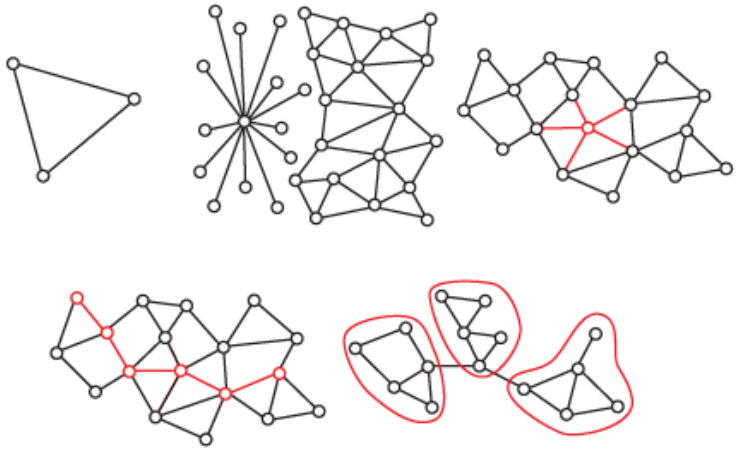
Arthur Charpentier

arthur.charpentier@univ-rennes1.fr

https://freakonometrics.github.io/

Université Rennes 1, 2017



Graphs, Networks & Flows

Network Flows & Matching

Ahuja, Magnanti & Orlin (1993) Network Flows: Theory, Algorithms, and Applications. Prentice-Hall

Bazaraa, Jarvis & Sherali (2010) Linear Programming and Network Flows Wiley.

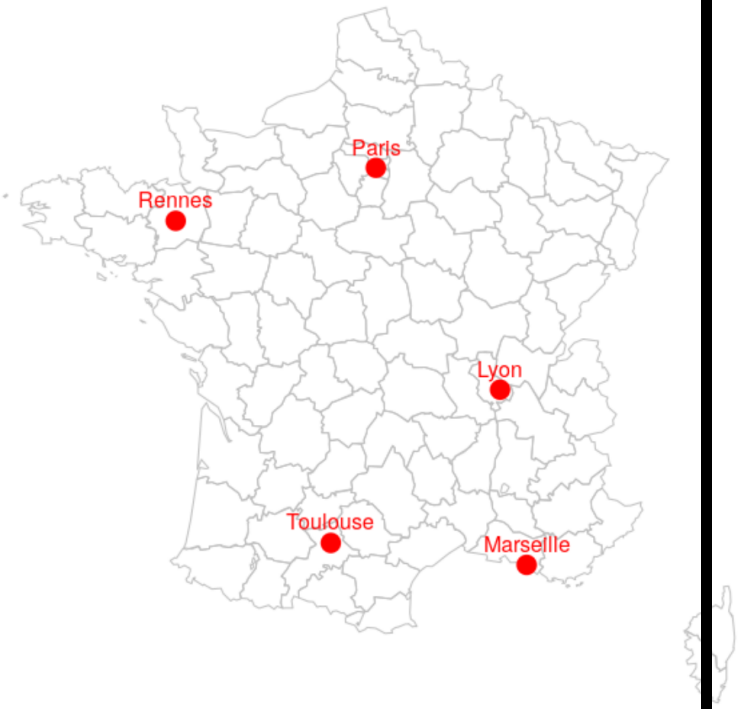
Ford & Fulkerson (1962) Flows in Networks. Princeton University Press.

Goldberg, Tardos & Tarjan Network Flow Algorithms

Traveling Salesman Problem

Consider 5 cities,

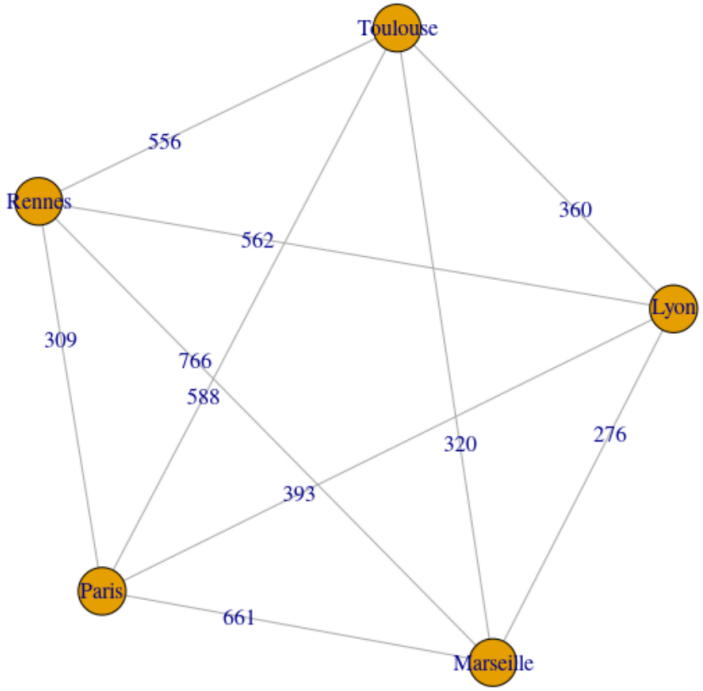
```
> v=c("Rennes","Paris","Lyon","Marseille","
    Toulouse")
> x=c(-1.6742900,2.348800,4.846710,
    5.38107,1.443670)
> y=c(48.1119800,48.853410,45.748460,
    43.29695,43.6042600)
> library(maps)
> france<-map(database="france",col="grey")
> points(x,y,pch=19,cex=2,col="red")
> text(x,y,v,pos=3,col="red")
```



It can also be seen as a complete weighted graph.

Traveling Salesman Problem

```
> df=data.frame(name=v,lat=y,lon=x,index
    =1:5)
> D=round(GeoDistanceInMetresMatrix(df)/
    1000)
> library(igraph)
> i=c(rep(1,4),rep(2,3),rep(3,2),4)
> j=c(2:5,3:5,4:5,5)
> df=data.frame(a = i, b=j, w=diag(D[i,j]))
> g=graph.data.frame(df, directed=FALSE)
> V(g)$label=v
> plot(g, edge.label=E(g)$w)
```



What is the path with minimal cost (length) that goes through all cities (and returns back home) ?

Traveling Salesman Problem

Can be formulated as an integer linear programming problem. Set $x_{ij} = A_{i,j}$ (x used since it is the unknown component), u_i be some dummy variable, and c_{ij} denote the distance from city i to city j. Consider

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

$$0 \leq x_{ij} \leq 1 \qquad i, j = 1, \ldots, n;$$

$$u_i \in \mathbb{Z} \qquad i = 1, \ldots, n;$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n;$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n;$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n.$$

Traveling Salesman Problem

Let $\boldsymbol{D} = [D_{i,j}]$ denote the distance matrix, with $D_{i,i} = \infty$, $c_{i,j}$ otherwise

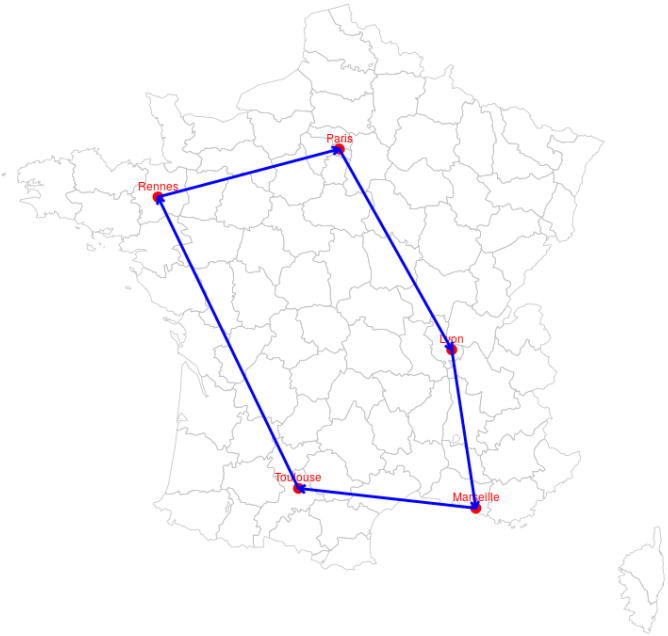
	Rennes	Paris	Lyon	Marseille	Toulouse
Rennes	∞	309	562	766	556
Paris	309	∞	393	661	588
Lyon	562	393	∞	276	360
Marseille	766	661	276	∞	320
Toulouse	556	588	360	320	∞

5 cities, $5! = 24$ possibilities (with assymmetric costs, otherwise 12).

Traveling Salesman Problem

One can use brute force search

```
> library(combinat)
> M=matrix(unlist(permn(1:5)),nrow=5)
> sM=M[,M[1,]==1]
> sM=rbind(sM,1)
> traj=function(vi){
+   s=0
+   for(i in 1:(length(vi)-1)) s=s+D[vi[i],
    vi[i+1]]
+   return(s) }
> d=unlist(lapply(1:ncol(sM),function(x)
    traj(sM[,x])))
> sM=rbind(sM,d)
> sM[,which.min(d)]
                                        d
  1    2    3    4    5    1 1854
```

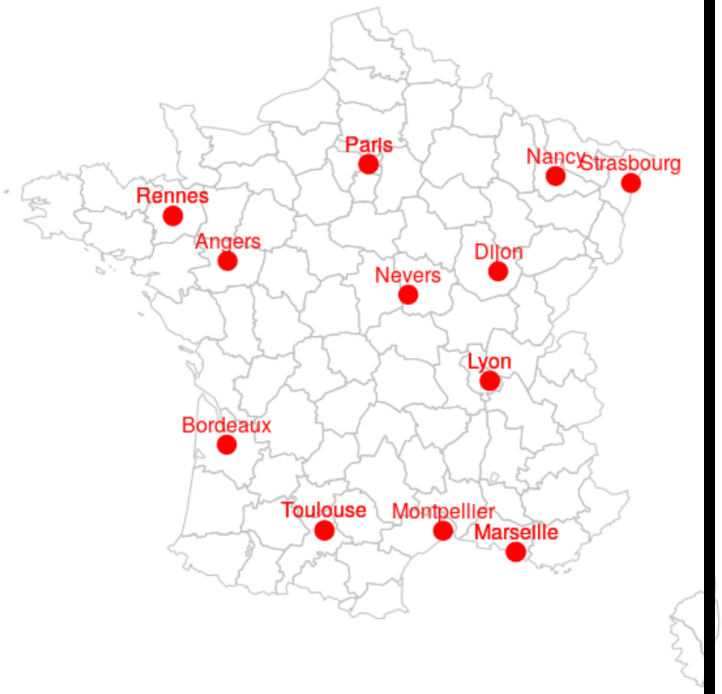


Traveling Salesman Problem

12 cities, $12! = 39,916,800$ possibilities (with assymmetric costs, otherwise 20 millions).

```
> v2=c(v,"Strasbourg","Angers","Bordeaux","
    Dijon","Nancy","Montpellier", "Nevers")
> x2=c(x
    ,7.75,-0.55,-0.566667,5.016667,6.2,3.883333

> y2=c(y
    ,48.583333,47.466667,44.833333,47.316667,48
     )
```



Little Algorithm

Branch and bound, Little algorithm, from Little, Murty, Sweeney & Karel (1963)
An algorithm for the traveling salesman problem

Step 1 find a lower bound for the total cost

Derive reduced matrix $\widetilde{\boldsymbol{D}}$ by removing to each raw the smallest element

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0	253	457	247	(-309)
Paris	0	∞	84	352	279	(-309)
Lyon	286	117	∞	0	84	(-276)
Marseille	490	385	0	∞	44	(-276)
Toulouse	236	268	40	0	∞	(-320)

Little Algorithm

and by removing to each column the smallest element

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0	253	457	203	(-309)
Paris	0	∞	84	352	235	(-309)
Lyon	286	117	∞	0	40	(-276)
Marseille	490	385	0	∞	0	(-276)
Toulouse	236	268	40	0	∞	(-320)
					(-44)	

Again $\boldsymbol{D}$ can be non-symmetric (and it will be afterwards).

