Chapter 20

More Network Flow Applications

OLD CS 473: Fundamental Algorithms, Spring 2015
April 4, 2015

20.1 Airline Scheduling

20.1.1 Airline Scheduling

20.1.1.1 Lower bounds

(A) The following example requires the ability to solve network flow with lower bounds on the edges.
(B) This can be reduced to regular network flow (we are not going to show the details – they are a bit tedious).
(C) The integrality property holds – if there is an integral solution our `network flow with lower bounds` solver would compute such a solution.

20.1.1.2 Airline Scheduling

Problem 20.1.1. *Given information about flights that an airline needs to provide, generate a profitable schedule.*

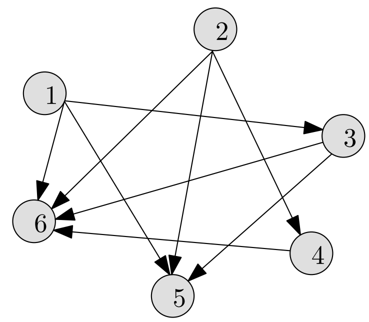
(A) Input: detailed information about "legs" of flight.
(B) $\mathcal{F}$: set of flights by
(C) Purpose: find minimum # airplanes needed.

20.1.2 Example

20.1.2.1 (i) a set $\mathcal{F}$ of flights that have to be served, and (ii) the corresponding graph G representing these flights.

1: Boston (depart 6 A.M.) - Washington DC (arrive 7 A.M,).
2: Urbana (depart 7 A.M.) - Champaign (arrive 8 A.M.)
3: Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)
4: Urbana (depart 11 A.M.) - San Francisco (arrive 2 P.M.)
5: San Francisco (depart 2:15 P.M.) - Seattle (arrive 3:15 P.M.)
6: Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.).

(i)



(ii)

20.1.2.2 Flight scheduling...

(A) Use same airplane for two segments i and j:
  (a) destination of i is the origin of the segment j,
  (b) there is enough time in between the two flights.
(B) Also, airplane can fly from dest(i) to origin(j) (assuming time constraints are satisfied).

Example 20.1.2. *As a concrete example, consider the flights:*

1. *Boston (depart 6 A.M.) - Washington D.C. (arrive 7 A.M,).*
2. *Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)*
3. *Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.)*

This schedule can be served by a single airplane by adding the leg "Los Angeles (depart 12 noon)- Las Vegas (1 P,M.)" to this schedule.

20.1.2.3 Modeling the problem

(A) model the feasibility constraints by a graph.
(B) G: directed graph over flight legs.
(C) For i and j (legs), $(i \to j) \in \mathsf{E}(\mathsf{G}) \iff$ same airplane can serve both i and j.
(D) G is acyclic.
(E) Q: Can required legs can be served using only k airplanes?

