CS420+500: Advanced Algorithm Design and Analysis

Lectures: Feb 8 + Feb 10, 2017

Prof. Will Evans *Scribe: Arthur de Sousa Marques*

In these lectures we discussed applications of network flow. More specifically:

- The pennant race problem
- The mining open pit problem

1 The pennant race problem

Problem We are fanatics for a team A, that is in a certain position in the baseball tournament. Has my team any chance of winning the baseball tournament, *i.e.* is it possible to determine if team A can win at least as many games as any other team by the end of the season?

Input *(i)* win and losses for each team, and *(ii)* list of games remaining to be played.

Output YES or NO output indicating if team A has hope or not, that is, team A is mathematically eliminated?

Algorithm 1: Pennant race network flow

Input: W list of wins, G list of remaining games
Output: YES or NO if team A has chance to win the tournament

Let A win all its remaining games;
Let w be the number of wins for team A (assuming step 1);
Let w_i be the number of wins for a team T_i (assuming step 1);
if $w < w_i$ for some i, **return NO**;
else solve the problem using **network flow** ;
    Let $L = (T_iT_j)$ be the set of games (*i.e.* pairs of teams) remaining to be played (assuming step 1) ;
    Create a *source* vertex ;
    Create a *target* vertex ;
    For all $T_iT_j \in L$, create edges (s, T_iT_j) with capacity 1 ;
    For all $T_iT_j \in L$, create edges (T_iT_j, T_i), and (T_iT_j, T_j) with capacity ∞ ;
    For all $T_i \in L$, create edges (T_i, t), with capacity $w - w_i$;
    Compute **max-flow** f in the graph created ;
    Check if **max-flow** $f = size(L)$;
    if $f = size(L)$, **return YES** ;
    else, **return NO**;

1.1 Example

We want to determine if team A has chance to win the tournament.

team	#wins
A	3
T_1	4
T_2	6
T_3	5
T_4	4

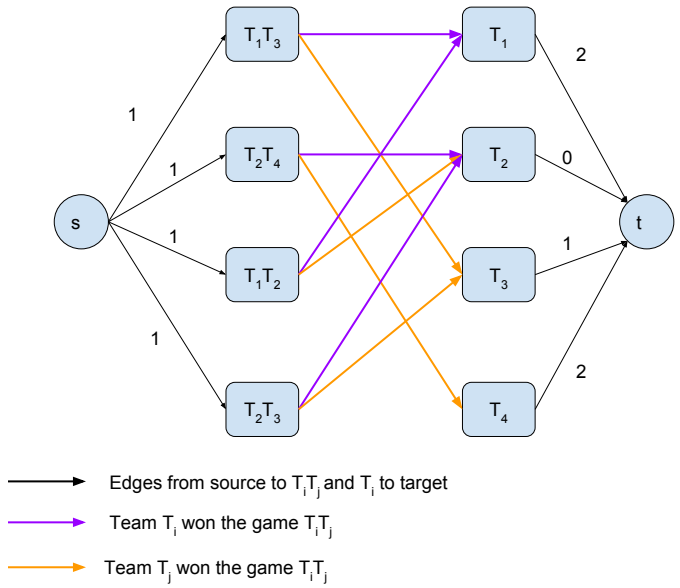
We know that the list of remaining games is:

$L = (A, T_1), (A, T_3), (A, T_4), (T_1, T_3), (T_2, T_3), (T_2, T_4), (T_1, T_2)$

Then, we compute w and w_i

team	#wins	$w - w_i$
A	3	w = 6, *i.e.* 3 + winning $(A, T_1), (A, T_3), (A, T_4)$
T_1	4	2
T_2	6	0
T_3	5	1
T_4	4	2

As all w_i satisfy step **4** of the algorithm, we need to create the network flow and compute max-flow f.



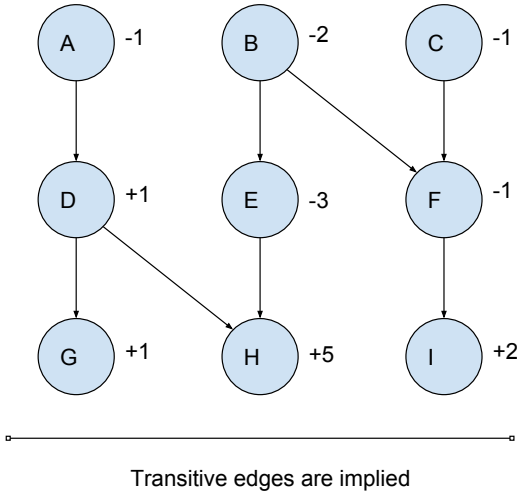
In the above network flow, $f = 4$ and the list of games to be played has size 4. Therefore, YES, team A has hope!