Starting value for the cost: 309+309+276+276+320+44=1534

Little Algorithm

Step 2: compute regrets (min on rows and columns)

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0	253	457	203	(203)
Paris	0	∞	84	352	235	
Lyon	286	117	∞	0	40	
Marseille	490	385	0	∞	0	
Toulouse	236	268	40	0	∞	
		(117)				(320)

Little Algorithm

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0 (320)	253	457	203	
Paris	0	∞	84	352	235	(84)
Lyon	286	117	∞	0	40	
Marseille	490	385	0	∞	0	
Toulouse	236	268	40	0	∞	
	(236)					(320)

Little Algorithm

	Rennes	Paris	Lyon	Marseille	Toulouse	
Rennes	∞	0 (320)	253	457	203	
Paris	0 (320)	∞	84	352	235	
Lyon	286	117	∞	0	40	(40)
Marseille	490	385	0	∞	0	
Toulouse	236	268	40	0	∞	
				(0)		(40)

Little Algorithm

	Rennes	Paris	Lyon	Marseille	Toulouse
Rennes	∞	0 (320)	253	457	203
Paris	0 (320)	∞	84	352	235
Lyon	286	117	∞	0 (40)	40
Marseille	490	385	0 (40)	∞	0 (40)
Toulouse	236	268	40	0 (40)	∞

Select the maximal regret path (arbitrarily if not unique) : Rennes $\rightarrow$ Paris (RP).

Consider now two alternatives:

- either we keep (RP)
- either we remove (RP)

Little Algorithm

If we exclude (RP), the cost is known : 1534+320=1854.

If we include, consider the simplified matrix

	Rennes	Lyon	Marseille	Toulouse
Paris	∞	84	352	235
Lyon	286	∞	0	40
Marseille	490	0	∞	0
Toulouse	236	40	0	∞

Here (PR) is ∞ otherwise, it will be a round trip.

Little Algorithm

Then derive the reduced matrix

	Rennes	Lyon	Marseille	Toulouse	
Paris	∞	0	268	151	(-84)
Lyon	50	∞	0	40	(0)
Marseille	254	0	∞	0	(0)
Toulouse	0	40	0	∞	(0)
	(-236)				

Total (reduction) cost is 84+236=320.

Little Algorithm

And again, compute regrets

	Rennes	Lyon	Marseille	Toulouse
Paris	∞	0 (151)	268	151
Lyon	50	∞	0 (40)	40
Marseille	254	0 (0)	∞	0 (40)
Toulouse	0 (50)	40	0 (0)	∞

The maximal regret path is Paris $\rightarrow$ Lyon (PL). Consider now two alternatives:

- either we keep (PL)
- either we remove (PL) : cost will be 1534+320+151=2005

If we include, consider the simplified matrix (with possibly ∞ for (LP))

Little Algorithm

	Rennes	Marseille	Toulouse
Lyon	50	0	40
Marseille	254	∞	0
Toulouse	0	0	∞

Note that this matrix cannot be simplified, here.

	Rennes	Marseille	Toulouse	
Lyon	50	0	40	(0)
Marseille	254	∞	0	(0)
Toulouse	0	0	∞	(0)
	(0)	(0)	(0)	

So the total cost if we keep (RL) is 1534+320=1854

Little Algorithm

Then compute regrets

	Rennes	Marseille	Toulouse
Lyon	50	0 (40)	40
Marseille	254	∞	0 (294)
Toulouse	0 (50)	0 (0)	∞

The maximal regret path is Marseille $\rightarrow$ Toulouse (MT)

Consider now two alternatives:

- either we keep (MT)
- either we remove (MT) : cost will be 1854+294

If we include, consider the simplified matrix (with ∞ for (TM))

	Rennes	Marseille
Lyon	50	0
Toulouse	0	∞

Here again no simplification. The regret matrix is

	Rennes	Marseille
Lyon	50	0 (50)
Toulouse	0 (50)	∞

The maximal regret path is Lyon $\rightarrow$ Marseille (LM)

Consider now two alternatives:

- either we keep (MT): cost will be 1854+0
- either we remove (MT) : cost will be 1854+50

Little Algorithm

Here we have constructed a search tree.

click to visualize the construction

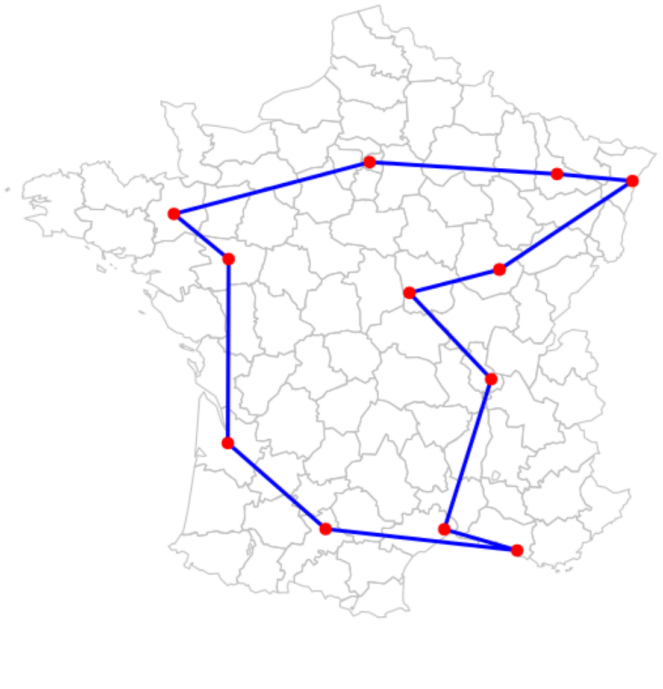
click to visualize the construction

Remark This is an exact algorithm

Traveling Salesman

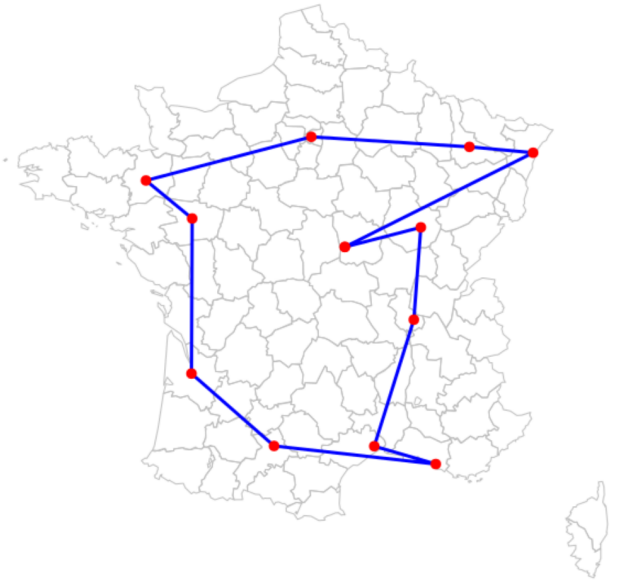
Exact algorithms are usually extremely slow. For Little Algorithm, the maximal number of nodes, with n cities, is of order N_n with $N_n = (n-1)[1+N_{n-1}]$ (with is of order $n!$).

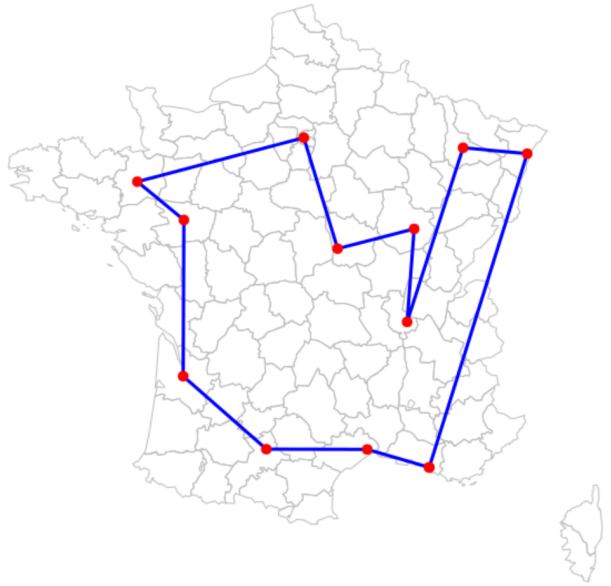
```
> library(TSP)
> df2=data.frame(name=v2,lat=y2,lon=x2,index
    =1:12)
> D2=round(GeoDistanceInMetresMatrix(df2)/
    1000)
> listeFR=TSP(D2,v2)
> tour=solve_TSP(listeFR, method = "nn")
> COORD=df2[as.numeric(tour),]
> COORD=rbind(COORD,COORD[1,])
> france<-map(database="france",col="grey")
> lines(COORD$lon,COORD$lat,lwd=3,col="blue"
    )
```



Traveling Salesman

```
> tour=solve_TSP(listeFR, method = "nn")
> COORD=df2[as.numeric(tour),]
> COORD=rbind(COORD,COORD[1,])
> france<-map(database="france",col="grey")
> lines(COORD$lon,COORD$lat,lwd=3,col="blue")
>
> tour=solve_TSP(listeFR, method = "nn")
> COORD=df2[as.numeric(tour),]
> COORD=rbind(COORD,COORD[1,])
> france<-map(database="france",col="grey")
> lines(COORD$lon,COORD$lat,lwd=3,col="blue")
```

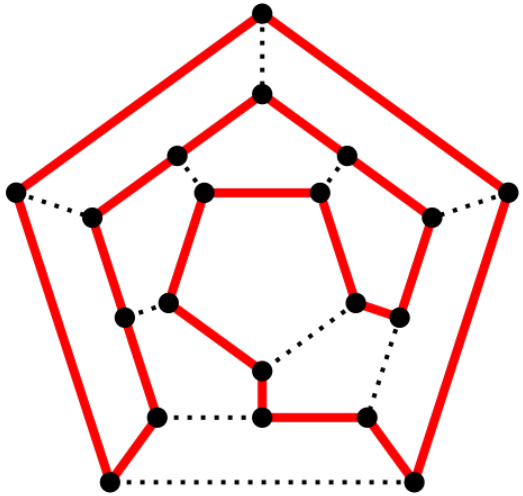




It is not stable... local optimization?

Traveling Salesman

Historically, introduced by William Rowan Hamilton in the context of dodecahedrons.
A Hamiltonian path is a path in agraph that visits each vertice exactly once
Done with $n = 20$ cities (regular aglebraic structure)



Dantzig, Fulkerson & Johnson (1954) Solution of a large-scale traveling-salesman problem with $n = 49$ cities (US State capitals)

In 2000, it was perfomed on $n = 15,000$ cities, nowadays close to 2 million.

See also Applegate *et al.* (2006) The traveling salesman problem: a computational study.

Stochastic Algorithm

One can use a Greedy Algorithm

Heuristically, makes locally optimal choices at each stage with the hope of finding a global optimum.

nearest neighbour, "*at each stage visit an unvisited city nearest to the current city*"

(1) start on an arbitrary vertex as current vertice.

(2) find out the shortest edge connecting current vertice and an unvisited vertice v.

(3) set current vertice to v.

(4) mark v as visited.

(5) if all the vertices in domain are visited, then terminate.

(6) Go to step 2.

click to visualize the construction

Stochastic Algorithm

E.g. descent algorithm, where we start from an initial guess, then consider at each step one vertice s, and in the neighborhood of s, denoted $V(s)$ search

$$f(s^\star) = \min_{s' \in V(s)} \{f(s')\}$$

If $f(s^\star) < f(s)$ then $s = s^\star$.

