10 Network problems

Historical note

The Shortest Path Problem is one of the most important efficient computational tools at our disposal. It is used everywhere, from GPS navigation and network communication to project management, layout design, robotics, computer graphics, and the list goes on. Often many problems reduce to finding shortest paths or use shortest paths as guidelines to optimal solutions. First algorithms for the Shortest Path problem were designed by Ford (1956), Bellman (1958), and Moore (1959). For non-negative weights, a more efficient algorithm was first suggested by Dijkstra (1959). Since then, many improved and specialized algorithms have been developed, with close to linear running time. For instance, in the context of navigation, current record holder is the Hub labelling algorithm (2011), which uses precomputed distances from carefully chosen nodes–hubs to speed up path finding on maps from hours to tenths of microseconds. For All-pairs Shortest Path problem the first algorithms were founds by Shimbel (1953), and by Roy (1959), Floyd (1962), and Warshall (1962). A more efficient approach is a combination of Dijkstra's and Bellman-Ford's algorithms known as Johnson's algorithm (1977).

The Minimum Spanning Tree problem also has a rich history. The first known algorithm was developed by Borůvka (1926) for efficient distribution of electicity. Later independently discovered by many researchers over the years: Jarník (1930), Kruskal (1956), Prim (1957), and Dijkstra (1959). All are based on similar ideas and have very efficient, almost linear-time implementations. There also exist efficient parallel and distributed implementations. The latter is notably used in minimum spanning tree protocols found frequently in routing broadcast/multicast communication in computer and wireless networking.

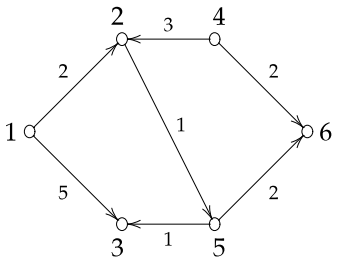
G = **graph** or **network** consists of

- a set V of **vertices** (nodes, points) and
- a set E of **edges** (arcs, lines) which are connections between vertices.

write $G = (V, E)$; write $V(G)$ for vertices of G, and $E(G)$ for edges of G.

(vertices are usually denoted u or v with subscripts; edges we usually denote e)

edges may have **direction**: an edge e between u and v may go from u to v, we write $e = (u, v)$,



vertices $V = \{1,2,3,4,5,6\}$

edges $E = \Big\{(1,2),(1,3),(2,5),(4,2),$
$(4,6),(5,3),(5,6)\Big\}$

weights $c(1,2) = 2 \quad c(1,3) = 5$
$c(2,5) = 1 \quad c(4,2) = 3 \quad c(4,6) = 2$
$c(5,3) = 1 \quad c(5,6) = 2$

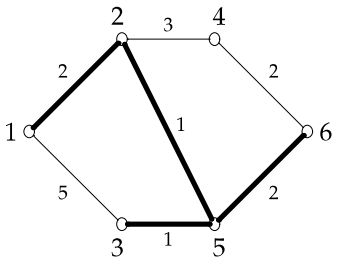


Figure 10.1: network (left), undirected network (right), edges $(1,2)$, $(2,5)$, $(3,5)$, $(5,6)$ form a tree of weight 5

or from v to u, we write $e = (v, u)$

(if an edge e does not have a direction, we treat it the same way as having both directions)

if all edges do not have a direction (are undirected), we say that the network is **undirected**

edges may have **weight**: a weight of edge $e = (u, v)$ is a real number denoted $c(e)$ or $c(u, v)$, c_e, c_{uv}

a sequence of nodes and edges $v_1, e_1, v_2, e_2, \ldots v_{k-1}, e_k, v_k$ is

- a **path** (directed path) if each e_i goes from v_i to v_{i+1}
- a **chain** (undirected path) if each e_i connects v_i and v_{i+1} (in some direction)

(often we write: $e_1, e_2, \ldots, e_k$ is a path (we omit vertices) or write: $v_1, v_2, \ldots, v_k$ is a path (we omit edges))

a network is **connected** if for every two nodes there is a path connecting them; otherwise it is **disconnected**

a **cycle** (loop, circuit) is a path starting and ending in the same node, never repeating any node or edge

a **forest** (acyclic graph) is an undirected graph that contains no cycles

a **tree** is a connected forest

Claim: A tree with n nodes contains exactly $n - 1$ edges. Adding any edge to a tree creates a cycle. Removing any edge from a tree creates a disconnected forest.

10.1 Shortest Path Problem

Given a network $G = (V, E)$ with two distinguished vertices $s, t \in V$, find a shortest path from s to t

Example: In Figure 1 (left), a shortest path from $s = 1$ to $t = 6$ is $1, 2, 5, 6$ of total length 5, while for $t = 3$ a shortest path is $1, 2, 5, 3$ of length 4. We say that **distance** from node 1 to node 6 is 5. Note that there is no path from s to $t = 4$; we indicate this by defining the distance to 4 as ∞.

