CPSC 500
Nov. 4, 2013

Scribe: Kevin Woo

1 Overview

This lecture covered constrained linear optimization and network flow maximization. Linear programming allows for describing and solving a linear objective function for a maximum or minimum value under a convex and linearly constrained input space. This has uses in budget and resource calculations in operational research. Network flow describes resource flow and distribution over a network of nodes. This has uses for describing distributions in electrical, plumbing, ecological, informatics, transportation and even thermodynamics systems, Network flow can be mapped to a linear programming and thus can be optimized using the same techniques, but the described Ford—Fulkerson algorithm can improve the speed.

2 Linear Programming

2.1 Problem Formulation

For a linear function $c_1x_1 + c_2x_x \cdots + c_nx_n$ and finite set of linear constraints $X_1, X_2, \cdots X_k$ involving $x_1, x_2, \cdots x_n$, the problem of finding nonnegative values for $x_1, x_2, \cdots x_n$ such that $c_1x_1 + c_2x_x \cdots + c_nx_n$ is maximal while all $X_1, X_2, \cdots X_k$ remain true is known as a Linear Programming (LP) problem. This problem can be expressed in standard form as follows:

$$
\begin{array}{lll}
\text{maximize} & c_1x_1 + c_2x_x \cdots + c_nx_n & \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} \text{Linear objective function} \\
\text{subject to} & & \\
& \left.\begin{array}{rl} a_{11}x_1 + a_{12}x_x \cdots + a_{1n}x_n & \le b_1 \\ a_{21}x_1 + a_{22}x_x \cdots + a_{2n}x_n & \le b_2 \\ \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_x \cdots + a_{mn}x_n & \le b_m \end{array}\right\} & \text{Linear inequality constraints} \\
& \left.\begin{array}{rl} x_1 & \ge 0 \\ x_2 & \ge 0 \\ \vdots & \vdots \\ x_n & \ge 0 \end{array}\right\} & \text{Non-negativity constraint}
\end{array}
$$

Note that searching for the minimum of an objective function is the same as searching for the maximum of its negative, and these linear inequality constraints are equivalent:

$$a_{i1}x_1 + a_{i2}x_x \cdots + a_{in}x_n \ge b_i \quad \Leftrightarrow \quad -a_{i1}x_1 - a_{i2}x_x \cdots - a_{in}x_n \le -b_i$$

$$a_{i1}x_1 + a_{i2}x_x \cdots + a_{in}x_n = b_i \quad \Leftrightarrow \quad \left(-a_{i1}x_1 - a_{i2}x_x \cdots - a_{in}x_n \le -b_i\right) \& \left(a_{i1}x_1 + a_{i2}x_x \cdots + a_{in}x_n \le b_i\right)$$

For an LP problem, the values for $x_1, x_2, \cdots x_n$ that satisfy the constraints $X_1, X_2, \cdots X_k$ form a convex[1] bounded region of feasible solutions. The optimal solution is feasible assignment of values $x_1, x_2, \cdots x_n$ that maximizes the objective function, and should occur at a vertex of the region. However, note that this does not preclude optimal solutions occurring at other vertices. This can be intuitively understood by considering straight contour lines of increasing value sweeping across the feasibility region. The line reaches a maximal value either at a vertex, in which case there is a unique optimal solution and value, or at an edge, in which case while there is a unique optimal value, any point along that edge is also a valid optimal solution.