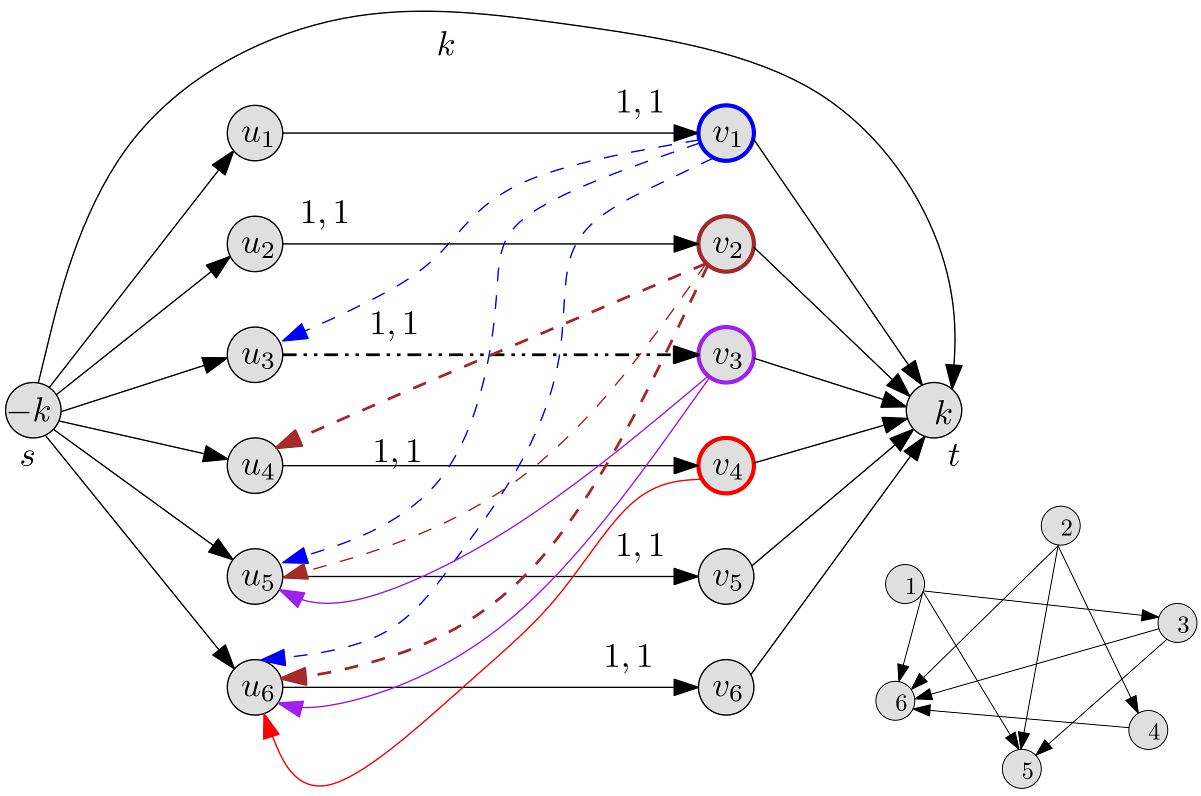
20.1.2.4 Solution

(A) Reduction to computation of circulation.
(B) Build graph H.

(C) $\forall$ leg i, two new vertices $u_i, v_i \in \mathsf{VH}$.
s: source vertex. t: sink vertex.
(D) Set demand at t to k, Demand at s to be $-k$.
(E) Flight must be served: New edge $e_i = (u_i \to v_i)$, for leg i.
Also $\ell(e_i) = 1$ and $c(e_i) = 1$.
(F) If same plane can so i and j (i.e., $(i \to j) \in \mathsf{E}(\mathsf{G})$) then add edge $(v_i \to u_j)$ with capacity 1 to H.
(G) Since any airplane can start the day with flight i: add an edge $(s \to u_i)$ with capacity 1 to H, $\forall i$.
(H) Add edge $(v_j \to t)$ with capacity 1 to G, $\forall j$.
(I) Overflow airplanes: "overflow" edge $(s \to t)$ with capacity k.
Let H denote the resulting graph.

20.1.3 Example of resulting graph

20.1.3.1 The resulting graph H for the instance of airline scheduling show before.



20.1.3.2 Lemma

Lemma 20.1.3. *$\exists$ way perform all flights of $\mathcal{F} \leq k$ planes $\iff$ $\exists$ circulation in H.*

Proof:

(A) Given feasible solution → translate into valid circulation.
(B) Given feasible circulation...
(C) ... extract paths from flow.
(D) ... every path is a plane.

■

20.1.3.3 Extensions and limitations

(A) a lot of other considerations:
    (i) airplanes have to undergo long term maintenance treatments every once in awhile,
    (ii) one needs to allocate crew to these flights,
    (iii) schedule differ between days, and
    (iv) ultimately we interested in maximizing revenue.
(B) Network flow is used in practice, real world problems are complicated, and network flow can capture only a few aspects.
(C) ... a good starting point.