LP formulation: decision variables x_{ij} for each $(i, j) \in E$

$$\text{Min} \quad \sum_{(i,j)\in E} w_{ij} x_{ij}$$

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad \text{for each } i \in V$$

$$x_{ij} \in \{0, 1\} \qquad \text{for each } (i, j) \in E$$

Modeling

A new car costs \$12,000. Annual maintenance costs are as follows: $m_1 = \$2,000$ first year, $m_2 = \$4,000$ second year, $m_3 = \$5,000$ third year, $m_4 = \$9,000$ fourth year, and $m_5 = \$12,000$ fifth year and on. The car can be sold for $s_1 = \$7,000$ in the first year, for $s_2 = \$6,000$ in the second year, for $s_3 = \$2,000$ in the third year, and for $s_4 = \$1,000$ in the fourth year of ownership.

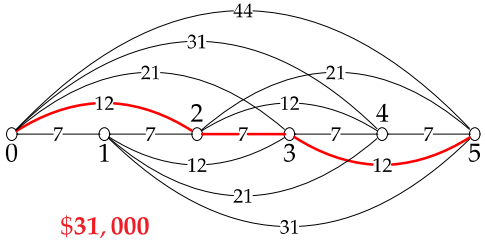
An existing car can be sold at any time and another new car purchased at \$12,000. What buying/selling strategy for the next 5 years minimizes the total cost of ownership?

Nodes = $\{0, 1, 2, 3, 4, 5\}$

Edge (i, j) represents the act of buying a car in year i and selling in year j. The weight is the price difference plus the maintanence cost, i.e., the weight is

$$c(i, j) = \$12,000 - s_{(i-j)} + m_1 + m_2 + \ldots + m_{(i-j)}$$

Answer: the length of a shortest path from node 0 to node 5.



A company wants to introduce a new model of its ever-popular *cell phone* to the market. To gauge the demand, a limited batch of product is to be brought to the market. The product is assembled from two parts, a circuit board, and housing. In addition, workers need to be trained and raw material procured.

After the initial assesment, the project manager put together a list of all tasks along with estimated duration of each task. The tasks are interdependent, some need to finish before others can start. First, (A) workers need to be trained and (B) raw materials purchased which takes 6 and 9 days, respectivelly. After these tasks are finished, both parts of the product, (C) the circuit board and (D) the housing, are manufactured taking 7 and 8 days, respectively. Each circuit board, once manufactured, needs to undergo additional testing (E) to comply with FCC regulation which takes 10 days. Afterwards, the cell phone is assembled, packaged, and shipped (F) which takes 12 days.

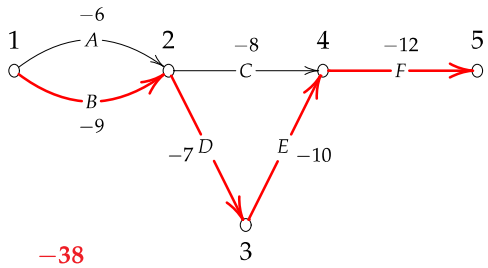
What is the minimum number of days before the product can reach the market? What is the **critical path** of the production, i.e., the sequence of tasks delaying any of which delays the whole production ?

We identify main check-points:
(1) start, (2) completion of tasks A and B,
(3) completion of task D, (4) completion of tasks C,E,
and (5) finish.

Nodes $\{1,2,3,4,5\}$

Edges correspond to tasks that connect checkpoints, weights are durations with negative sign.

Answer: A shortest path from 1 to 5 is a critical path.



(38 days = least time to hit the market)

Dijkstra's algorithm

Algorithm finds the length of a shortest path from s to every vertex of G (not only t)

Weights of edges are assumed to be **non-negative**, else the algorithm may output incorrect answer.

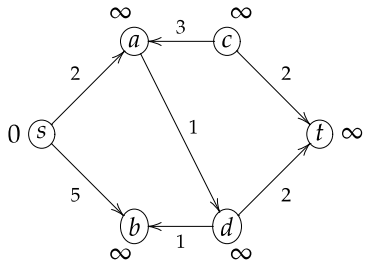
variables: d_u for each $u \in V$, an **estimate** on the distance from s to u

