capacities and no leaks.

We can view this as a Linear Programming problem, where the objective function we want to maximize is the net flow from the sink, which is just the amount of flow leaving the sink s given in equation (3). If we allow flow to enter the sink then we would have to modify equation (3) to:

$$size(f) = \sum_{(s,u)\in E} f_{su} - \sum_{(v,s)\in E} f_{vs} \tag{4}$$

We also have the constraints that we need:

- For each edge we have two constraint given by the capacity constraint. One is that the flow in the edge cannot be less than zero and the other is that it cannot exceed the capacity as shown in equation (1).
- For each vertex there is a constraint due to the conservation of flow which we can formulate using equation(2).

Using the Simplex method we can maximize the flow, but the linear programs arising from network flows have more structure than general linear programs and we can take advantage of that structure to design algorithms for directly maximizing flow, for example the Max-Flow algorithm by Ford–Fulkerson (1962) and in class this algorithm was described.

5 Max flow via path augmentation by Ford-Fulkerson (1962)

It should be easier to understand a direct method for solving the Max-flow problem if we derive it by observing the workings of the Simplex method while solving the Max-flow problem.

We can first assume that the Simplex method works in the following manner:

- Start with zero flow.
- Repeat : choose an appropriate path from s to t , and increase flow along the edges of this path as much as possible.