20.1.4 Baseball Pennant Race

20.1.4.1 Pennant Race

TUESDAY, SEPTEMBER 10, 1996

San Francisco Chronicle

The Gate
Sports Online
▶ http://www.sfgate.com

SPORTING G

49ers, Young Get Big Brea

Quarterback m

By Gary Swan
Chronicle Staff Writer

The bye week has come at a perfect time for the 49ers and quarterback Steve Young. If they had a game next Sunday, there's a good chance Young would not play.

But the pulled groin muscle on his up-

Giants Officially Leave the NL West Race

By Nancy Gay
Chronicle Staff Writer

With the smack of another National League West bat 500 miles away, the Giants' run at the division title ended last night, just as they were handing the visiting St. Louis Cardinals an even bigger lead in the NL Central.

CARDINALS 6
GIANTS 2

In San Diego, Greg Vaughn's three-run homer in the eighth pushed the Padres over the Pirates and officially shoved the rest of the Giants' season into the background. On the heels of their tedious 6-2 loss before an announced crowd of 10,307 at Candlestick Park, the Giants fell 19½ games off the lead.

As it is, the worst the Padres (80-65) can finish is 80-82. The Giants have fallen to 59-83 with 20 games left; they cannot win 80 games. Coming off a miserable 2-8 mark on a three-city road trip that saw their road record drop to 27-47, the Giants were hoping to get off on the right foot in their longest homestand of the year (15 games, 14 days).

Financing In Place For Giants' New Stadium
SEE PAGE B1, MAIN NEWS

"Where we are, you're going to be eliminated sooner or later," Baker said quietly. "But it doesn't alter the fact that we've still got to play ball. You've still got to play hard, the fans come out to watch you play. You've got to play for the fact of loving to play, no matter where you are in the standings.

"You've got to play the role of spoiler, to not make it easier on

GIANTS: Page D5 Col. 3

20.1.4.2 Pennant Race: Example

Can Boston win the pennant?
No, because Boston can win at most 91 games.

20.1.4.3 Another Example

Can Boston win the pennant?
Not clear unless we know what the remaining games are!

Example 20.1.4.

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	89	2

Example 20.1.5.

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	90	2

20.1.4.4 Refining the Example

Example 20.1.6.

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	–	1	1	0
Baltimore	91	3	1	–	1	1
Toronto	91	3	1	1	–	1
Boston	90	2	0	1	1	–

Can Boston win the pennant? Suppose Boston does

(A) Boston wins both its games to get 92 wins

(B) New York must lose both games; now both Baltimore and Toronto have at least 92

(C) Winner of Baltimore-Toronto game has 93 wins!

20.1.4.5 Abstracting the Problem

Given

(A) A set of teams S

(B) For each $x \in S$, the current number of wins w_x

(C) For any $x, y \in S$, the number of remaining games g_{xy} between x and y

(D) A team z

Can z win the pennant?

20.1.4.6 Towards a Reduction

$\overline{z}$ can win the pennant if

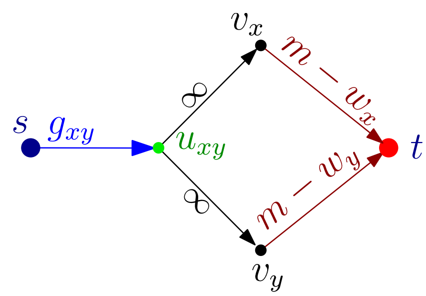
(A) $\overline{z}$ wins at least m games
- (A) to maximize $\overline{z}$'s chances we make $\overline{z}$ win all its remaining games and hence $m = w_{\overline{z}} + \sum_{x \in S} g_{x\overline{z}}$

(B) no other team wins more than m games
- (A) for each $x, y \in S$ the g_{xy} games between them have to be *assigned* to either x or y.
- (B) each team $x \neq \overline{z}$ can win at most $m - w_x - g_{x\overline{z}}$ remaining games

Is there an assignment of remaining games to teams such that no team $x \neq \overline{z}$ wins more than $m - w_x$ games?