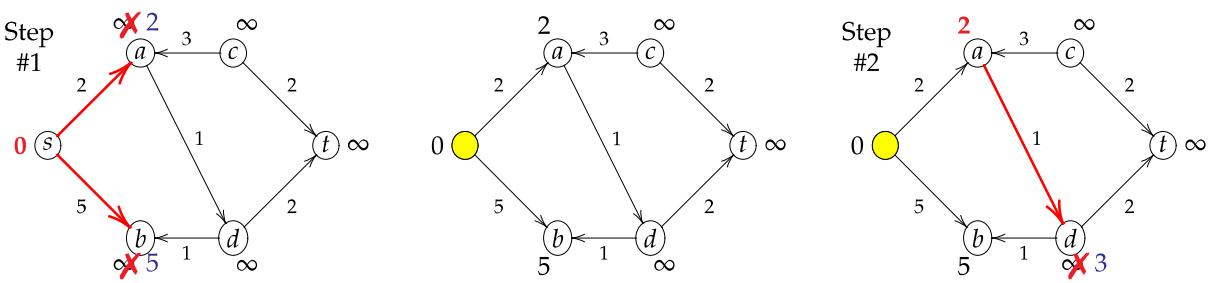
initialize: $d_u = \begin{cases} 0 & \text{if } u = s \\ \infty & \text{otherwise} \end{cases}$ all vertices are initially *unprocessed*

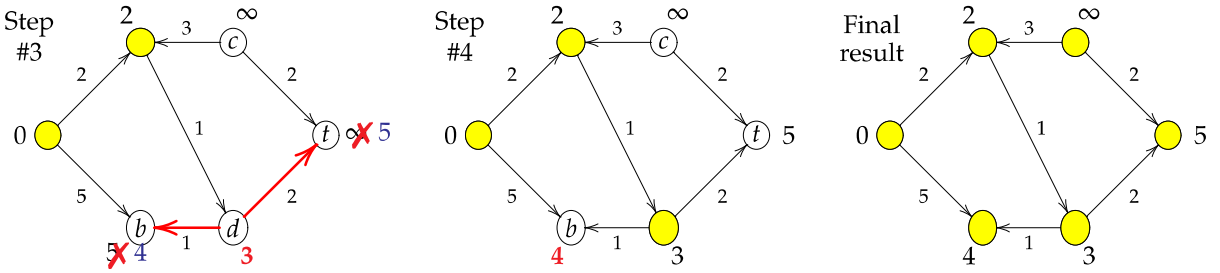
1. Find an *unprocessed* vertex u with smallest d_u
2. For each $(u,v) \in E$, update $d_v = \min\{d_v, d_u + c_{uv}\}$
3. Mark u as processed; repeat until all vertices are processed.
4. Report d_t as distance from s to t

Step#	s	a	b	c	d	t
1.	0	∞	∞	∞	∞	∞
		2	5			
2.	0*	2	5	∞	∞	∞
					3	
3.	0*	2*	5	∞	3	∞
			4			5
4.	0*	2*	4	∞	3*	5
5.	0*	2*	4*	∞	3*	5
6.	0*	2*	4*	∞	3*	5*
final	0*	2*	4*	∞*	3*	5*

Example:







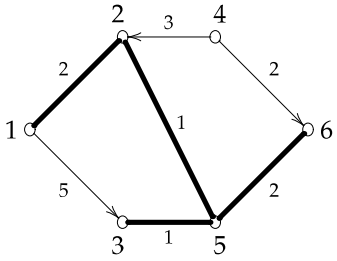
Can be augmented to actually find a shortest path.

2′ For each $(u,v) \in E$, if $d_v > d_u + w_{uv}$, then $d_v = d_u + w_{uv}$ and set $p_v = u$.

For the above example, we have:

$p_1 = undefined$, $p_2 = 1$, $p_3 = 5$, $p_4 = undefined$, $p_5 = 2$, $p_6 = 5$.

We can depict this by marking the edge from p_u to u for each $u \in V$.
(the picture on right). Note that the edges we marked form a tree.



A shortest path from 1 to 6 is found backwards by taking 6, p_6, p_{p_6}, ...
The path this yields is $1, 2, 5, 6$, since $5 = p_6$, $2 = p_5$, and $1 = p_2$.

10.2 Minimum Spanning Tree

A power company delivers electricity from its power plant to neighbouring cities. The cities are interconnected by power lines operated by various operators. The power company wants to rent power lines in the grid of least total cost that will allow it to send electricity from its power plant to all cities.

Given an undirected network $G = (V, E)$ find a collection $F \subseteq E$ of minimum weight so that (V, F) is a tree.

(we say that (V, F) is a **spanning** tree because it spans all vertices)

Example: The tree in Figure 1 (right) is not spanning because it does not contain the vertex 4. Adding the edge $(2,4)$ yields a spanning tree of weight 8, while adding the edge $(4,6)$ yields a spanning tree of weight 7. Note that adding the edge $(1,3)$ is not possible, for it creates a cycle $1, 2, 5, 3$ which is not allowed in a tree.