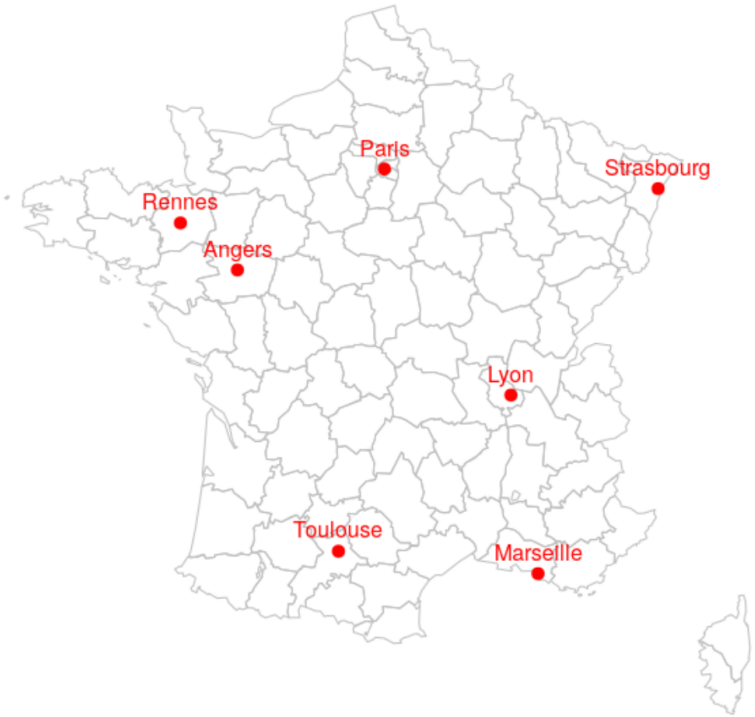
See Lin (1965) with 2 opt or 3 opt, see also Lin & Kernighan (1973) An Effective Heuristic Algorithm for the Traveling-Salesman Problem

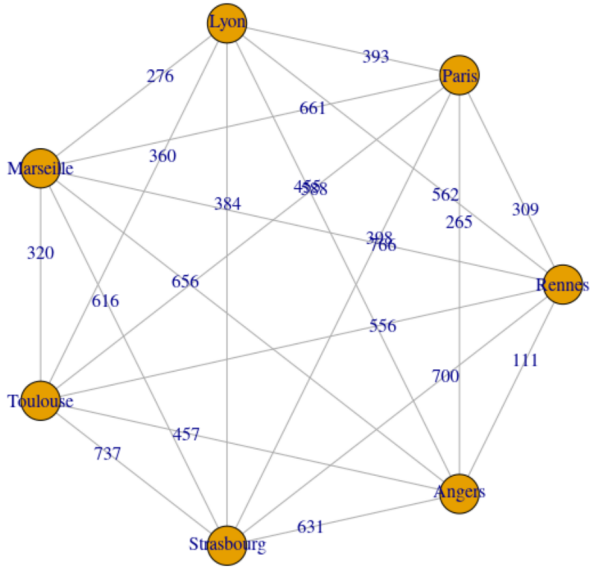
The pairwise exchange or 2-opt technique involves iteratively removing two edges and replacing these with two different edges that reconnect the fragments created by edge removal into a new and shorter tour.

click to visualize the construction

Spanning Algorithm

Kruskal's algorithm is a minimum-spanning-tree algorithm, see Kruskal (1956)
On the shortest spanning subtree of a graph and the traveling salesman problem



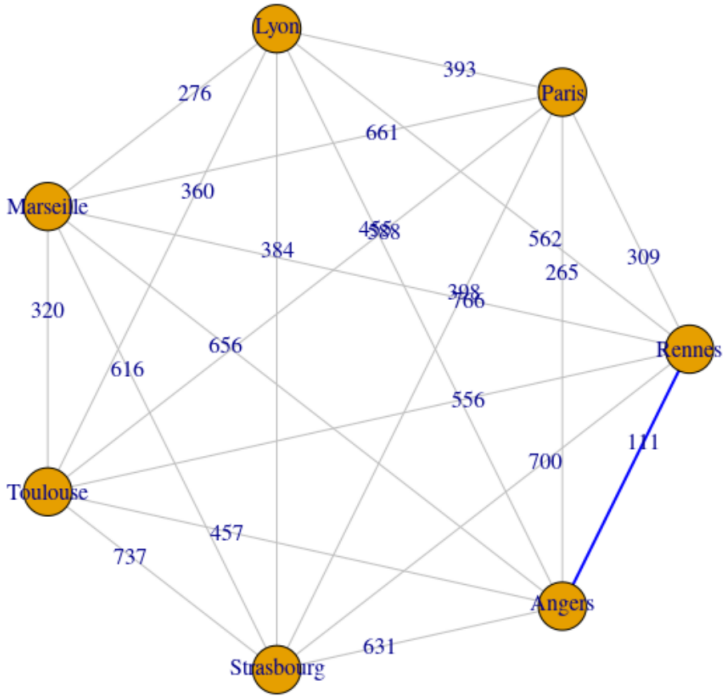


Spanning Algorithm

Select edge $e = (u, v) \in E$ which minimizes ω_ϵ,

$$e = \underset{\epsilon \in E}{\operatorname{argmin}}\{\omega_\epsilon\}$$

Then remove e from the set of electible edges
$\mathcal{E} = E \backslash \{e\}$
and mark that edge $\mathcal{F} = \{e\}$



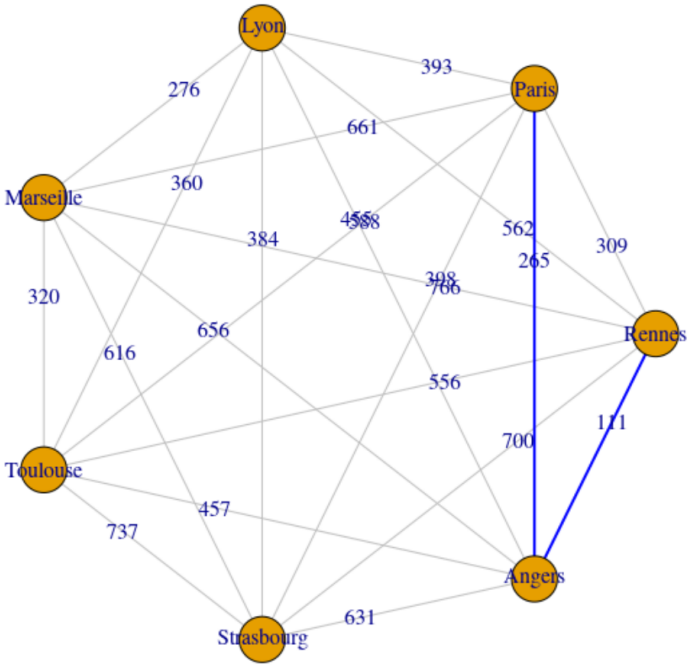
Spanning Algorithm

Select edge $e = (u, v) \in \mathcal{E}$ which minimizes ω_e,

$$e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$$

Then remove e from the set of electible edges
$\mathcal{E} = \mathcal{E} \backslash \{e\}$
and mark that edge $\mathcal{F} = \mathcal{F} \cup \{e\}$

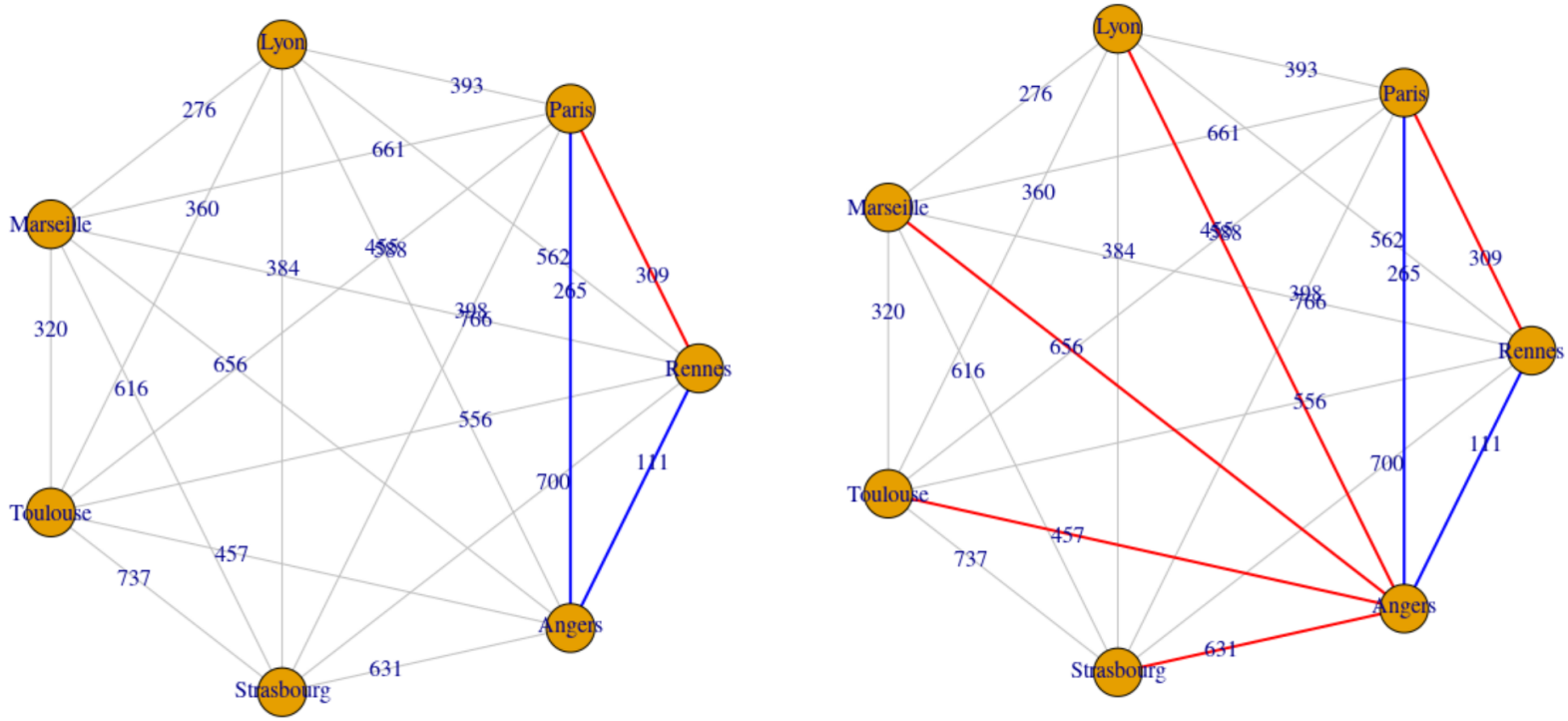
If the minima is not unique, choose randomly



Spanning Algorithm

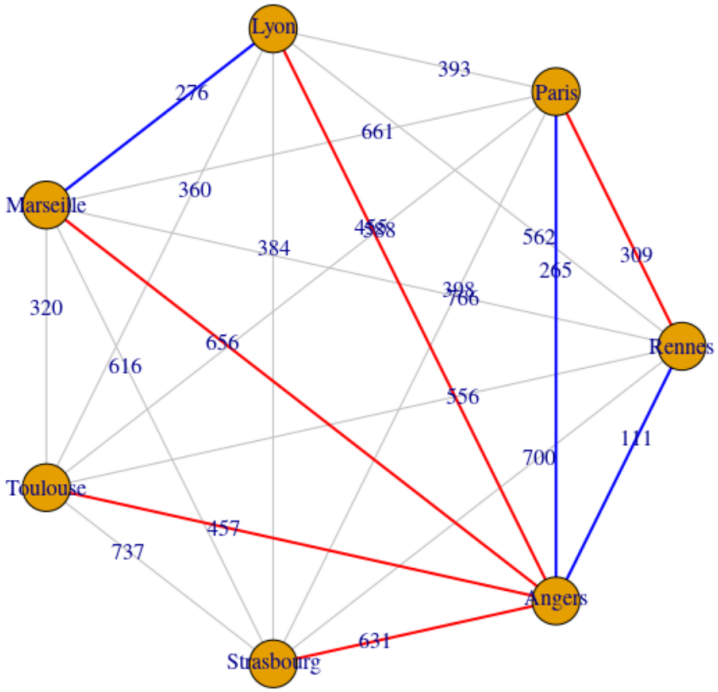
$\mathcal{E} = \mathcal{E} \backslash \{E_1 \cup E_2\}$ where we remove