20.1.4.7 Flow Network: The basic gadget

(A) s: source
(B) t: sink
(C) x, y: two teams
(D) g_{xy}: number of games remaining between x and y.
(E) w_x: number of points x has.
(F) m: maximum number of points x can win before team of interest is eliminated.

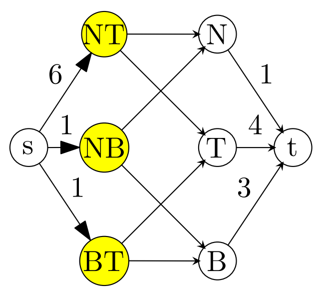


20.1.5 Flow Network: An Example

20.1.5.1 Can Boston win?

Team	Won	Left	NY	Bal	Tor	Bos
New York	90	11	–	1	6	4
Baltimore	88	6	1	–	1	4
Toronto	87	11	6	1	–	4
Boston	79	12	4	4	4	–

(A) $m = 79 + 12 = 91$: Boston can get at most 91 points.



20.1.5.2 Constructing Flow Network

Notations
(A) S: set of teams,
(B) w_x wins for each team, and
(C) g_{xy} games left between x and y.
(D) m be the maximum number of wins for $\overline{z}$,
(E) and $S' = S \setminus \{\overline{z}\}$.

Reduction Construct the flow network G as follows
(A) One vertex v_x for each team $x \in S'$, one vertex u_{xy} for each pair of teams x and y in S'
(B) A new source vertex s and sink t
(C) Edges (u_{xy}, v_x) and (u_{xy}, v_y) of capacity ∞
(D) Edges (s, u_{xy}) of capacity g_{xy}
(E) Edges (v_x, t) of capacity equal $m - w_x$

20.1.5.3 Correctness of reduction

Theorem 20.1.7. *G' has a maximum flow of value $g^* = \sum_{x,y \in S'} g_{xy}$ if and only if $\overline{z}$ can win the most number of games (including possibly tie with other teams).*

20.1.5.4 Proof of Correctness

Proof: Existence of g^* flow $\Rightarrow \overline{z}$ wins pennant

(A) An integral flow saturating edges out of s, ensures that each remaining game between x and y is added to win total of either x or y

(B) Capacity on (v_x, t) edges ensures that no team wins more than m games

Conversely, $\overline{z}$ wins pennant $\Rightarrow$ flow of value g^*

(A) Scenario determines flow on edges; if x wins k of the games against y, then flow on (u_{xy}, v_x) edge is k and on (u_{xy}, v_y) edge is $g_{xy} - k$

■

20.1.5.5 Proof that $\overline{z}$ cannot with the pennant

(A) Suppose $\overline{z}$ cannot win the pennant since $g^* < g$. How do we *prove* to some one *compactly* that $\overline{z}$ cannot win the pennant?

(B) Show them the min-cut in the reduction flow network!

(C) See text book for a natural interpretation of the min-cut as a certificate.

20.1.6 An Application of Min-Cut to Project Scheduling

20.1.6.1 Project Scheduling

Problem:

(A) n projects/tasks $1, 2, \ldots, n$

(B) *dependencies* between projects: i depends on j implies i cannot be done unless j is done. dependency graph is *acyclic*