2 The open-pit mining problem

Problem We are mining cubes of dirt. For each cube, there is a profit p (which might be negative) that we can obtain by mining that cube. Additionally, we cannot dig cubes that are "bellow" other cubes. How can we maximize our profit by digging some, or all, of the cubes?

Input Directed acyclic graph $G = (V, E)$, where V is the set of tasks, and $E = \{(u, v)\}|u$ must be done before v. A function $w(v)$ that specifies the profit from doing task v.

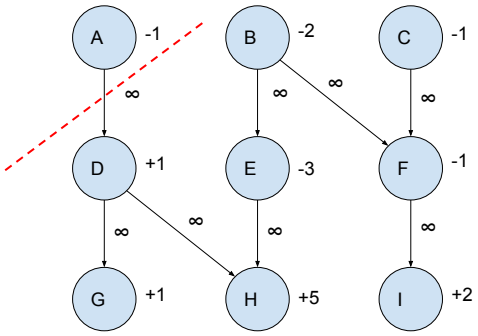
Output The most profitable set of tasks to perform, subject to the tasks' precedence.



A **feasible set** in a set of vertices has no edges coming into it from outside. **(D, G)** is not feasible. **(A, D, G)** is feasible.

Let us convert this problem to a **network flow** problem such that *1)* any finite capacity cut corresponds to an initial set *2)* a minimum capacity cut corresponds to max profit initial set.

In this network, any finite capacity cut (S, T) defines an initial set $T - \{t\}$



Proof: If cut (S, T) has finite capacity then, no original edge is directed into t from s, which implies $T - t$ is the initial set.

If set U is an initial set, then $T = U \cup \{t\}$. $S = V \setminus T$ is a cut with no original edge entering t. So to maximize profit, we want to **minimize the loss**.

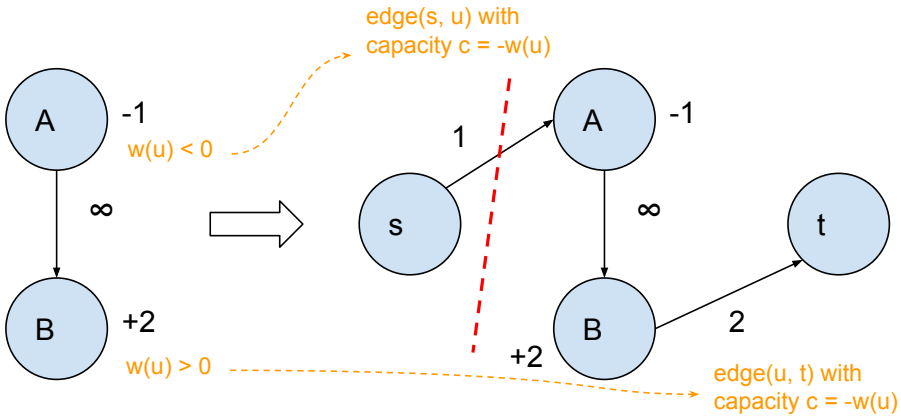
Algorithm 2: Open-pit mining problem network flow

Input: G mine graph, w(u) profit function
Output: Max profit

```
Create a source vertex ;
Create a target vertex ;
Add ∞ capacities on the edges of G ;
for each u ∈ G do ;
    if w(u) > 0 then create an edge (u, t) with capacity c = w(u) ;
    if w(u) < 0 then create an edge (s, u) with capacity c = −w(u) ;
Find the maximum value flow and the associated minimum capacity cut (S, T) ;
All vertices in T (except t) should be dug to get max profit ;
```