2.2 Example

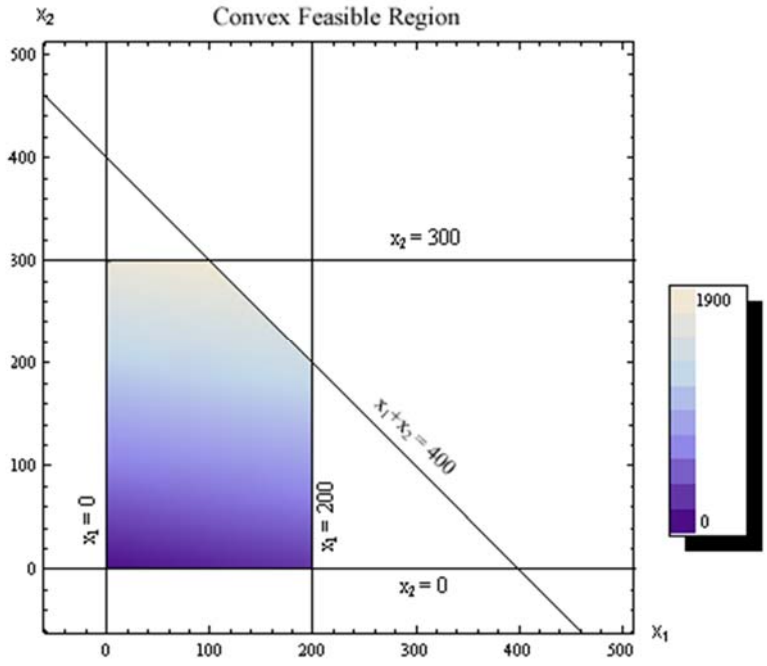
A chocolate shop produces and sells two boxes of chocolate, Box 1 and Box 2. Box 1 sells for $1 but sells no more than 200 boxes per day. Box 2 sells for $6 but sells no more than $300 boxes per day. The shop produces a maximum total of 400 boxes per day. How many boxes should be produced to maximize profit?

This problem can be formalized as an LP problem. Let x_1 and x_2 be the daily production of Box 1 and Box 2, respectively. The profit or the objective function to maximize is therefore 1 x_1+6 x_2. Box 1 ranges between $0 \le x_1 \le 200$, Box 2 ranges between $0 \le x_2 \le 300$, and the total production is limited to $x_1 + x_2 \le 400$. Therefore, the problem can be formulated in standard form as follows:

$$\begin{aligned} \text{maximize} \quad & x_1 + 6x_2 \\ \text{subject to} \quad & x_1 \le 200 \\ & x_2 \le 300 \\ & x_1 + x_2 \le 400 \\ & x_1, x_2 \ge 0 \end{aligned}$$

[1] A polytope P is convex if for each pair of points $x, y \in P$, $0 \le \lambda \le 1 \Rightarrow \lambda x + (1-\lambda) y \in P$

The constraints correspond to the following feasibility region to optimize the objective function:



2.3 Simplex Algorithm [1]

The Simplex algorithm described by Dantzig in 1947 iteratively solves an LP problem by traversing between neighboring vertices of the convex feasible region that increase the value of the objective function.

Introduce slack variables $s_1, s_2 \cdots s_k$ into the constraints of the form $X_i \le a_i$ such that $s_i = a_i - X_i$. In the chocolate shop example above, this corresponds to a rewrite of the LP problem as the equivalent:

$$\begin{aligned} \text{maximize} \quad & z = x_1 + 6x_2 \\ \text{subject to} \quad & s_1 = 200 - x_1 \\ & s_2 = 300 - x_2 \\ & s_3 = 400 - x_1 - x_2 \\ & x_1, x_2, s_1, s_2, s_3 \ge 0 \end{aligned}$$

Note that every feasible solution found for this formulation is also a feasible solution of the original.

A dictionary is used for bookkeeping to keep track of the variables x_1, x_2, s_1, s_2, s_3 and objective value *z*:

$$\begin{aligned} s_1 &= 200 - x_1 \\ s_2 &= 300 - x_2 \\ s_3 &= 400 - x_1 - x_2 \\ z &= x_1 + 6x_2 \end{aligned}$$

Simplex traverses between vertices of feasible region, the above dictionary serves as an initialization. Each dictionary constructed during Simplex algorithm corresponds to a feasible solution to the LP problem by setting all right-hand-side (RHS) variables to zero. The initial dictionary therefore has a feasible solution:

$$\begin{aligned} s_1 &= 200 \\ s_2 &= 300 \\ s_3 &= 400 \\ x_1 &= 0 \\ x_2 &= 0 \\ z &= 0 \end{aligned}$$

Simplex will now iterate through these steps and update the dictionary until *z* no further improves:

1. In the objective function $z = \cdots$, select a variable *y* with a positive coefficient.
2. While keeping all other RHS variables as zero, increase the value in *y* furthest while satisfying all variable non-negativity constraints.
3. Rearrange the dictionary to bring *y* to the LHS as a new non-zero *basis* **using the constraint that was the limiter**. At this point, setting all RHS non-basis variables to zero reveals a new *basic feasible solution* to the LP problem.
4. Repeat until $z = \cdots$ has no variable with a positive coefficient (and thus has no variable through which increasing its value will continue to increase the objective value *z*).