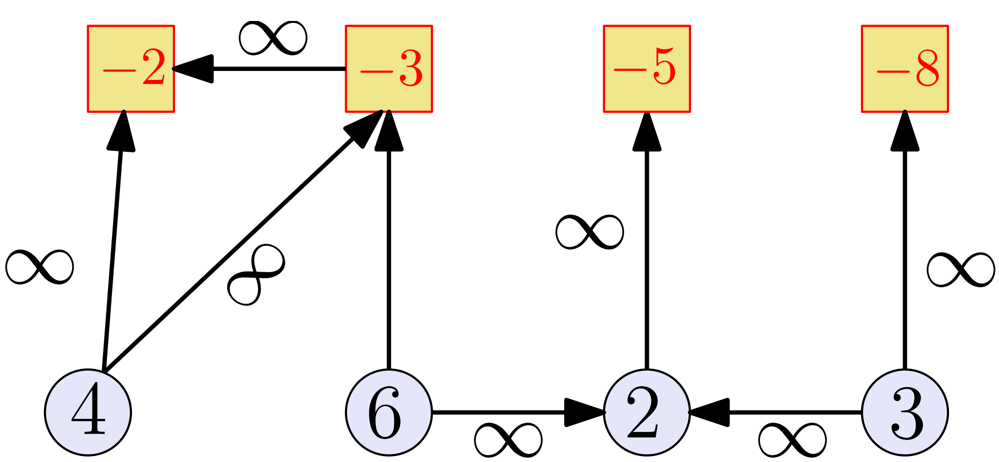
(C) each project i has a cost/profit p_i

  (A) $p_i < 0$ implies i requires a cost of $-p_i$ units

  (B) $p_i > 0$ implies that i generates p_i profit

Goal: Find projects to do so as to *maximize* profit.

20.1.6.2 Project selection example



20.1.6.3 Notation

For a set A of projects:

(A) A is a *valid* solution if A is *dependency closed*, that is for every $i \in A$, all projects that i depends on are also in A.
(B) $profit(A) = \sum_{i \in A} p_i$. Can be negative or positive.

Goal: find valid A to maximize $profit(A)$.

20.1.6.4 Idea: Reduction to Minimum-Cut

Finding a set of projects is partitioning the projects into two sets: those that are done and those that are not done.

Can we express this is a minimum cut problem?

Several issues:

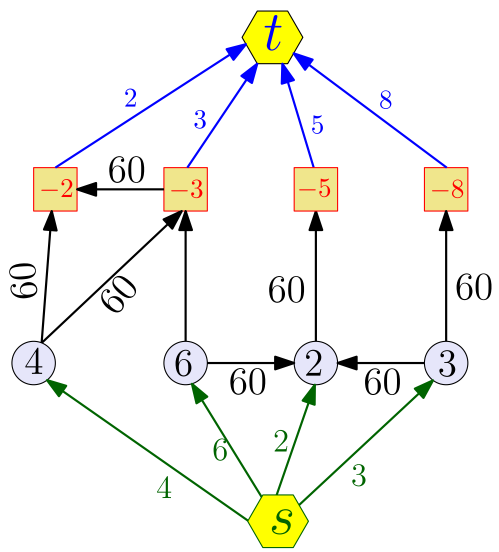
(A) We are interested in maximizing profit but we can solve minimum cuts.
(B) We need to convert negative profits into positive capacities.
(C) Need to ensure that chosen projects is a valid set.
(D) The cut value captures the profit of the chosen set of projects.

20.1.6.5 Reduction to Minimum-Cut

Note: We are reducing a *maximization* problem to a *minimization* problem.

(A) projects represented as nodes in a graph
(B) if i depends on j then (i, j) is an edge
(C) add source s and sink t
(D) for each i with $p_i > 0$ add edge (s, i) with capacity p_i
(E) for each i with $p_i < 0$ add edge (i, t) with capacity $-p_i$
(F) for each dependency edge (i, j) put capacity ∞ (more on this later)

20.1.6.6 Reduction: Flow Network Example



20.1.6.7 Reduction contd

Algorithm:

(A) form graph as in previous slide
(B) compute s-t minimum cut (A, B)
(C) output the projects in $A - \{s\}$

20.1.6.8 Understanding the Reduction

Let $C = \sum_{i:p_i>0} p_i$: maximum possible profit.