Kruskal's (Prim's) algorithm

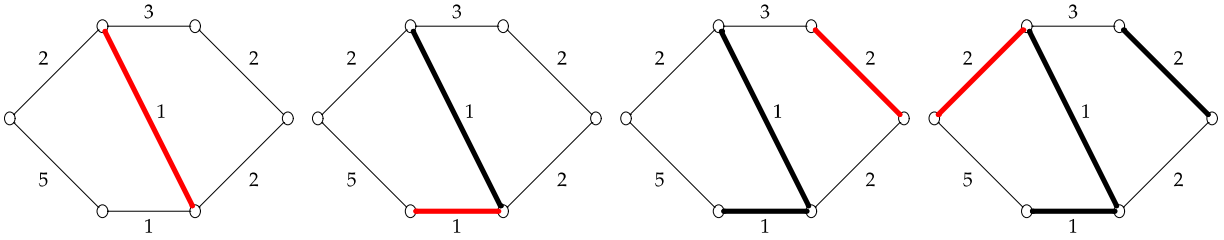
initialize: F to be empty; all edges are initially *unprocessed*

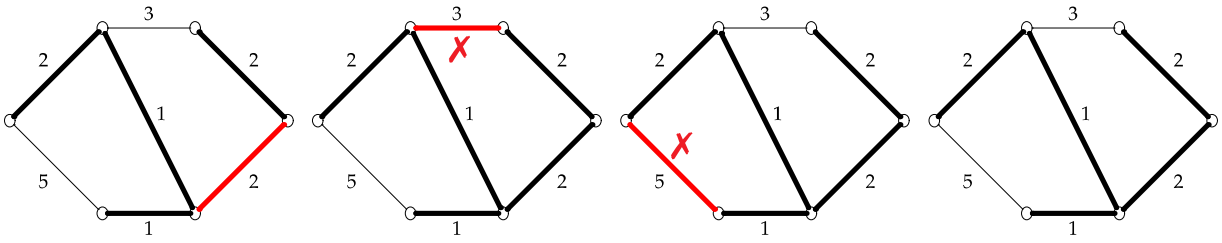
Kruskal's algorithm:

1. Find an unprocessed edge e of smallest weight w_e.
2. If $(V, F \cup \{e\})$ is a forest, then add e to F.
3. Mark e as processed and repeat until all edges have been processed.
4. Report (V, F) as a minimum-weight spanning tree.

Prim's algorithm: replace 1 by 1′

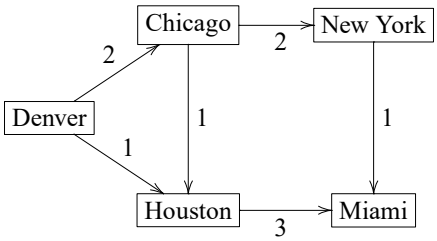
1′ Find an unprocessed edge e of smallest weight that shares an endpoint with some edge in F





10.3 Maximum Flow problem

A delivery company runs a delivery network between major US cities. Selected cities are connected by routes as shown below. On each route a number of delivery trucks is dispatched daily (indicated by labels on the corresponding edges). A customer is interested in hiring the company to deliver his products daily from Denver to Miami, and needs to know how much product can be delivered on a daily basis.



$$\begin{array}{rrrrrl} \text{maximize} & z & & & & \\ -x_{DC} & -x_{DH} & & & & = -z \\ x_{DC} & & -x_{CH} & -x_{CN} & & = 0 \\ & x_{DH} & +x_{CH} & & -x_{HM} & = 0 \\ & & & x_{CN} & -x_{NM} & = 0 \\ & & & & x_{NM}+x_{HM} & = z \end{array}$$

conservation of flow

$$\begin{array}{l} 0 \le x_{DC} \le 2 \\ 0 \le x_{DH} \le 1 \\ 0 \le x_{CH} \le 1 \\ 0 \le x_{CN} \le 2 \\ 0 \le x_{NM} \le 1 \\ 0 \le x_{HM} \le 3 \end{array}$$

In general, network $G = (V, E)$:

$s =$ **source** (Denver) $\qquad u_{ij} =$ *capacity* of an edge ij (# trucks dispatched daily between i and j)

$t =$ **sink** (Miami) $\qquad x_{ij} =$ *flow* on an edge ij (# trucks delivering the customer's products)

max z

$$\underbrace{\sum_{\substack{j\in V \\ ji\in E}} x_{ji}}_{\text{flow into } i} - \underbrace{\sum_{\substack{j\in V \\ ij\in E}} x_{ij}}_{\text{flow out of } i} = \begin{cases} -z & i = s \\ z & i = t \\ 0 & \text{otherwise} \end{cases}$$

$$0 \le x_{ij} \le u_{ij} \qquad \text{for all } ij \in E$$

Ford-Fulkerson algorithm

Initial feasible flow $x_{ij} = 0$ for all $ij \in E$.

A sequence of nodes $v_1, v_2, \ldots, v_n$ is a *chain* if $v_i v_{i+1} \in E$ (*forward* edge) or $v_{i+1} v_i \in E$ (*backward* edge) for all $i = 1, \ldots, n-1$. If $v_1 = s$ and $v_n = t$, then we call it an (s,t)-chain. Consider an (s,t)-chain P.