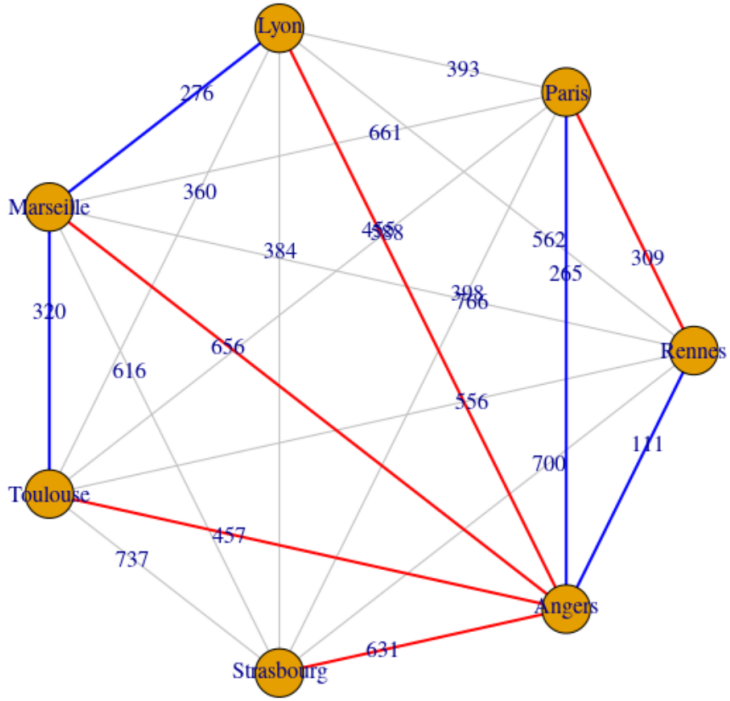
- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)



Spanning Algorithm

As long as there are no cycle, set $e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$ and then remove e from the set of electible edges $\mathcal{E} = \mathcal{E} \backslash \{e\}$, and mark those $\mathcal{F} = \mathcal{F} \cup \{e\}$

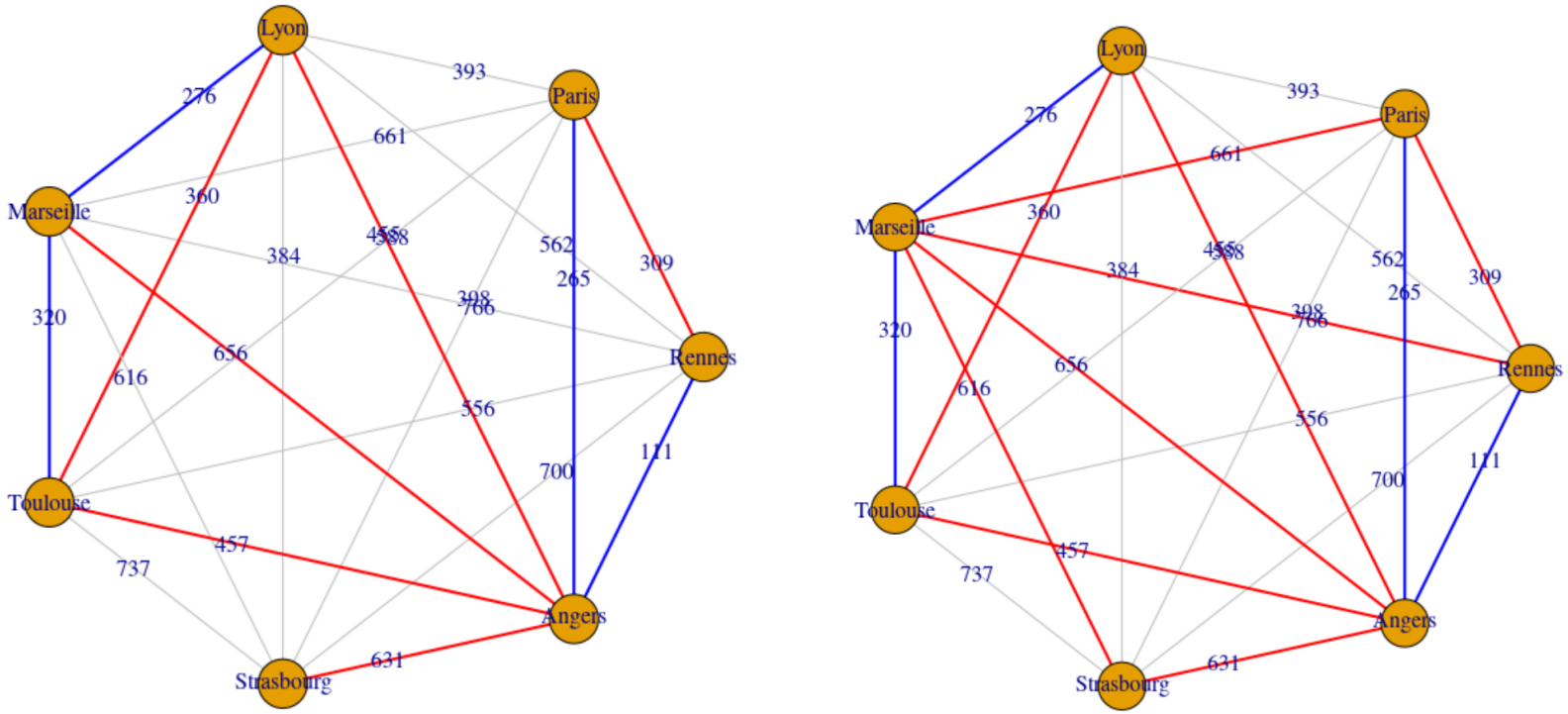




Spanning Algorithm

Here again remove

- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)

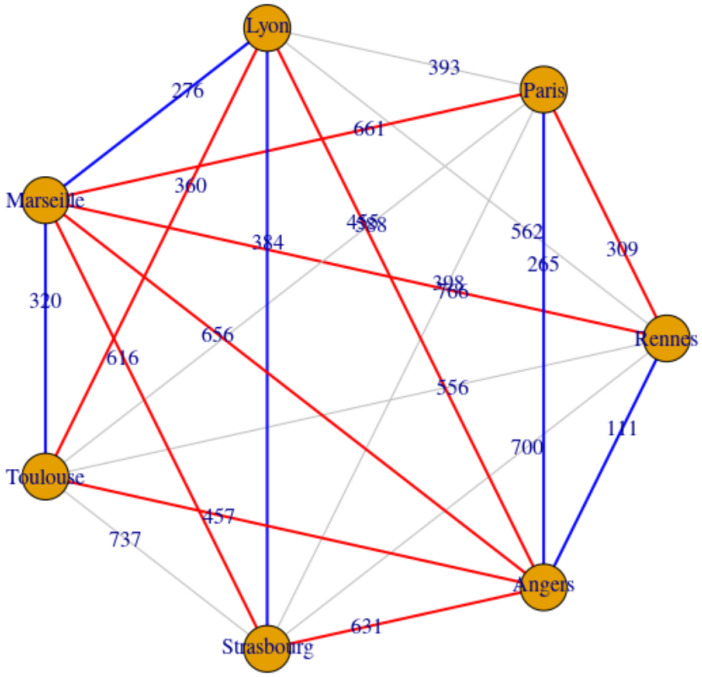


Spanning Algorithm

Select edge $e = (u, v) \in \mathcal{E}$ which minimizes ω_e,

$$e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$$

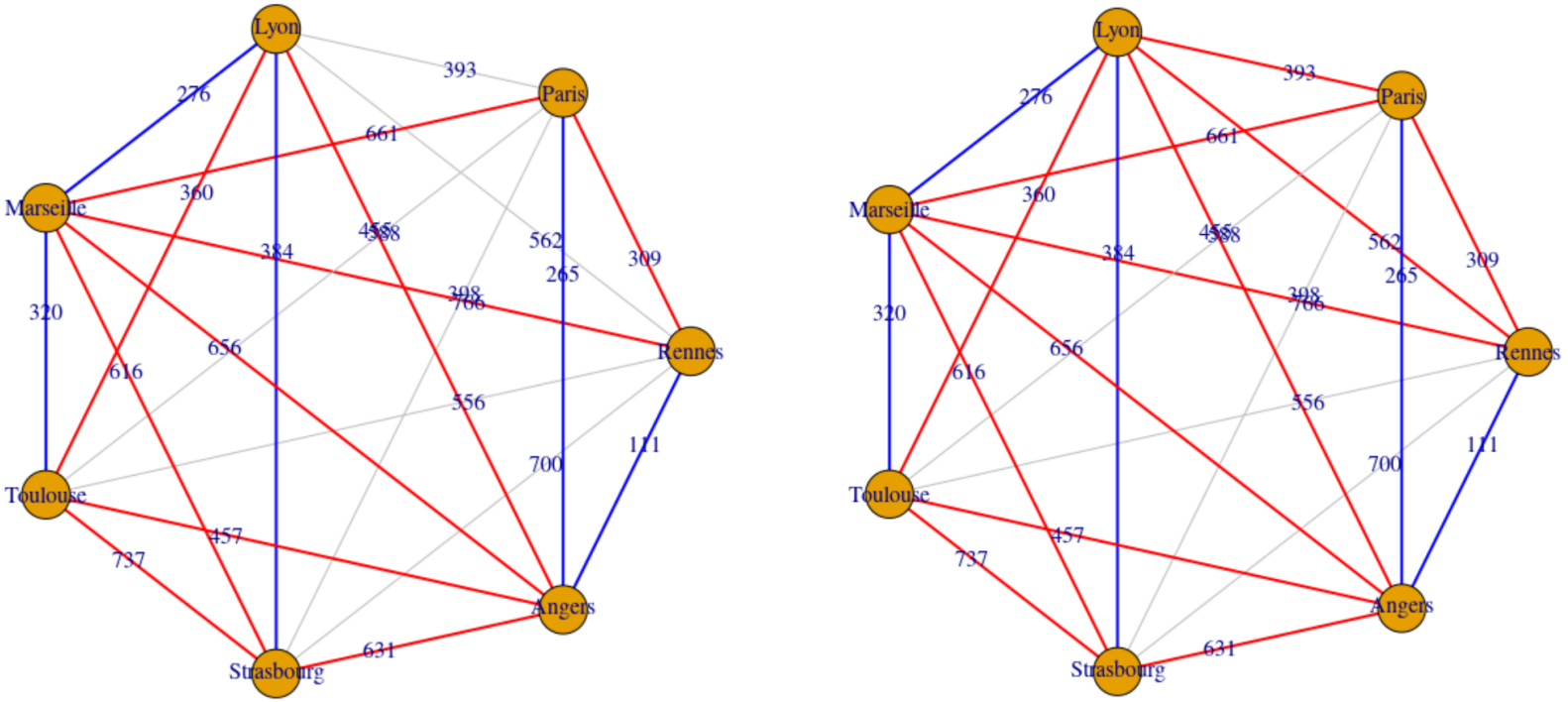
Then remove e from the set of electible edges
$\mathcal{E} = \mathcal{E} \backslash \{e\}$
and mark it $\mathcal{F} = \mathcal{F} \cup \{e\}$



Spanning Algorithm

Remove

- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)



Spanning Algorithm

Select edge $e = (u, v) \in \mathcal{E}$ which minimizes ω_ϵ,

$$e = \underset{\epsilon \in \mathcal{E}}{\text{argmin}}\{\omega_\epsilon\}$$

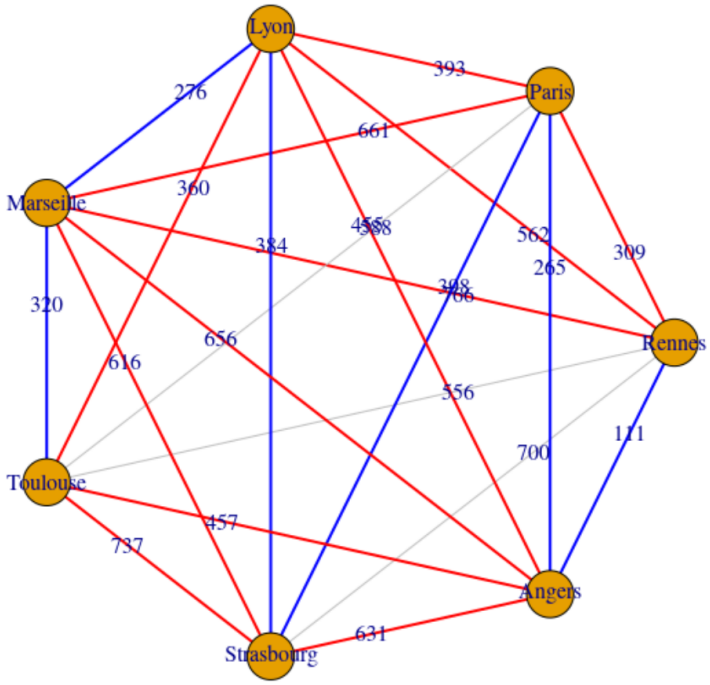
Then remove e from the set of electible edges