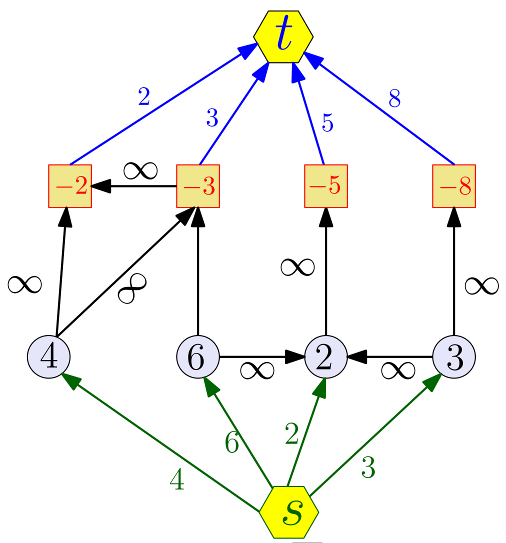
Observation: The minimum s-t cut value is $\leq C$. Why?

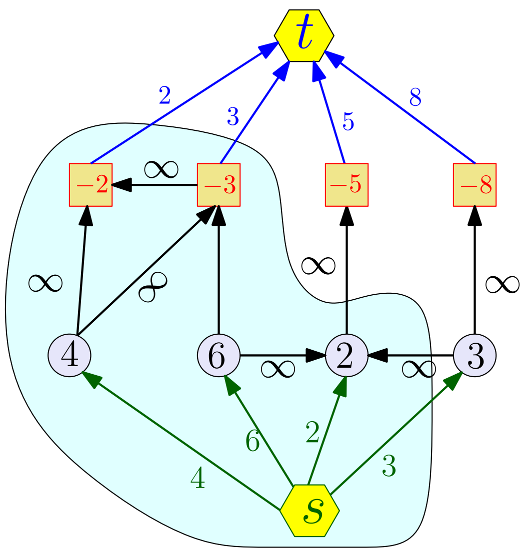
Lemma 20.1.8. *Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then projects in $A - \{s\}$ are a* valid *solution.*

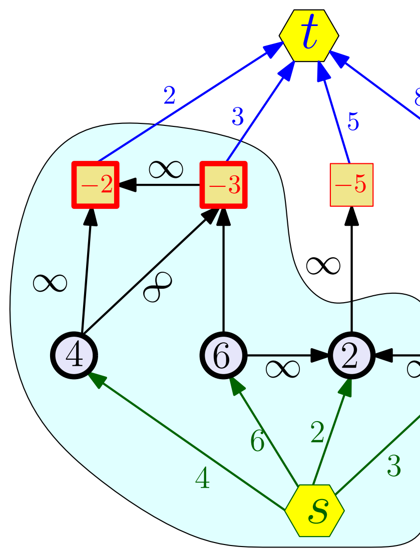
Proof: If $A - \{s\}$ is not a valid solution then there is a project $i \in A$ and a project $j \notin A$ such that i depends on j
Since (i, j) capacity is ∞, implies (A, B) capacity is ∞, contradicting assumption. ∎