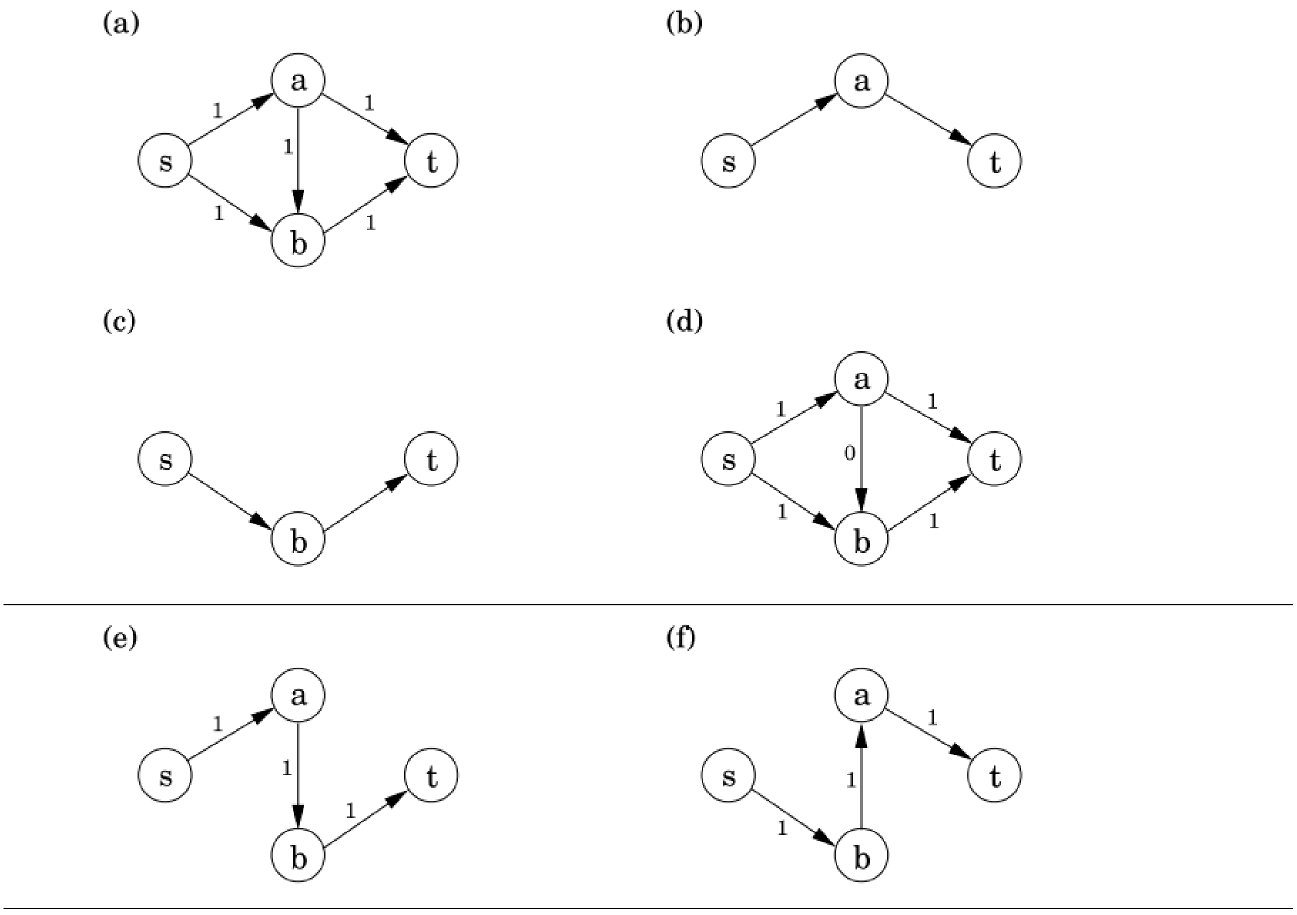


Figure 5: An illustration of the max-flow algorithm. (a) A toy network. (b) The first path chosen. (c) The second path chosen. (d) The final flow. (e) We could have chosen this path first. (f) In which case, we would have to allow this second path.

Figure 5(a)–(d) shows a small example in which simplex halts after two iterations. Here the algorithm chooses two augmenting paths, $s \to a \to t, s \to b \to t$ in two iterations, and it finds the maximum flow of size 2 which is optimal.
There is just one complication. What if we had initially chosen a different path, the one in Figure 5(e)? This gives only one unit of flow and yet seems to block all other paths. Simplex gets around this problem by also allowing paths to *cancel existing flow.* In this particular case, it would subsequently choose the path of Figure 5(f). Edge (b, a) of this path is not in the original network and has the effect of canceling flow previously assigned to edge (a, b) .

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

(1) (u, v) is in the original network, and is not yet at full capacity.

(2) The reverse edge (v, u) is in the original network, and there is some flow along it.

If the current flow is f , then in the first case, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow, and in the second case, upto f_{vu} additional units (canceling all or part of the existing flow on (v, u)).

Therefore, we introduce the notion of residual network. A residual network $G^f = (V, E^f)$ is based on the original flow network: $G = (V, E)$, with new edges added. The residual network has the same vertices as the network G, but has edges with capacities given by the formulae:

$$\begin{cases} c_{uv} - f_{uv} \quad if(u,v) \in E \quad and \quad f_{uv} < c_{uv} \\ f_{vu} \quad if(v,u) \in E \quad and \quad f_{vu} > 0 \end{cases}$$