The *residual capacity* of a forward edge ij on P is defined as $u_{ij} - x_{ij}$ (the remaining capacity on the edge ij). The *residual capacity* of a backward edge ij on P is defined as x_{ij} (the used capacity of the edge ij).
The *residual capacity* of P is the **minimum** taken over residual capacities of edges on P.

If the *residual capacity* of P is positive $\varepsilon > 0$, then P is an **augumenting chain**. If this happens, we can increase the flow by increasing the flow on all forward edges by ε, and decreasing the flow on all backward edges by ε. This yields a feasible flow of larger value $z + \varepsilon$. (Notice the similarity with the Transportation Problem and the ratio test in the Simplex Method – same thing in disguise.)

Optimality criterion: The flow x_{ij} is optimal if and only if there is no augmenting chain.

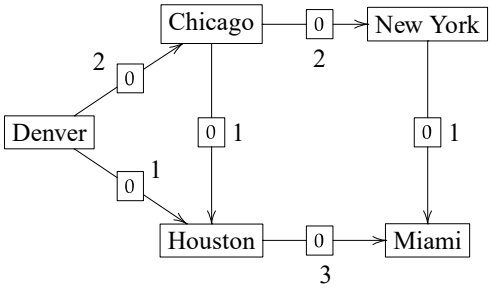
Residual network

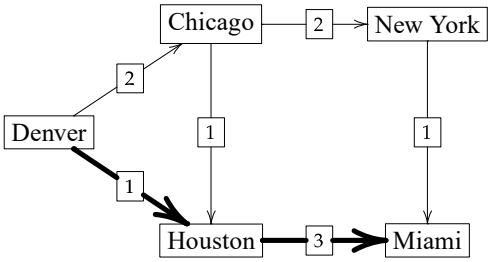
Finding an augmenting chain efficiently → residual network G_x constructed as follows:

- start by making V the nodes of G_x (no edges yet)
- then for every edge $ij \in E$,
  (a) add edge ij to G_x if $x_{ij} < u_{ij}$
  (b) add edge ji to G_x if $x_{ij} > 0$
  (if both happen then the residual network contains both the edge ij and ji)

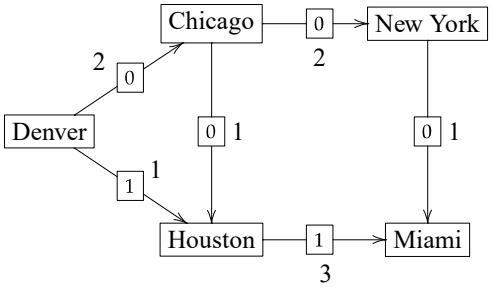
Any **path** from s to t in the residual network is an augmenting chain.

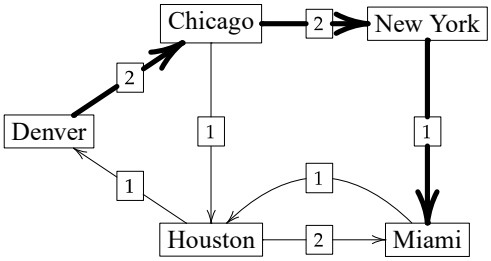
Starting feasible flow $x_{ij} = 0$ (indicated in boxes) → residual network (residual capacity shown on edges)



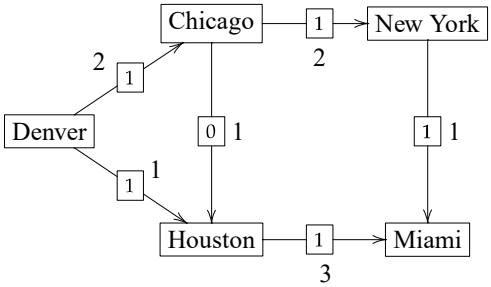


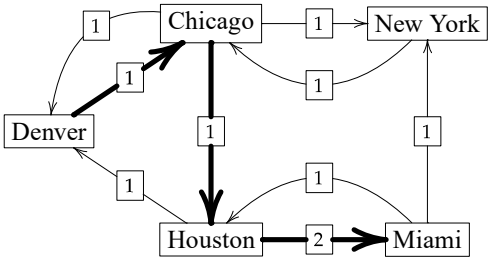
augmenting chain of residual capacity 1 → increase flow by 1



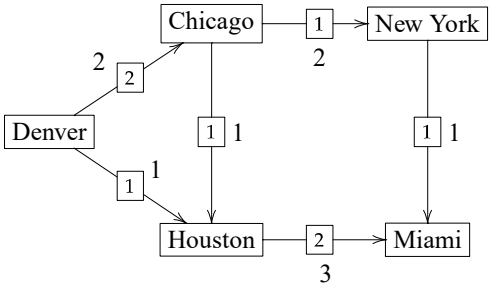


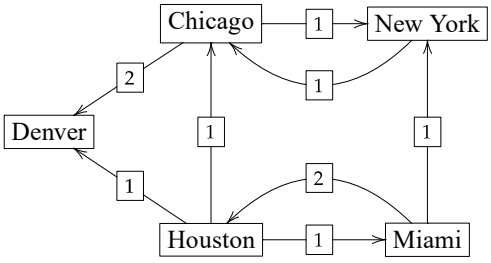
augmenting chain of residual capacity 1 → increase flow by 1





augmenting chain of residual capacity 1 → increase flow by 1





no path from Denver to Miami in the residual network → no augmenting chain → **optimal solution** found