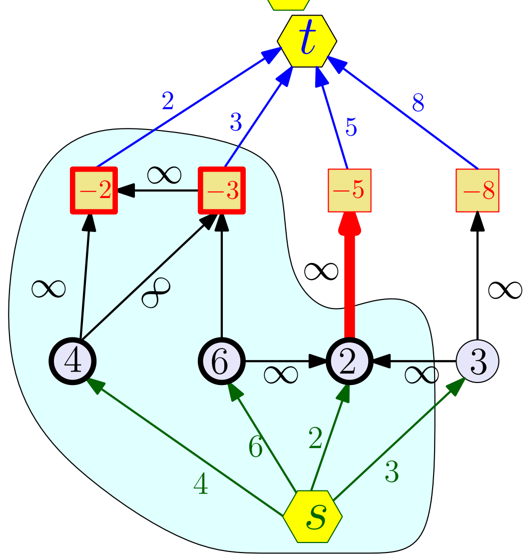
20.1.7 Reduction: Flow Network Example

20.1.7.1 Bad selection of projects



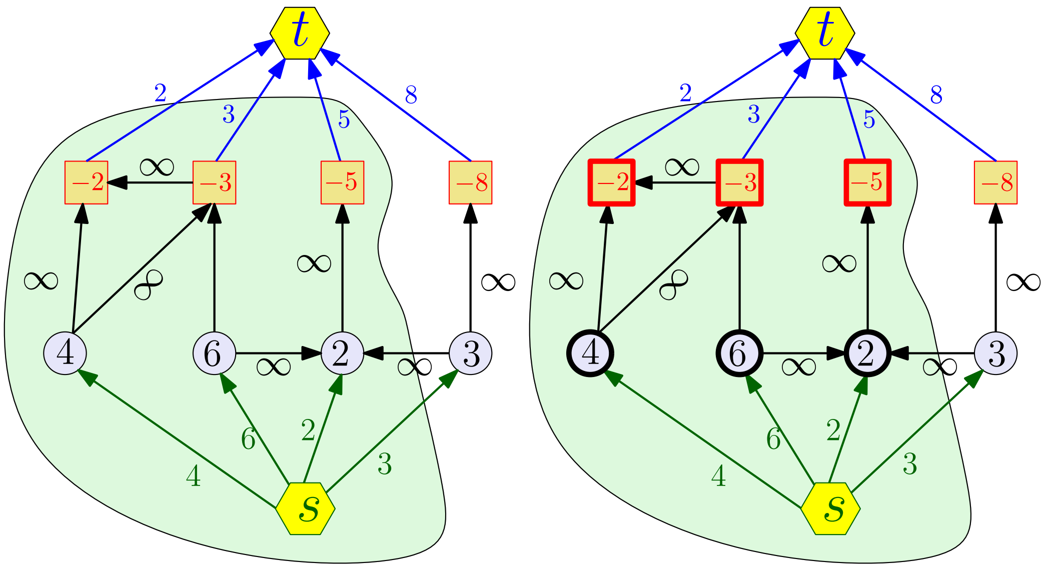






20.1.8 Reduction: Flow Network Example

20.1.8.1 Good selection of projects



20.1.8.2 Correctness of Reduction

Recall that for a set of projects X, $profit(X) = \sum_{i \in X} p_i$.

Lemma 20.1.9. *Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then $c(A, B) = C - profit(A - \{s\})$.*

Proof: Edges in (A, B):
(A) (s, i) for $i \in B$ and $p_i > 0$: capacity is p_i
(B) (i, t) for $i \in A$ and $p_i < 0$: capacity is $-p_i$
(C) cannot have ∞ edges

■

20.1.8.3 Proof contd

For project set A let
(A) $cost(A) = \sum_{i \in A: p_i < 0} -p_i$
(B) $benefit(A) = \sum_{i \in A: p_i > 0} p_i$
(C) $profit(A) = benefit(A) - cost(A)$.

Proof: Let $A' = A \cup \{s\}$.

$$\begin{aligned} c(A', B) &= cost(A) + benefit(B) \\ &= cost(A) - benefit(A) + benefit(A) + benefit(B) \\ &= -profit(A) + C \\ &= C - profit(A) \end{aligned}$$

■

20.1.8.4 Correctness of Reduction contd

We have shown that if (A, B) is an s-t cut in G with finite capacity then
(A) $A - \{s\}$ is a valid set of projects
(B) $c(A, B) = C - profit(A - \{s\})$
Therefore a *minimum* s-t cut (A^*, B^*) gives a *maximum* profit set of projects $A^* - \{s\}$ since C is fixed.
Question: How can we use ∞ in a real algorithm?
Set capacity of ∞ arcs to $C + 1$ instead. Why does this work?

20.1.9 Extensions to Maximum-Flow Problem

20.1.9.1 Lower Bounds and Costs

Two generalizations:
(A) flow satisfies $f(e) \leq c(e)$ for all e. suppose we are given *lower bounds* $\ell(e)$ for each e. can we find a flow such that $\ell(e) \leq f(e) \leq c(e)$ for all e?
(B) suppose we are given a cost $w(e)$ for each edge. cost of routing flow $f(e)$ on edge e is $w(e)f(e)$. can we (efficiently) find a flow (of at least some given quantity) at minimum cost?