$\mathcal{E} = \mathcal{E} \backslash \{e\}$

and mark it $\mathcal{F} = \mathcal{F} \cup \{e\}$

Here e=Strasbourg-Paris

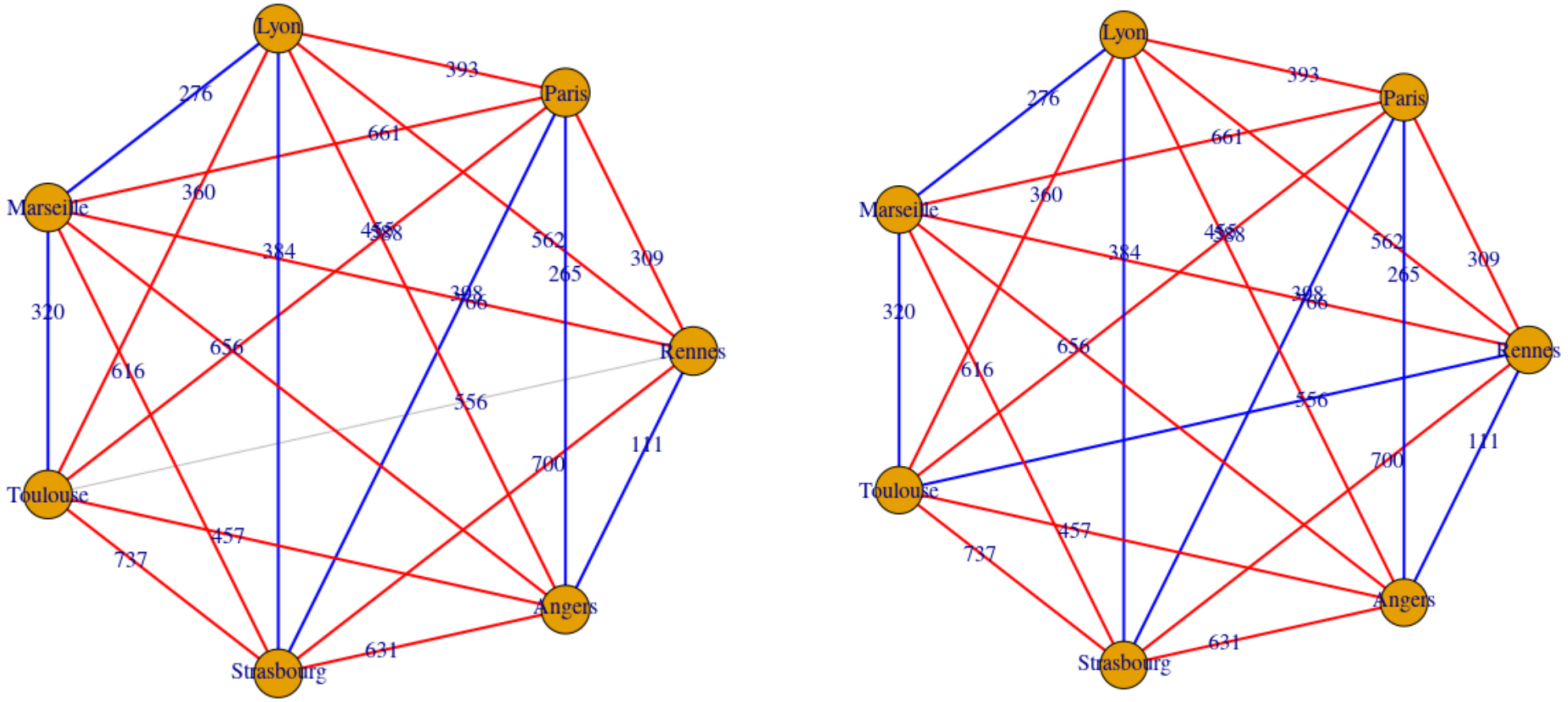
If the minima is not unique, choose randomly



Spanning Algorithm

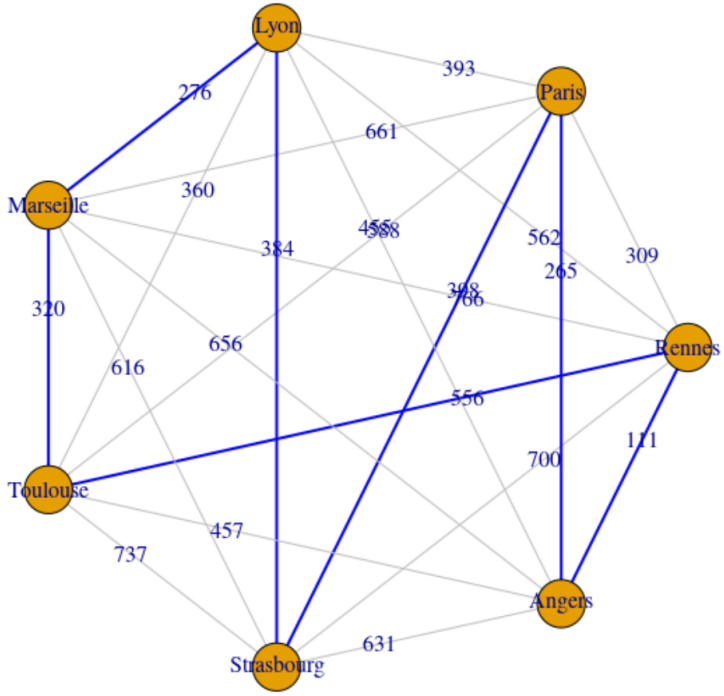
Last but not least, remove

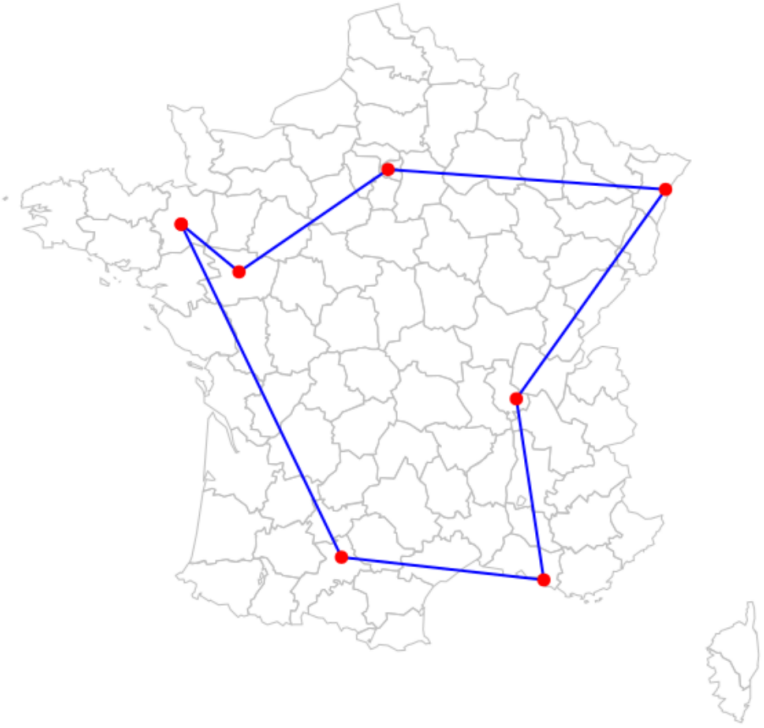
- all edges E_1 such that $\mathcal{E} \cup E_1$ contains a cycle (here Rennes-Paris)
- all edges from a node that appears twice in $\mathcal{E}$ (here all edges connected to Angers)



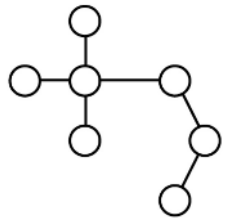
Spanning Algorithm

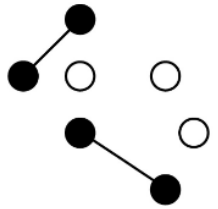
The collection of edges that were marked in $\mathcal{F}$ is our best cycle.





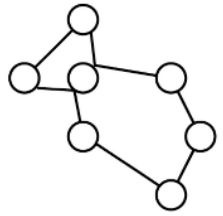
Christofides Algorithm





Use a spanning algorithm (as in Kruskal algorithm) $G = (V, E)$ but allow for vertices to have more than 2 incident edges.

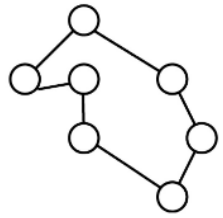
Let V_1 denote the set of vertices with odd degree, and let $G_1 = (V_1, E_1)$ denote the induced subgraph of $G = (V, E)$.



Find a perfect coupling G_1' of G_1 (discussed in the next course).

Consider graph G' with vertices V and edges $E \cup E_1'$.

If there are still vertices v with degree exceeding 2, consider shortcuts i.e. (u, v) and (v, w) becomes (u, w).

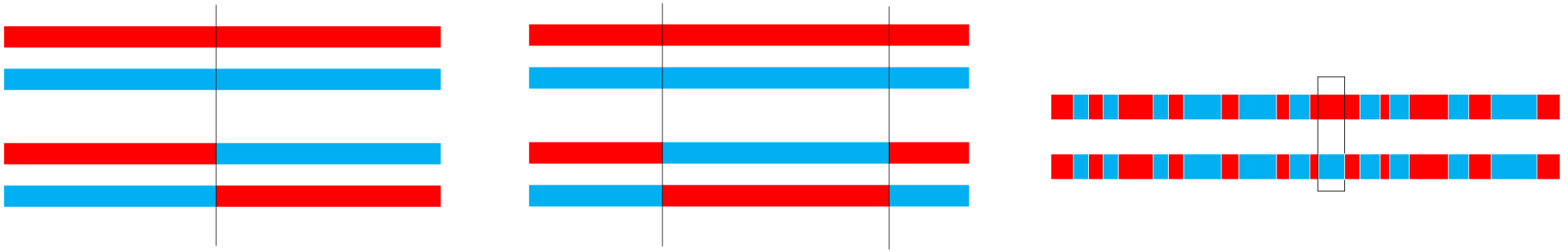


Genetic Algorithm

See Potvin (1996) Genetic algorithms for the traveling salesman problem

Mutation can be used to diversify solutions by alterating (randomly) an individual (possibly bit string)

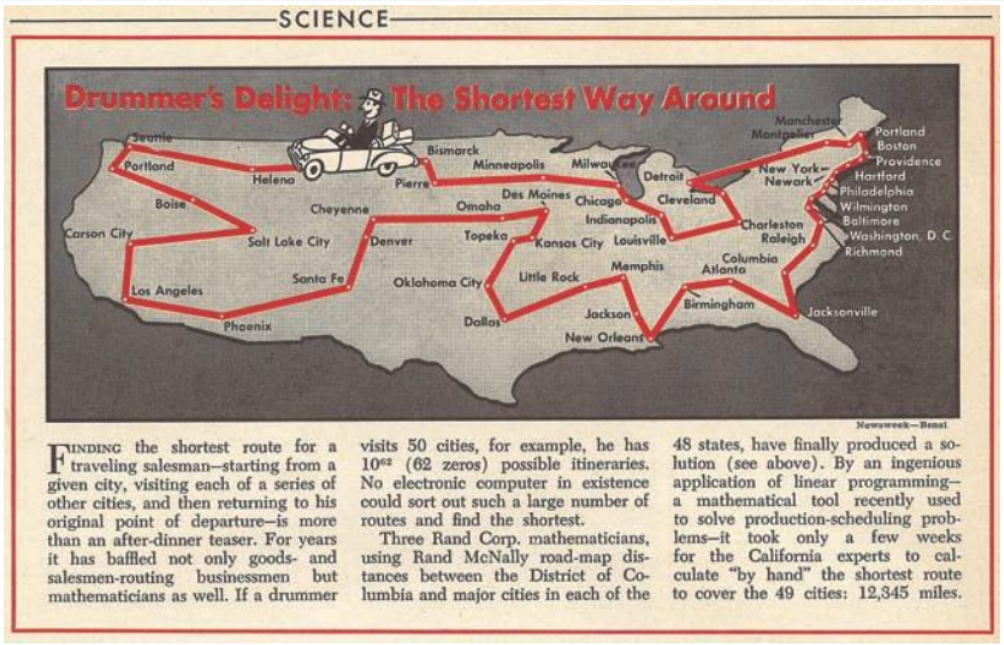
Cross-Over is obtained by mixing two individuals (single point or two points)



Traveling Salesman

Can be performed on a (much) larger scale
e.g. 48 State capitals in mainland U.S.

see Cook (2011) In Pursuit of the Traveling Salesman
for a survey.