Assuming no cycles, the basic feasible solution at the end of this process will be the optimal solution.

Stepping through the algorithm, with the previous dictionary it can be seen that the objective function $z = x_1 + 6x_2$ can be improved by increasing either x_1 or x_2 . Arbitrarily choosing x_1 for this step, and keeping $x_2 = 0$, the constraints show:

$$\begin{aligned} s_1 &= 200 - x_1 && \Rightarrow x_1 \le 200 \\ s_2 &= 300 - x_2 && \Rightarrow x_1 \text{ unbounded} \\ s_3 &= 400 - x_1 - x_2 && \Rightarrow x_1 \le 400 \end{aligned}$$

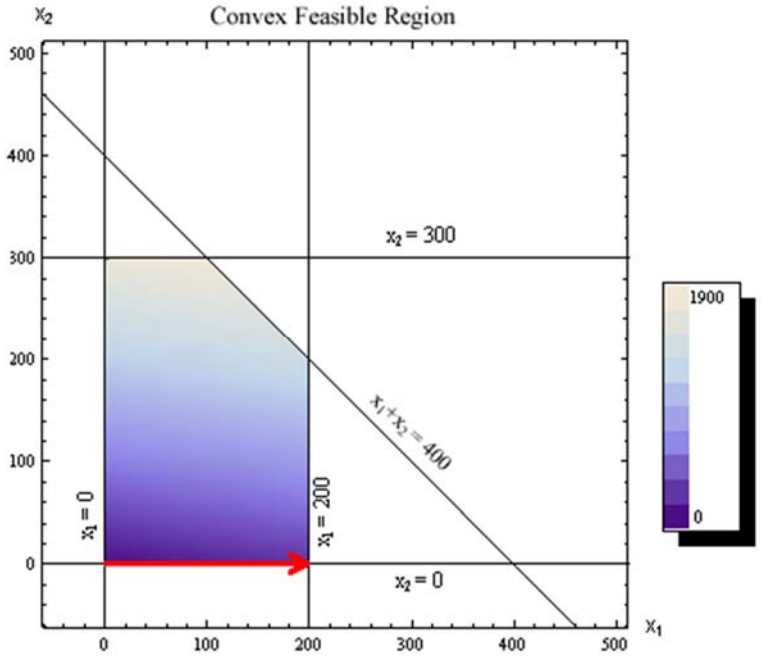
Therefore, for this step x_1 can be pushed to most 200 while satisfying all constraints. Now, rearranging the dictionary as per (3):

$$\begin{aligned} x_1 &= 200 - s_1 \\ s_2 &= 300 - x_2 \\ s_3 &= 200 + s_1 - x_2 \\ z &= 200 - s_1 + 6x_2 \end{aligned}$$

Notice that now a basic feasible solution from this dictionary is as expected:

$$
\begin{aligned}
x_1 &= 200 \\
x_2 &= 0 \\
s_1 &= 0 \\
s_2 &= 300 \\
s_3 &= 200 \\
z &= 200
\end{aligned}
$$

This corresponds the move between these vertices:



Clearly, the current vertex is not yet optimal, and this can be seen from the updated objective $z = 200 - s_1 + 6x_2$. From both this equation as well as the diagram above, x_2 can still increase. Therefore, in this next iteration, x_2 is chosen to *pivot*. Keeping $s_1 = 0$, the constraints show:

$$
\begin{aligned}
x_1 &= 200 - s_1 && \Rightarrow x_2 \text{ unbounded} \\
s_2 &= 300 - x_2 && \Rightarrow x_2 \le 300 \\
s_3 &= 200 + s_1 - x_2 && \Rightarrow x_2 \le 200
\end{aligned}
$$