2.1 Example

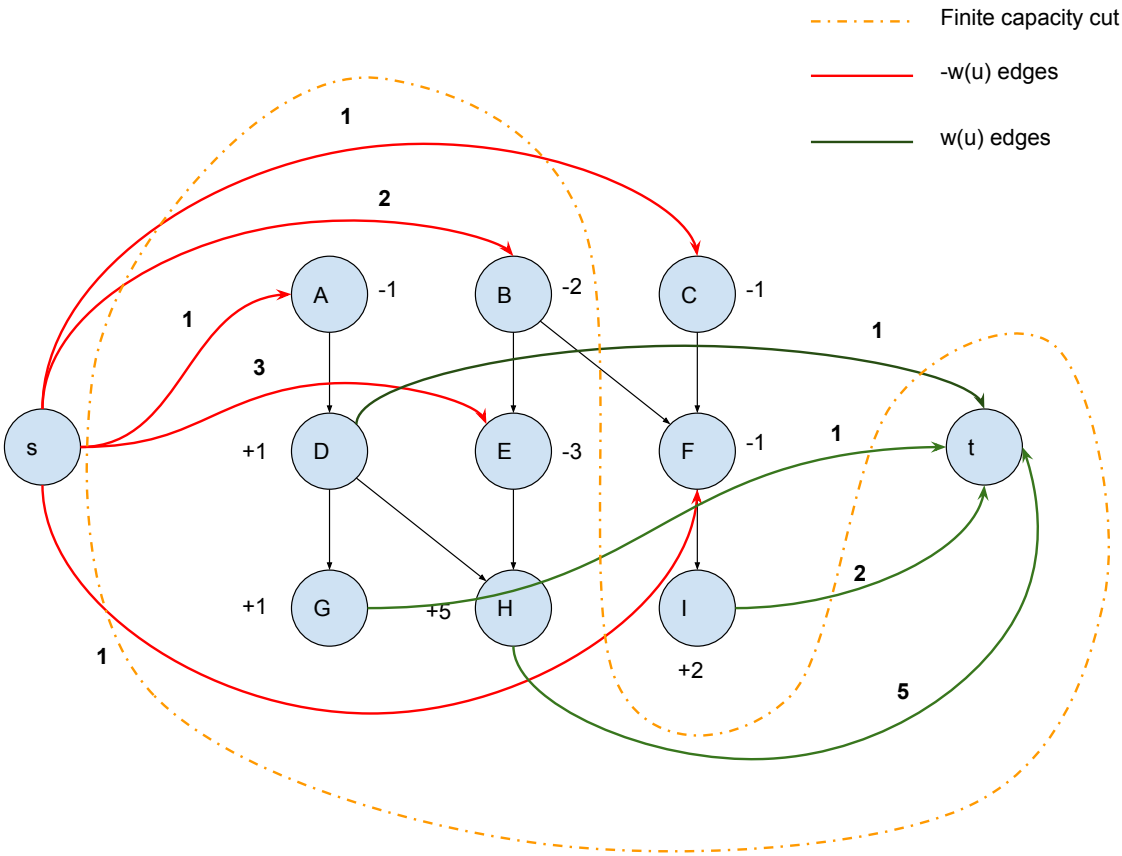
We want to determine the max profit of $G = (\{A, B\}, \{(A, B)\})$. With $w(A) = -1$, and $w(B) = +2$. Clearly, the max profit is achieved by digging $\{A, B\}$. If we use the open-pit algorithm to build this network flow, we get the flow on the right:



In this flow, the min cut occurs on (s, A). Therefore, we get $\{A, B\}$ as our feasible set with max profit.

2.2 Profit

After the presented example, let us run the algorithm on our initial graph (presented in the beginning of Section 2) to check some properties about the tasks' profit. In the output below, $w(u) < 0$ edges are labeled in red, $w(u) > 0$ edges are labeled in green, and finally, the min capacity cut is labeled in dashed orange.



Notice that the **min capacity cut** divides vertices $\{A, B, D, E, G, H\}$ on the t side, and vertices $\{C, F, I\}$ on the s side, such that our profit $p = 1$. One can express the profit formula as:

$$profit_{initial\ set\ U} = \sum_{u \in U | w(u) > 0} w(u) + \sum_{v \in U | w(v) < 0} w(v)$$

Since minimizing the cut maximizes the profit we can state:

$$\sum_{u \in V | w(u) > 0} w(u) - profit$$
is the same as
$$\sum_{u \notin U | w(u) > 0} w(u) - \sum_{v \in U | w(v) < 0} w(v)$$

Where:

$\sum_{u \in V | w(u) > 0} w(u)$ is the sum of all positive profits
$\sum_{u \notin U | w(u) > 0} w(u)$ is the sum of all positive profits not in the initial set U
$\sum_{v \in U | w(v) < 0} w(v)$ is the sum of all negative profits in the initial set U

Considering the presented network flow:

$profit = 1$
$\sum_{u \in V | w(u) > 0} w(u) = 9$
$\sum_{u \notin U | w(u) > 0} w(u) = 2$
$\sum_{v \in U | w(v) < 0} w(v) = -6$

Therefore,

$$9 - 1 = 2 - (-6) \rightarrow 8 = 8$$

This example serves to express that maximizing the profit is the same as minimizing the loss and thus, there is a dual way to express the open-pit mining problem.