Many applications.

20.1.9.2 Flows with Lower Bounds

Definition 20.1.10. *A flow in a network $G = (V, E)$, is a function $f : E \to \mathbb{R}^{\geq 0}$ such that*
(A) ***Capacity Constraint:*** *For each edge e, $f(e) \leq c(e)$*
(B) ***Lower Bound Constraint:*** *For each edge e, $f(e) \geq \ell(e)$*
(C) ***Conservation Constraint:*** *For each vertex v*

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Question: Given G and $c(e)$ and $\ell(e)$ for each e, is there a flow?
As difficult as finding an s-t maximum-flow without lower bounds!

20.1.9.3 Regular flow via lower bounds

Given usual flow network G with source s and sink t, create lower-bound flow network G' as follows:
(A) set $\ell(e) = 0$ for each e in G
(B) add new edge (t, s) with lower bound v and upper bound ∞

Claim 20.1.11. *There exists a flow of value v from s to t in G if and only if there* exists *a feasible flow with lower bounds in G'.*

Above reduction show that lower bounds on flows are naturally related to **circulations**. With lower bounds, cannot guarantee acyclic flows from s to t.

20.1.9.4 Flows with Lower Bounds

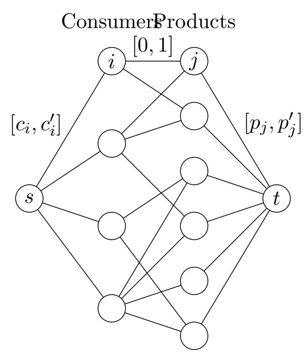
(A) Flows with lower bounds can be reduced to standard maximum flow problem. See text book. Reduction goes via circulations.
(B) If all bounds are integers then there is a flow that is integral. Useful in applications.

20.1.10 Survey Design

20.1.10.1 Application of Flows with Lower Bounds

(A) Design survey to find information about n_1 products from n_2 customers.
(B) Can ask customer questions only about products purchased in the past.
(C) Customer can only be asked about at most c'_i products and at least c_i products.
(D) For each product need to ask at east p_i consumers and at most p'_i consumers.

20.1.10.2 Reduction to Circulation



(A) include edge (i, j) is customer i has bought product j
(B) Add edge (t, s) with lower bound 0 and upper bound ∞.
    (A) Consumer i is asked about product j if the integral flow on edge (i, j) is 1

20.1.10.3 Minimum Cost Flows

(A) **Input:** Given a flow network G and also edge costs, $w(e)$ for edge e, and a flow requirement F.
(B) **Goal;** Find a *minimum cost* flow of value F from s to t
Given flow $f : E \to R^+$, cost of flow $= \sum_{e \in E} w(e)f(e)$.

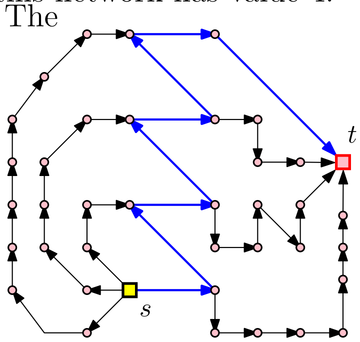
20.1.10.4 Minimum Cost Flow: Facts

(A) problem can be solved efficiently in polynomial time
    (A) $O(nm \log C \log(nW))$ time algorithm where C is maximum edge capacity and W is maximum edge cost
    (B) $O(m \log n(m + n \log n))$ time strongly polynomial time algorithm
(B) for integer capacities there is always an optimum solutions in which flow is integral

20.1.10.5 How much damage can a single path cause?

Consider the following network. All the edges have capacity 1. Clearly the maximum flow in this network has value 4.

The network



Why removing the shortest path might ruin everything

(A) However... The shortest path between s and t is the blue path.

(B) And if we remove the shortest path, s and t become disconnected, and the maximum flow drop to 0.