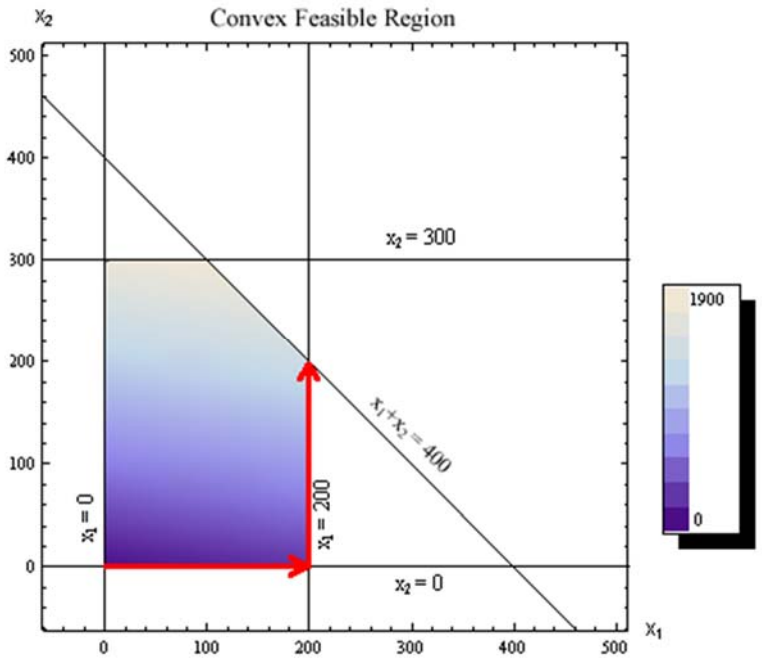
Therefore, for this step x_2 can be pushed to most 200 while satisfying all constraints. Now, rearranging the dictionary as per (3):

$$
\begin{aligned}
x_1 &= 200 - s_1 \\
s_2 &= 100 - s_1 + s_3 \\
x_2 &= 200 + s_1 - s_3 \\
z &= 1400 + 5s_1 - 6s_3
\end{aligned}
$$

The basic feasible solution from this dictionary is now:

$$\begin{aligned} x_1 &= 200 \\ x_2 &= 200 \\ s_1 &= 0 \\ s_2 &= 100 \\ s_3 &= 0 \\ z &= 1400 \end{aligned}$$

This corresponds another vertex move:



Lastly, the objective $z = 1400 + 5s_1 - 6s_3$ shows that s_1 can still increase. Choosing s_1 and keeping $s_3 = 0$, the constraints show:

$$\begin{aligned} x_1 &= 200 - s_1 && \Rightarrow s_1 \le 200 \\ s_2 &= 100 - s_1 + s_3 && \Rightarrow s_1 \le 100 \\ x_2 &= 200 + s_1 - s_3 && \Rightarrow s_1 \text{ unbounded} \end{aligned}$$

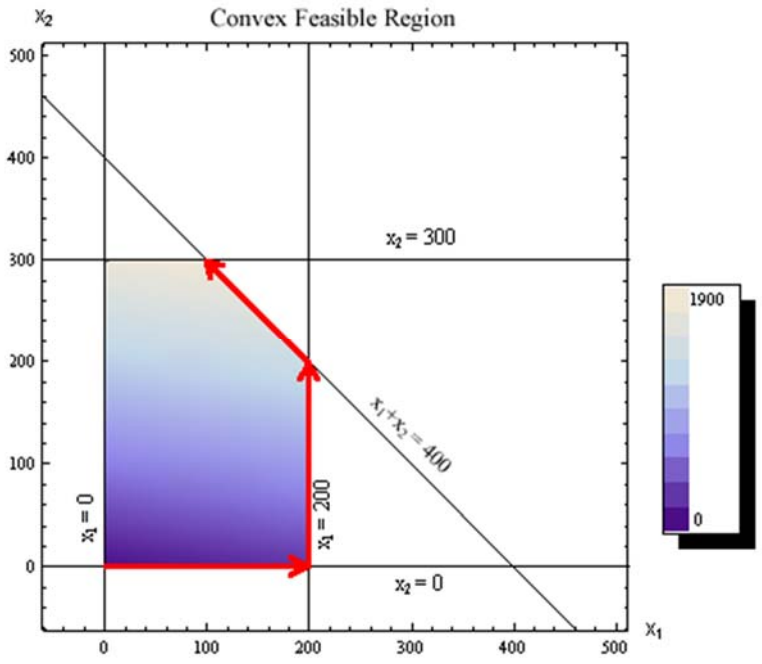
Therefore, for this step s_1 can be pushed to most 100 while satisfying all constraints. Now, rearranging the dictionary as per (3):