→ maximum flow has value 3

Minimum Cut

For a subset of vertices $A \subseteq V$, the edges going between the nodes in A and the rest of the graph is called a **cut**. We write (A, A) to denote this cut. The edges going out of A are called **forward edges**, the edges coming into A are **backward edges**. If A contains s but not t, then it is an (s,t)-cut.

The **capacity** of a cut (A, A) is the sum of the capacities of its forward edges.

For example, let $A = \{\text{Denver,Chicago}\}$. Then (A, A) is an (s,t)-cut of capacity 4. Similarly, let $A_* = \{\text{Denver, Chicago, New York}\}$. Then (A_*, A_*) is an (s,t)-cut of capacity 3.

Theorem 5. *The maximum value of an (s,t)-flow is equal to the minimum capacity of an (s,t)-cut.*

This is known as the Max-Flow-Min-Cut theorem – a consequence of strong duality of linear programming.

maximize z

$$\begin{array}{rrrrrrr} -x_{DC} & -x_{DH} & & & & = & -z \\ x_{DC} & & -x_{CH} & -x_{CN} & & = & 0 \\ & x_{DH} & +x_{CH} & & -x_{HM} & = & 0 \\ & & & x_{CN} & -x_{NM} & = & 0 \\ & & & & x_{NM} & +x_{HM} = & z \end{array}$$

$$\begin{array}{l} 0 \le x_{DC} \le 2 \\ 0 \le x_{DH} \le 1 \\ 0 \le x_{CH} \le 1 \\ 0 \le x_{CN} \le 2 \\ 0 \le x_{NM} \le 1 \\ 0 \le x_{HM} \le 3 \end{array}$$

Dual:

minimize $2v_{DC} + v_{DH} + v_{CH} + 2v_{CN} + v_{NM} + 3v_{HM}$

$$\begin{array}{rrrrrrr} y_D & -y_C & & & & \le & v_{DC} \\ y_D & - & y_H & & & \le & v_{DH} \\ & y_C & -y_H & & & \le & v_{CH} \\ & y_C & - & y_N & & \le & v_{CN} \\ & & & y_N & -y_M & \le & v_{NM} \\ & & y_H & & -y_M & \le & v_{HM} \\ y_D & & & & -y_M & \ge & 1 \end{array}$$

$v_{DC}, v_{DH}, v_{CH}, v_{CN}, v_{NM}, v_{HM} \ge 0$

y_D, y_C, y_H, y_N, y_M unrestricted

Optimal solution (of value 3)

$y_D = y_C = y_N = 1 \quad \rightarrow \quad A = \{D, C, N\}$ min-cut

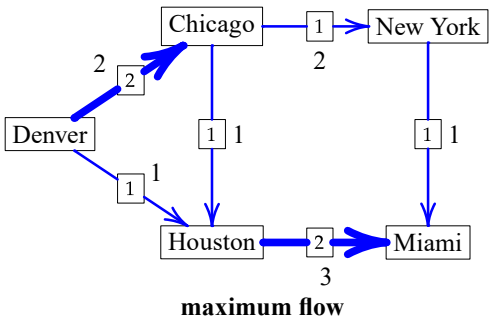
$y_H = y_M = 0$

$v_{DH} = v_{CH} = v_{NM} = 1$

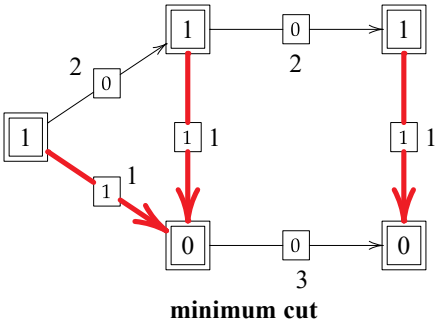
$v_{DC} = v_{CN} = v_{HM} = 0$

→ given an optimal solution, let A be the nodes whose y value is the same as that of source

→ (A, A) **minimum cut**



maximum flow



minimum cut

max z

$$\sum_{\substack{j \in V \\ ji \in E}} x_{ji} - \sum_{\substack{j \in V \\ ij \in E}} x_{ij} = \begin{cases} -z & i = s \\ z & i = t \\ 0 & \text{otherwise} \end{cases}$$