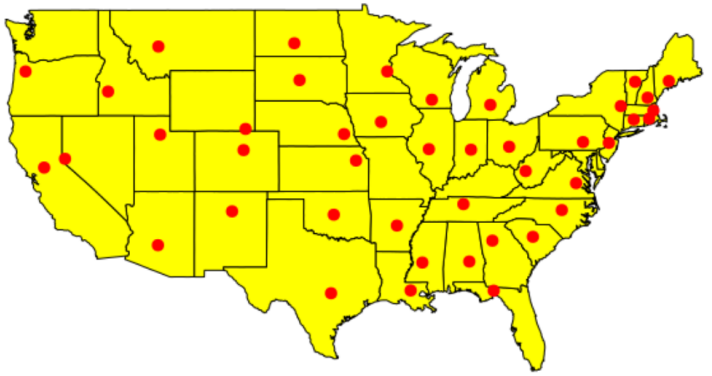
SCIENCE

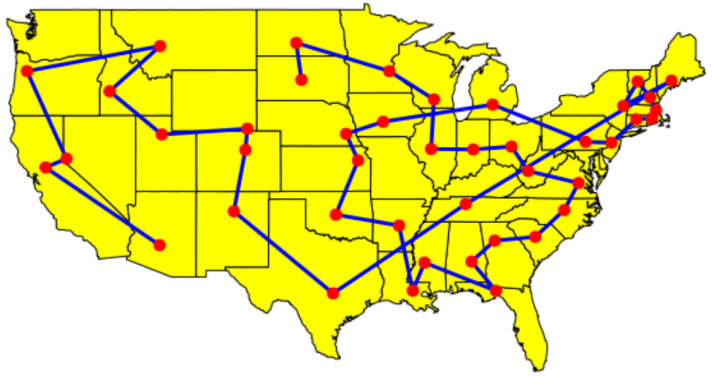
Drummer's Delight: The Shortest Way Around

Newsweek—Benzi

Finding the shortest route for a traveling salesman—starting from a given city, visiting each of a series of other cities, and then returning to his original point of departure—is more than an after-dinner teaser. For years it has baffled not only goods- and salesmen-routing businessmen but mathematicians as well. If a drummer visits 50 cities, for example, he has 10^{62} (62 zeros) possible itineraries. No electronic computer in existence could sort out such a large number of routes and find the shortest.

Three Rand Corp. mathematicians, using Rand McNally road-map distances between the District of Columbia and major cities in each of the 48 states, have finally produced a solution (see above). By an ingenious application of linear programming—a mathematical tool recently used to solve production-scheduling problems—it took only a few weeks for the California experts to calculate "by hand" the shortest route to cover the 49 cities: 12,345 miles.





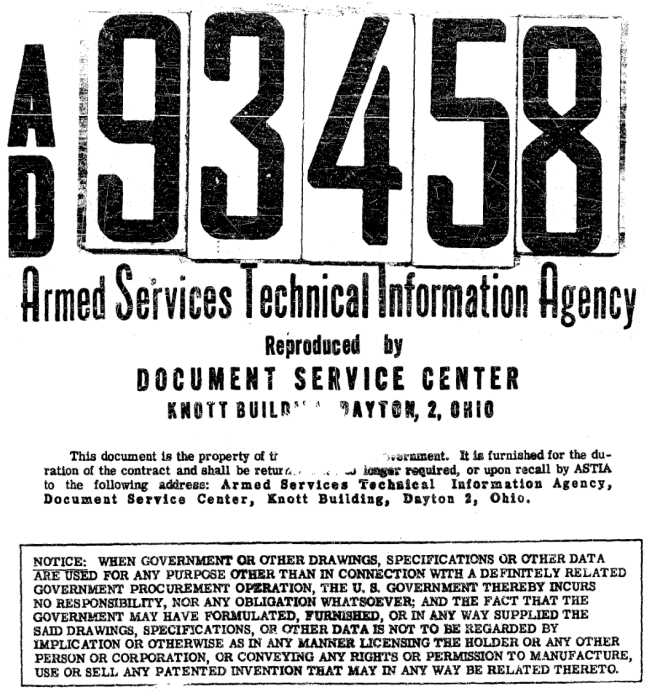
Application of the Traveling Salesman Problem

See Fischetti & Pisinger (2016) Optimizing wind farm cable routing considering power losses

See Fischetti & Monaci (2016) Proximity search heuristics for wind farm optimal layout

Networks & Flows : Motivation

Harris & Ross (1955) Fundamentals of a Method for Evaluating Rail Net Capacities



AD 93458

Armed Services Technical Information Agency

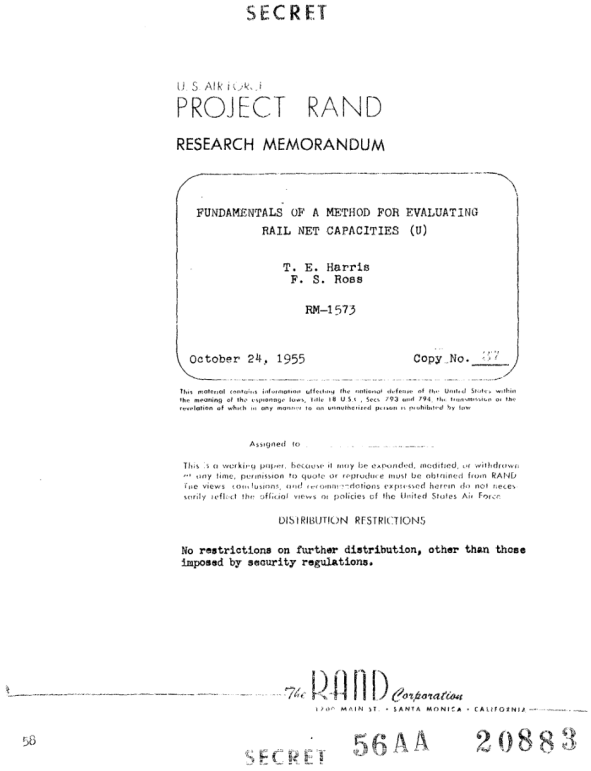
Reproduced by

DOCUMENT SERVICE CENTER

KNOTT BUILDING, DAYTON 2, OHIO

This document is the property of the Government. It is furnished for the duration of the contract and shall be returned when no longer required, or upon recall by ASTIA to the following address: Armed Services Technical Information Agency, Document Service Center, Knott Building, Dayton 2, Ohio.

NOTICE: WHEN GOVERNMENT OR OTHER DRAWINGS, SPECIFICATIONS OR OTHER DATA ARE USED FOR ANY PURPOSE OTHER THAN IN CONNECTION WITH A DEFINITELY RELATED GOVERNMENT PROCUREMENT OPERATION, THE U. S. GOVERNMENT THEREBY INCURS NO RESPONSIBILITY, NOR ANY OBLIGATION WHATSOEVER; AND THE FACT THAT THE GOVERNMENT MAY HAVE FORMULATED, FURNISHED, OR IN ANY WAY SUPPLIED THE SAID DRAWINGS, SPECIFICATIONS, OR OTHER DATA IS NOT TO BE REGARDED BY IMPLICATION OR OTHERWISE AS IN ANY MANNER LICENSING THE HOLDER OR ANY OTHER PERSON OR CORPORATION, OR CONVEYING ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE OR SELL ANY PATENTED INVENTION THAT MAY IN ANY WAY BE RELATED THERETO.



SECRET

U.S. AIR FORCE

PROJECT RAND

RESEARCH MEMORANDUM

FUNDAMENTALS OF A METHOD FOR EVALUATING RAIL NET CAPACITIES (U)

T. E. Harris
F. S. Ross

RM-1573

October 24, 1955 Copy No. 127

This material contains information affecting the national defense of the United States within the meaning of the espionage laws, Title 18 U.S.C., Secs. 793 and 794, the transmission or the revelation of which in any manner to an unauthorized person is prohibited by law.

Assigned to

This is a working paper. It may be expanded, modified, or withdrawn at any time. Permission to quote or reproduce must be obtained from RAND. The views, conclusions, and recommendations expressed herein do not necessarily reflect the official views or policies of the United States Air Force.

DISTRIBUTION RESTRICTIONS

No restrictions on further distribution, other than those imposed by security regulations.

The RAND Corporation

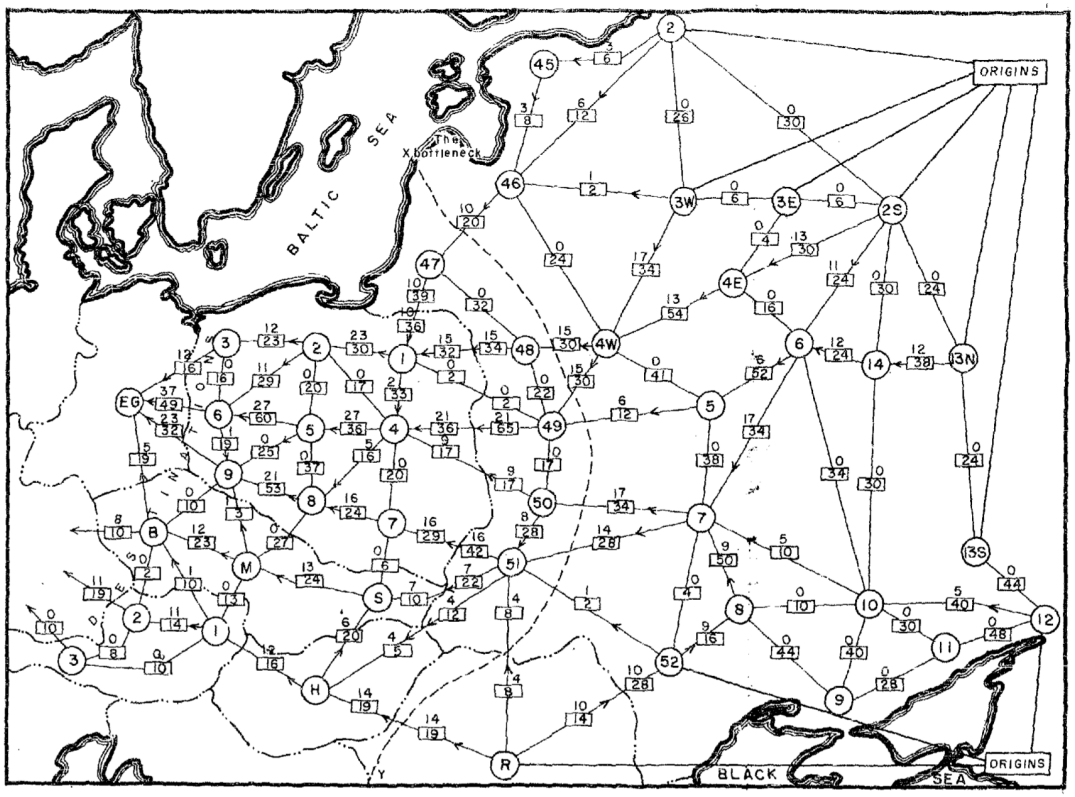
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

58

SECRET 56AA 20883

Networks & Flows : Motivation

Harris & Ross (1955) Fundamentals of a Method for Evaluating Rail Net Capacities



SECRET RM-1573 10-24-55 -59-

Fig. 8 — Traffic pattern: part of network available

Legend:
- International boundary
- (8) Railway operating division
- 9 / 12 Capacity: 12 each way per day. Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction.