$$\begin{aligned} x_1 &= 100 + s_2 - s_3 \\ s_1 &= 100 - s_2 + s_3 \\ x_2 &= 300 - s_2 \\ z &= 1900 - 5s_2 - s_3 \end{aligned}$$

The basic feasible solution from this dictionary is now:

$$x_1 = 100$$
$$x_2 = 300$$
$$s_1 = 100$$
$$s_2 = 0$$
$$s_3 = 0$$
$$z = 1900$$

This corresponds to the final vertex move:



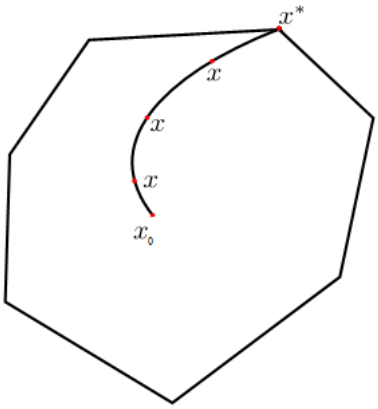
The current vertex is finally optimal, and this can be seen from the updated objective $z = 1900 - 5s_2 - s_3$ as well as the diagram above. Any further increase in variables s_2 or s_3 will only decrease the objective value. Therefore, the algorithm stops and returns the optimal solution: $x_1 = 100,\ x_2 = 300$.

2.4 Side Note: Interior Point Methods [2]

While the Simplex algorithm can find the optimal solution, its necessary traversal between vertices may resulting choosing a long path to reach the optimal vertex and require a dictionary update for every vertex along the way, even if the vertex does not offer much improvement. In the worst case, this could result in traversing through **every** vertex. Another strategy for optimizing under constraints is called interior point methods, or *barrier methods*, which traverse within the convex region and steer directly towards to optimal vertex. Non-negativity constraints become part of the objective function by incurring heavy penalty when nearing zero, which can be accomplished by a negative logarithm of the variable, thus:

$$z = c_1 x_1 + \cdots + c_k x_k - \mu \sum_i \log x_i \quad \text{for a particular choice of } \mu$$

Equality constraints can be optimized via Newton's method by noting that the primal and dual solutions to the LP problem are simultaneously satisfied at the optimal solution, and therefore the gap between the primal and dual objective functions becomes zero and thus can be iteratively minimized.



Newton's method iteration for the barrier method turns to approach the optimal solution. [2]

3 Flow Network

3.1 Problem Formulation

A flow network is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a positive capacity $c(u, v)$. The capacity is zero if there is no edge between two nodes. There is a source vertex $s \in V$ and a sink vertex $t \in V$. A flow is an assignment of real numbers to edges of G that satisfy the following constraints.

For every edge $(u, v) \in E$:

1. Capacity Constraint: The flow through an edge must not exceed its capacity.
   $$0 \le f(u,v) \le c(u,v)$$

2. Flow Conservation: Total incoming flow to a vertex $v \in V - \{s,t\}$ equals its total outgoing flow.
   $$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w)$$

The size of the flow within a particular network from s to t is:

$$size(f) = \sum_{(s,v)\in E} f(s,v) - \underbrace{\sum_{(v,s)\in E} f(v,s)}_{\text{Flow back to source}}$$

Note that the problem of maximizing the size of the flow through a network can be mapped to an LP problem since the objective function and constraints are all linear. The capacity constraints can be described with non-negativity constraints and, since the capacity is a fixed constant for each edge, the constraints of the form $X_i \le b_i$. The conservation constraints can be expanded to apply for each vertex using equality constraints. Recall that equality constraints can be expanded into two inequalities constraints.