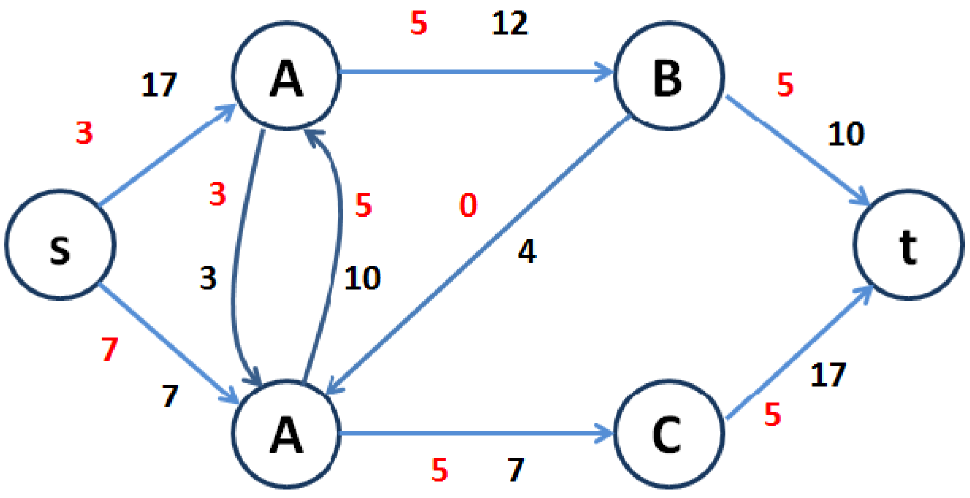
In the first case if the edge (u, v) is in the Flow network and the flow through it is less than it's full capacity, then the residual network has an edge (u, v) which is weighted by the amount of additional flow the edge (u, v) in the Flow network can handle , which is equal to $c_{uv} - f_{uv}$. In the second case, we have an edge (u, v) with capacity f_{vu} in the residual network, if there is the reverse edge (v, u) with capacity f_{vu} is in the Flow network. This capacity is the amount of flow that can be cancelled along the edge (v, u) in the Flow network.

An augmenting path is a path $(u_1, u_2, \ldots, u_k)$ in the residual network, where $u_1 = s$, $u_k = t$, and $c^f(u_i, u_{i+1}) > 0$. 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. To be more precise, if we have an augmenting path $(s, u_1, u_2, ...u_{k-1}, t)$, the maximum flow we can push along that path is the minimum of the capacities of the edges in the augmenting path. Therefore, for a given network G and flow f, if there exists an augmenting path in G^f , then the flow f is not a maximum flow and a network is at maximum flow if and only if there is no augmenting path in the residual network.

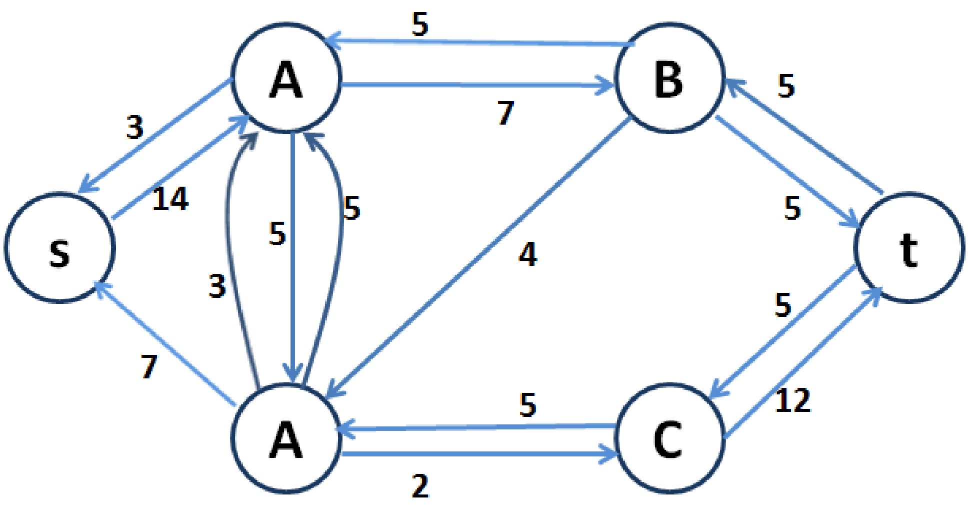
Figure 6. shows a flow graph where flows are assigned to edges and its corresponding residual graph.

Now the Ford-Fulkerson algorithm can proceed as follows:

- Start with zero flow.
- Repeat : choose an appropriate path from s to t from the residual network, and increase flow along the edges of this path as much as possible.



(a) Flows through edges shown in red and Capacities shown in black.



(b) The corresponding residual graph/flow network.

Figure 6: A Flow Network with its corresponding Residual Flow Network

References

[1] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani , "Algorithms," McGraw-Hill, 1st edition, 2006.

[2] Thomas S. Ferguson , "Linear programming: A concise introduction,"