All capacities in trains / 1000's of tons each way per day

Origins and destinations as in Fig. 7

Assumption:

Not to exceed 50% of any divisional capacity in USSR to be utilized for East–West military traffic and not to exceed 75% of similar satellite traffic to be utilized for East–West military traffic.

Networks & Flows : Motivation

Harris & Ross (1955) Fundamentals of a Method for Evaluating Rail Net Capacities

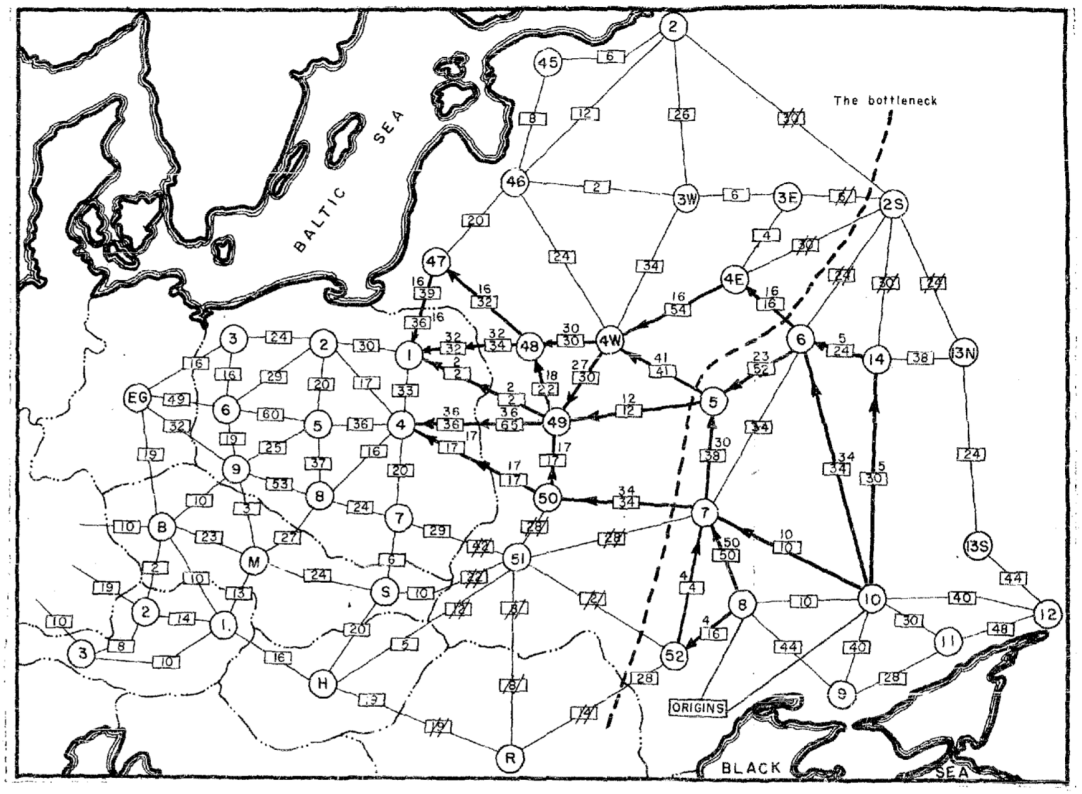


Fig. 9 — Traffic pattern: certain divisions inoperative

Legend:
— ·· — International boundary
(8) Railway operating division
9 / 12 — Capacity: 12 each way per day. Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction

All capacities in {trains / 1000's of tons} each way per day

103,000 tons daily can be delivered from origins (divisions 8 and 10 in USSR) to destinations (divisions 1 and 4 in Poland, thence west or south)

The goal was to study the Soviet ability to transport things from the Asian side to European side

Networks & Flows : Motivation

Harris & Ross (1955) Fundamentals of a Method for Evaluating Rail Net Capacities

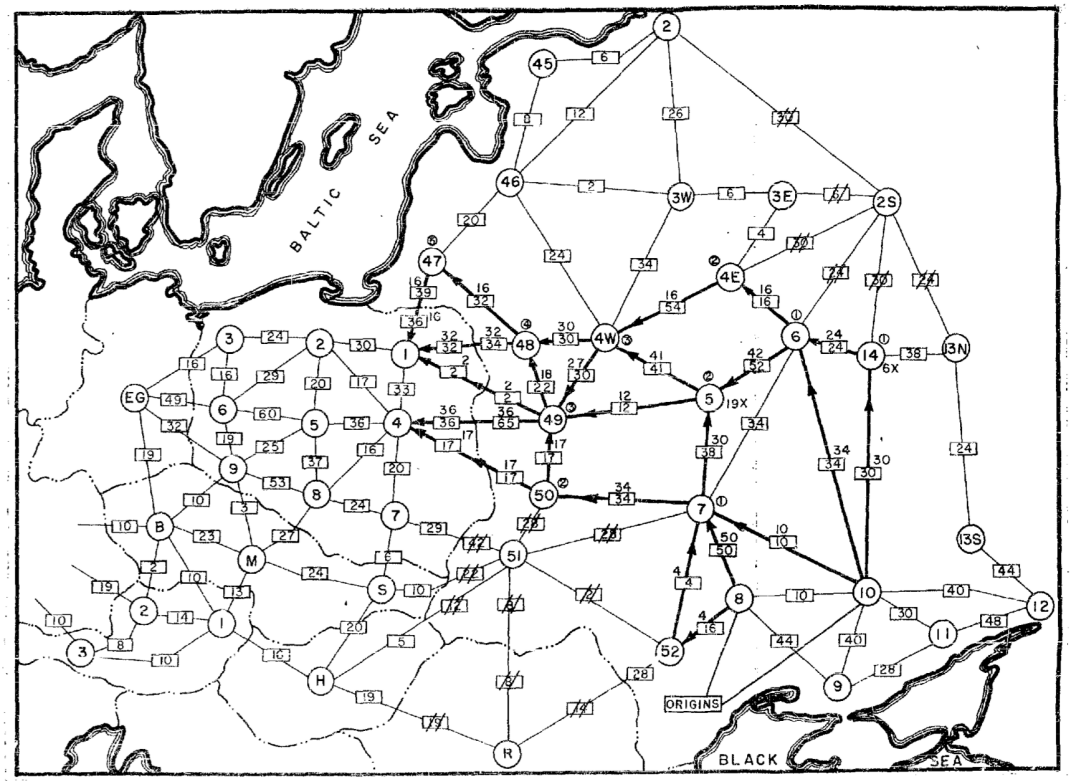


Fig. 13— Example: Step 4 and continuation

Legend:
International boundary
(B) Railway operating division
9 / 12 Capacity: 12 each way per day. Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction

All capacities in trains / 1000's of tons each way per day

Origins: Divisions 8 and 10 in USSR

Destinations: Divisions 1 and 4 in Poland, thence west or south

Also to study minimum cut

Network Flow

A Flow Network with capacities is $F = (G, c)$ where $G = (V, E)$ is a network, $c : E \to \mathbb{R}$ is a capacity function, on each edge.

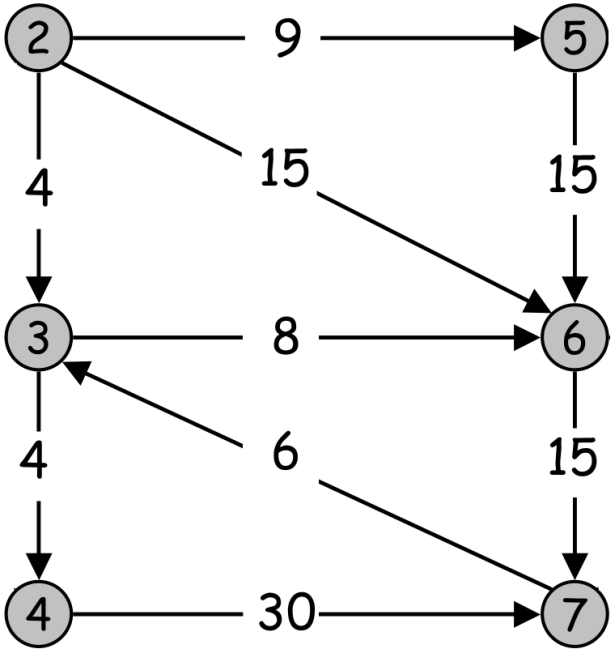
The absence of an edge can be seen equivalently as a 0 capacity: if $(u, v) \notin E$, we assume $c(u, v) = 0$.

Problem: it is not a cycle, water must come from somewhere (a source) and get out somewhere else (a sink).

The source node is supposed to have ressources, and the ink node is here insatiable.

For every vertice $v \in V$, there is a path from the source s to the sink t through v.

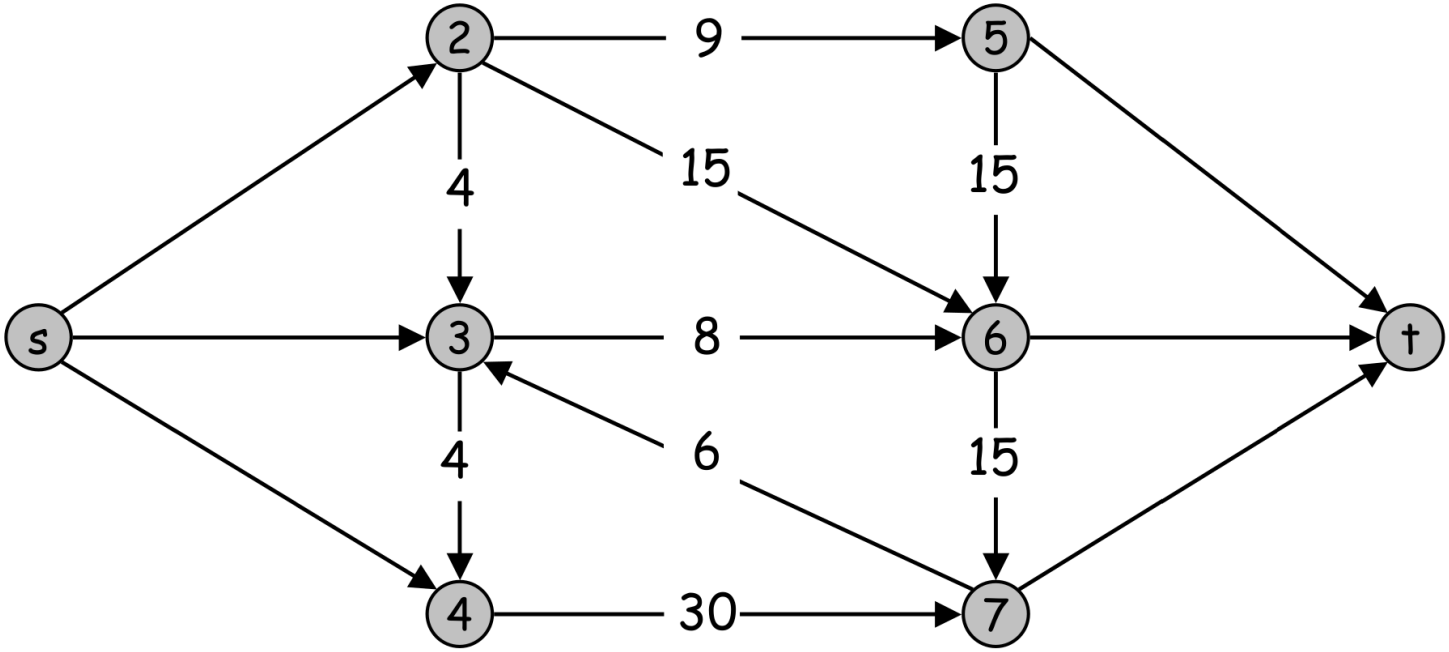
Network Flow



Transform the network by adding two vertices to V : a source s and a sink t.