3.2 Ford–Fulkerson Algorithm

While the Simplex algorithm can indeed solve for the maximum flow, it is possible to further exploit the structure of the flow network more efficiently. Ford and Fulkerson described such an algorithm in 1962:

1. Start with a zero-flow: f

2. Form a residual graph $G_f = (V, E_f)$ A residual graph is one containing edges between nodes (u, v) that can still admit flow; that is, $E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$ where $c_f(u, v) = c(u, v) - f(u, v)$ is the residual capacity. [3]

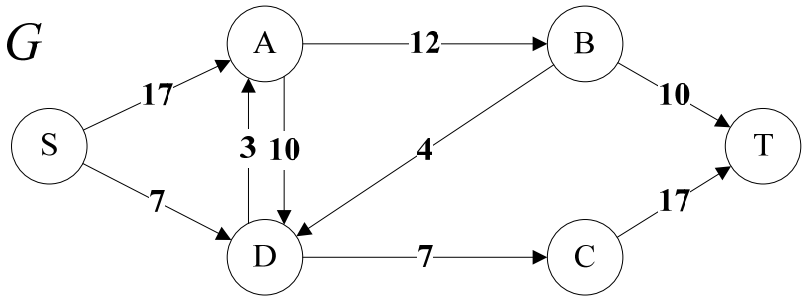
3. Find a directed path in G_f from s to t. Typically, the search for this path is done breadth-first as described by the Edmonds–Karp variant, which lowers the runtime to polynomial $O(|V||E|^2)$ from the original $O(|E| f^*)$, where f^* is the maximum flow of the graph [4].

4. Augment f by increasing flow along all edges of the path in G_f by the capacity of the path, which is determined by the lowest capacity edge.

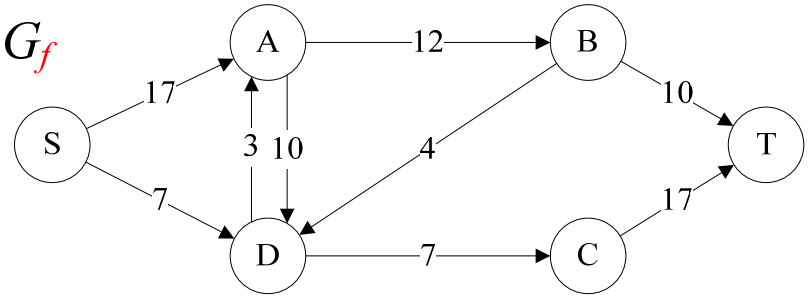
5. Repeat until there are no more paths in G_f from s to t in the residual graph.

3.3 Example

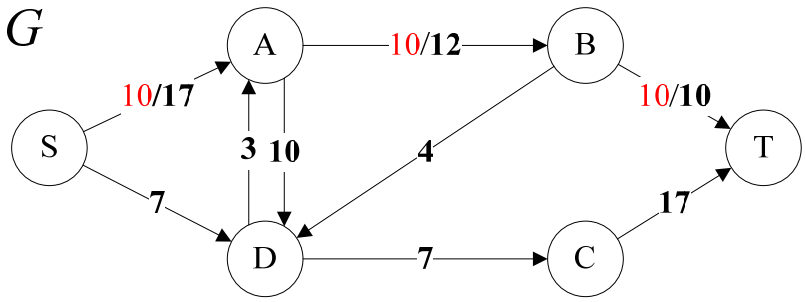
Consider a network with the following capacities:



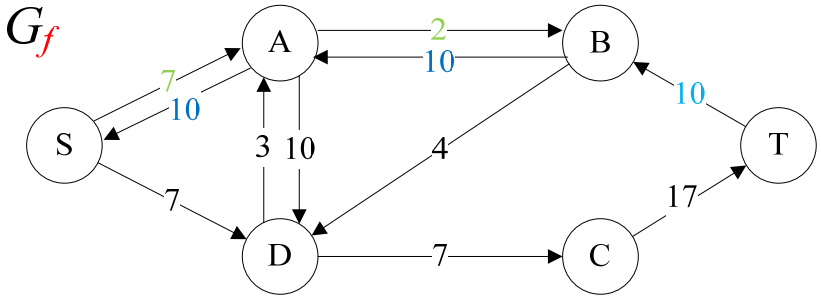
Initially, there is no flow and therefore the residual graph G_f is the same as G:



Suppose in the next iteration, the path S—A—B—T is found by breadth-first search. The limiting path capacity here is +10 due to edge B—T. Therefore, this additional amount of flow is pushed along the path:



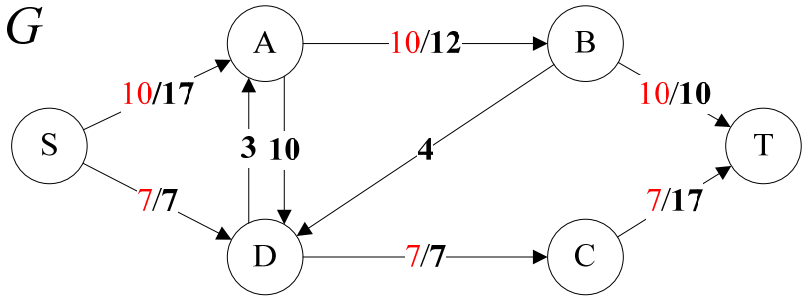
The corresponding residual graph G_f at this point is therefore:



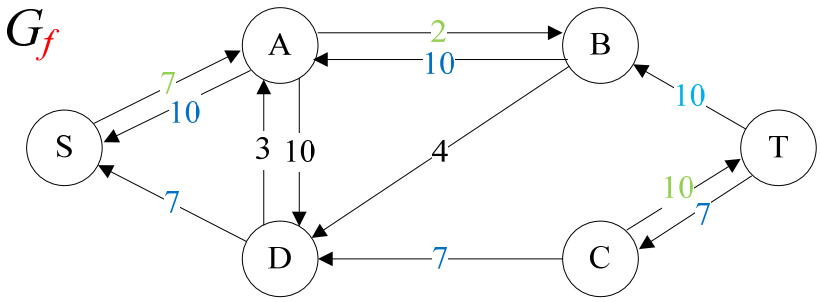
Notice that the backflow in the residual graph G_f is equal to the flow in G, and that the updated forward flow in the residual graph is the remaining flow capacity available given difference between the flow and original **capacity**. If said difference is zero, then that empty residual capacity is equivalent to the *omission* of that edge. Conversely, the edges that have no current flow remain unchanged with all available capacity.

With the updated residual graph, the breadth-first search in next iteration will find the path S—D—C—T. Note that since BFS chooses the shortest depth to explore, the paths S—A—D—C—T and S—A—B—D—C—T are currently skipped.

The path S—D—C—T has the limiting capacity +7 due to either of edges S—D or D—C. Therefore, this additional amount of flow is pushed along the path:



The corresponding residual graph G_f at this point is therefore:



There is now no longer a path from s to t in the residual graph G_f, and therefore the algorithm terminates with the current f as the maximum flow for the network.

4 Summary

This lecture introduced linear programming, network flows, and the mapping relations between the two while providing basic algorithms that have at least average polynomial-time running time complexity. The upcoming lectures should provide more reasoning into the correctness of the Ford—Fulkerson algorithm and along with its usefulness.

Sources

[1] UBC MATH 340 Linear Programming: Course lectures and notes.

[2] UBC CPSC 406 Computational Optimization: Course lectures and notes.

[3] Wikipedia, "Flow Network," [Online]. Available: http://en.wikipedia.org/wiki/Flow_network.

[4] Wikipedia, "Ford–Fulkerson algorithm," [Online]. Available: http://en.wikipedia.org/wiki/Ford–Fulkerson algorithm.