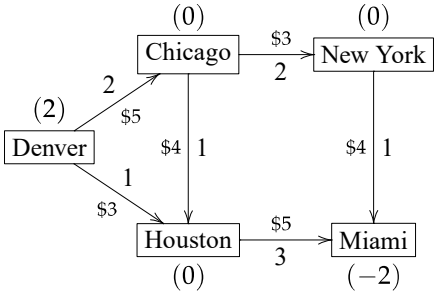
$$0 \le x_{ij} \le u_{ij} \qquad \text{for all } ij \in E$$

$$\min \sum_{ij \in E} u_{ij} v_{ij}$$

$$\begin{array}{rrrrr} y_i & - & y_j & \le & v_{ij} \qquad \text{for all } ij \in E \\ y_s & - & y_t & \ge & 1 \\ & & v_{ij} & \ge & 0 \qquad \text{for all } ij \in E \\ y_i & \text{unrestricted} & & & \text{for all } i \in V \end{array}$$

10.4 Minimum-cost Flow problem

A delivery company runs a delivery network between major US cities. Selected cities are connected by routes as shown below. On each route a number of delivery trucks is dispatched daily (indicated by labels on the corresponding edges). Delivering along each route incurs a certain cost (indicated by the $ figure (in thousands) on each edge). A customer hired the company to deliver two trucks worth of products from Denver to Miami. What is the least cost of delivering the products?



$$\begin{aligned}
&\text{minimize}\\
&5x_{DC}+3x_{DH}+4x_{CH}+3x_{CN}+4x_{NM}+5x_{HM}\\
&\begin{aligned}
x_{DC}+x_{DH} &= 2\\
-x_{DC}+x_{CH}+x_{CN} &= 0\\
-x_{DH}-x_{CH}+x_{HM} &= 0\\
-x_{CN}+x_{NM} &= 0\\
-x_{NM}-x_{HM} &= -2
\end{aligned}
\qquad \underbrace{\qquad\qquad}_{\text{conservation of flow}}\\
&\begin{aligned}
0 &\le x_{DC} \le 2\\
0 &\le x_{DH} \le 1\\
0 &\le x_{CH} \le 1\\
0 &\le x_{CN} \le 2\\
0 &\le x_{NM} \le 1\\
0 &\le x_{HM} \le 3
\end{aligned}
\end{aligned}$$

Minimum-cost Network Flow problem

Network $G=(V,E)$:

u_{ij} =*capacity* of an edge $(i,j)\in E$ (# trucks dispatched daily between i and j)

x_{ij} =*flow* on an edge $(i,j)\in E$ (# trucks delivering the customer's products)

c_{ij} =*cost* on an edge $(i,j)\in E$ (cost of transportation per each truck)

b_i =*net supply* of a vertex $i\in V$ (amount of products produced/consumed at node i)

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\underbrace{\sum_{\substack{j\in V\\ ij\in E}} x_{ij}}_{\text{flow out of } i} - \underbrace{\sum_{\substack{j\in V\\ ji\in E}} x_{ji}}_{\text{flow into } i} = \underbrace{b_i}_{\text{net supply}}$$

$$0 \le x_{ij} \le u_{ij} \qquad \text{for all } ij\in E$$

Necessary condition: $\sum_i b_i = 0$.

If there are no capacity constraints, the problem is called the **Transshipment problem**.

10.5 Network Simplex Algorithm

Primal

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\sum_{\substack{j\in V\\ (i,j)\in E}} x_{ij} - \sum_{\substack{j\in V\\ (j,i)\in E}} x_{ji} = \underbrace{b_i}_{\text{net supply}} \qquad \text{for all } i\in V$$

$$x_{ij} \ge 0 \qquad \text{for all } (i,j)\in E$$

Dual

$$\max \sum_{i\in V} b_i y_i$$

$$y_i - y_j \le c_{ij} \qquad \text{for all } (i,j)\in E$$

$$y_i \text{ unrestricted} \qquad \text{for all } i\in V$$

The Network Simplex method is a specialized form of the Simplex method for solving the Minimum-cost network flow problem. That is, starting from a basic feasible solution, we seek to improve the solution by increasing values

10.8 Summary

Network $G = (V, E)$ has **nodes** V and **edges** E.

- Each edge $(i, j) \in E$ has a **capacity** u_{ij} and **cost** c_{ij}.
- Each vertex $i \in V$ provides **net supply** b_i.

For a set $S \subseteq V$, write S for $V \setminus S$ and write $E(S, S)$ for the set of edges $(i, j) \in E$ with $i \in S$ and $j \in S$. The pair (S, S) is called a **cut**. (Where applicable) there are two distinguished nodes: s =**source** and t =**sink**.

Minimum spanning tree

Primal

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\underbrace{\sum_{(i,j)\in E(S,S)} x_{ij}}_{\text{edges from } S \text{ to } S} > 0 \qquad \begin{array}{l}\text{for all } S \subseteq V \\ \text{where } \emptyset \neq S \neq V\end{array}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E$$

Obstruction (to feasibility):
set $S \subseteq V$ with $\emptyset \neq S \neq V$ such that $E(S, S) = \emptyset$

Shortest path problem

Primal

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\underbrace{\sum_{\substack{j\in V \\ (i,j)\in E}} x_{ij}}_{\text{flow out of } i} - \underbrace{\sum_{\substack{j\in V \\ (j,i)\in E}} x_{ji}}_{\text{flow into } i} = \begin{cases} 1 & i = s \\ -1 & i = t \\ 0 & \text{else} \end{cases} \qquad \text{for all } i \in V$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in E$$

Dual

$$\max y_s - y_t$$

$$y_i - y_j \leq c_{ij} \qquad \text{for all } (i,j) \in E$$

$$y_i \text{ unrestricted} \qquad \text{for all } i \in V$$

Obstruction (to feasibility): set $S \subseteq V$ with $s \in S$ and $t \in S$ such that $E(S, S) = \emptyset$

Maximum-flow problem

Primal

$$\max\ z$$

$$\sum_{\substack{j\in V \\ (i,j)\in E}} x_{ij} - \sum_{\substack{j\in V \\ (j,i)\in E}} x_{ji} = \begin{cases} z & i = s \\ -z & i = t \\ 0 & \text{else} \end{cases} \qquad \text{for all } i \in V$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \text{for all } (i,j) \in E$$

$$z \text{ unrestricted}$$

Dual

$$\min \sum_{(i,j)\in E} u_{ij}v_{ij}$$

$$y_i - y_j + v_{ij} \geq 0 \qquad \text{for all } (i,j) \in E$$

$$y_t - y_s = 1$$

$$v_{ij} \geq 0 \qquad \text{for all } (i,j) \in E$$

$$y_i \text{ unrestricted} \qquad \text{for all } i \in V$$

Obstruction (to feasibility): set $S \subseteq V$ with $s \in S$ and $t \in S$ such that $z > \underbrace{\sum_{(i,j)\in E(S,S)} u_{ij}}_{\text{capacity of the cut } (S,S)}$

(no flow bigger than the capacity of a cut)

Minimum-cost (s,t)-flow problem

Primal

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\sum_{\substack{j\in V\\(i,j)\in E}} x_{ij} - \sum_{\substack{j\in V\\(j,i)\in E}} x_{ji} = \begin{cases} f & i=s \\ -f & i=t \\ 0 & \text{else} \end{cases} \quad \text{for all } i\in V$$

$$0 \le x_{ij} \le u_{ij} \quad \text{for all } (i,j)\in E$$

Dual

$$\max fy_s - fy_t - \sum_{(i,j)\in E} u_{ij}v_{ij}$$

$$y_i - y_j - v_{ij} \le c_{ij} \quad \text{for all } (i,j)\in E$$

$$v_{ij} \ge 0 \quad \text{for all } (i,j)\in E$$

$$y_i \text{ unrestricted} \quad \text{for all } i\in V$$

Obstruction (to feasibility): set $S\subseteq V$ with $s\in S$ and $t\in S$ such that $f > \sum_{(i,j)\in E(S,\bar{S})} u_{ij}$

Transshipment problem

Primal

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\sum_{\substack{j\in V\\(i,j)\in E}} x_{ij} - \sum_{\substack{j\in V\\(j,i)\in E}} x_{ji} = \underbrace{b_i}_{\text{net supply}} \quad \text{for all } i\in V$$

$$x_{ij} \ge 0 \quad \text{for all } (i,j)\in E$$

Dual

$$\max \sum_{i\in V} b_i y_i$$

$$y_i - y_j \le c_{ij} \quad \text{for all } (i,j)\in E$$

$$y_i \text{ unrestricted} \quad \text{for all } i\in V$$

Obstruction (to feasibility): set $S\subseteq V$ such that $\sum_{i\in S} b_i > 0$ and $E(S,\bar{S}) = \emptyset$

Minimum-cost network flow problem

Primal

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\sum_{\substack{j\in V\\(i,j)\in E}} x_{ij} - \sum_{\substack{j\in V\\(j,i)\in E}} x_{ji} = b_i \quad \text{for all } i\in V$$

$$0 \le x_{ij} \le u_{ij} \quad \text{for all } (i,j)\in E$$

Dual

$$\max \sum_{i\in V} b_i y_i - \sum_{(i,j)\in E} u_{ij}v_{ij}$$

$$y_i - y_j - v_{ij} \le c_{ij} \quad \text{for all } (i,j)\in E$$

$$v_{ij} \ge 0 \quad \text{for all } (i,j)\in E$$

$$y_i \text{ unrestricted} \quad \text{for all } i\in V$$

Obstruction (to feasibility): set $S\subseteq V$ such that $\sum_{i\in S} b_i > \sum_{(i,j)\in E(S,\bar{S})} u_{ij}$