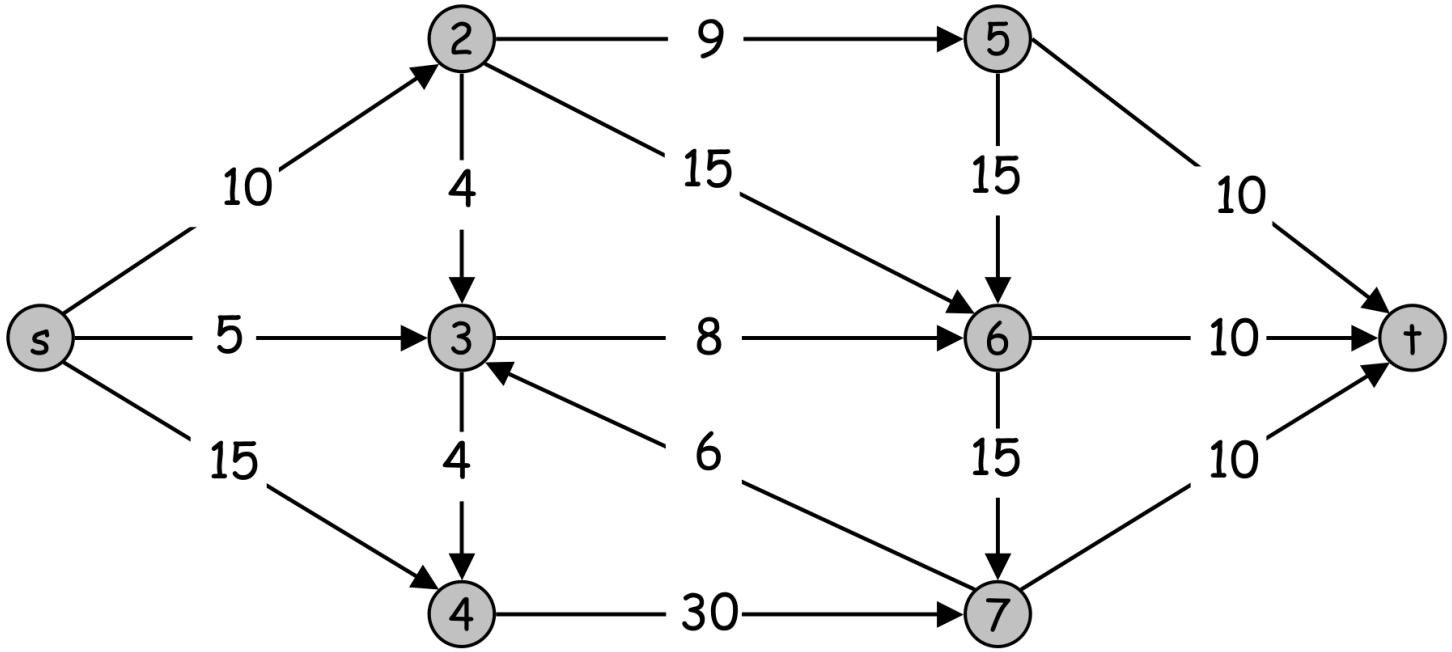
Network Flow

Then add edges to E, $e = (s, v)$ with $s \in \text{in}(v)$) and $e = (v, t)$ for $t \in \text{out}(v)$,



Network Flow

and set values c on those new edges.



Network Flow

A flow in G is a real-valued function $f : V \times V \mapsto \mathbb{R}$, that satisfies these three properties:

- Capacity constraint:

For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$

Flow from one vertex to another must not exceed given capacity.

- Skew symmetry (convention):

For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.

Flow from vertex u to vertex v is negative of flow in reverse direction.

- Flow conservation:

For all $u \in V \backslash \{s, t\}$, we require $\sum_{v \in V} f(u, v) = 0$.

Network Flow

The value of flow f is defined as the total flow leaving the source (and thus entering the sink):

$$|f| = \sum_{v \in V} f(s, v)$$

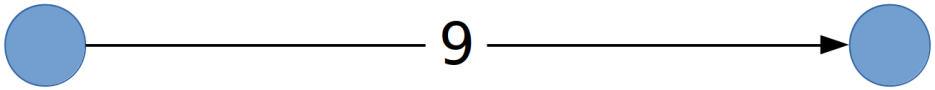
The total positive flow entering vertice $v \in V$ is $\sum_{u \in V, f(u,v)>0} f(u, v)$ and the total positive flow leaving vertice $u \in V$ is $\sum_{v \in V, f(u,v)>0} f(u, v)$

The total net flow at a vertice $v \in V$ is the total positive flow leaving v minus the total positive flow entering v

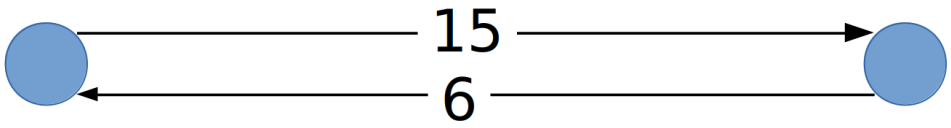
A flow f in flow network G is integer-valued if $f(u, v)$ is an integer for all $u, v \in V$.

Network Flow

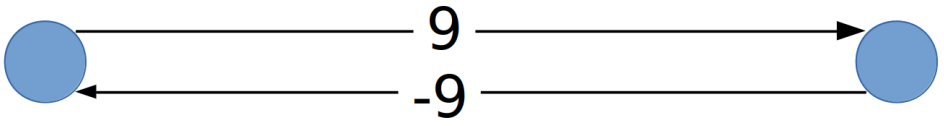
Consider two vertices $u, v \in V$, and a flow $f(u, v) = 9$ from u to v



One might think of having $f(u, v) = 15$ and $f(v, u) = 6$



but keep in mind the skew symmetry $f(u, v) = -f(v, u)$



Here the 6 from v to u are cancelled.

(cancellation will arise implicity in most algorithms).

Network Flow

For convenience, if $V_1, V_2 \subset V$, denote $f(V_1, V_2) = \sum_{u \in V_1, v \in V_2} f(u, v)$

Thus, flow conservation can be written $f(u, V) = 0$ for any $u \in V \backslash \{s, t\}$

Further $f(V_1, V_1) = 0$ (because of skew symmetry)

and $f(V_1, V_2) = -f(V_2, V_1) = 0$

Finally, consider $V_1, V_2, V_3 \subset V$ with $V_1 \cap C_2 = \emptyset$, then

$$f(V_1 \cup V_2, V_3) = f(V_1, V_3) + f(V_2, V_3).$$

$s-t$ Cuts

An $s-t$ cut is a partitionning of V between two blocks V_s and V_t, sur that $V_t = V \backslash V_s$, with $s \in V_s$ and $t \in V_t$

If f is a flow in $G = (V_1 \cup V_2, E)$, then the net flow across the cut (V_s, V_t) is defined to be $f(V_s, V_t)$, and its capacity is capacity $c(V_s, V_t)$.

Note that it can include negative flows.

A minimum cut is a cut with minimum capacity over all cuts.

Proposition Let f be a flow in a flow network G with source s and sink t, let (V_s, V_t) be a cut of G. Then the flow across (V_s, V_t) is $f(V_s, V_t) = |f|$.

Proof $f(V_s, V_t) = f(V_s, V) - f(V_s, V_s) = f(V_s, V) = f(s, V) + f(V_s \backslash \{s\}, V) = f(s, V) = |f|$.

$s-t$ Cuts

Proposition The value of a flow f in a network G is (upper-)bounded by the capacity of any cut (V_s, V_t) in G.

Proof $|f| = f(V_s, V_t) = \sum_{u \in V_s} \sum_{v \in V_t} f(u,v) \leq \sum_{u \in V_s} \sum_{v \in V_t} c(u,v) = c(V_s, V_t)$

The minimum cut problem is to find an $s-t$ cut with minimum capacity.

Example : on each edge e, write "$f(e)/c(e)$"

$s-t$ Cuts

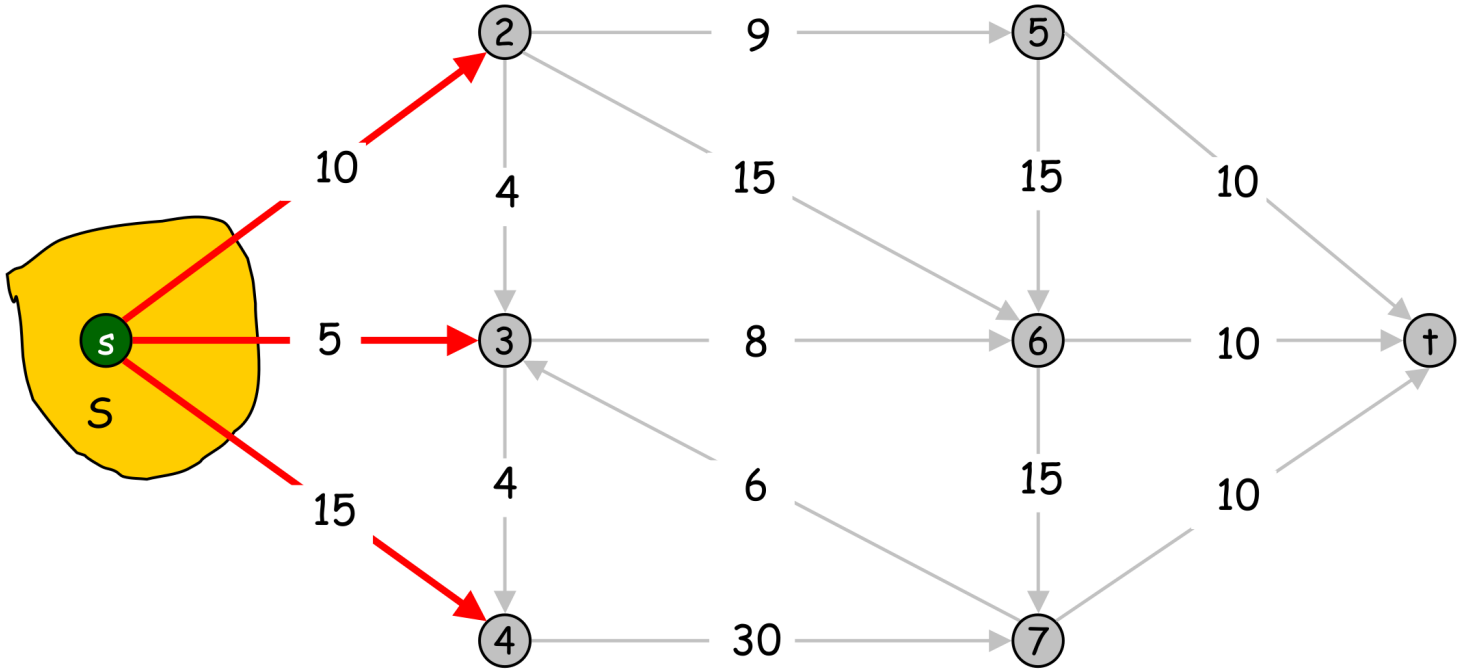
Flow Value Theorem : Let f be a flow, and (V_1, V_2) denote any $s-t$ cut, then the net flow accross the cut is equal to the flow value

$$\sum_{e \in \text{out of}(V_1)} f(e) - \sum_{e \in \text{into}(V_1)} f(e) = \sum_{v \in V} f(\{s, v\}) = |f|$$

As a consequence, $|f| \leq \sum_{e \in \text{out of} V_1} f(e)$, i.e. the flow value is at most the capacity of a cut.

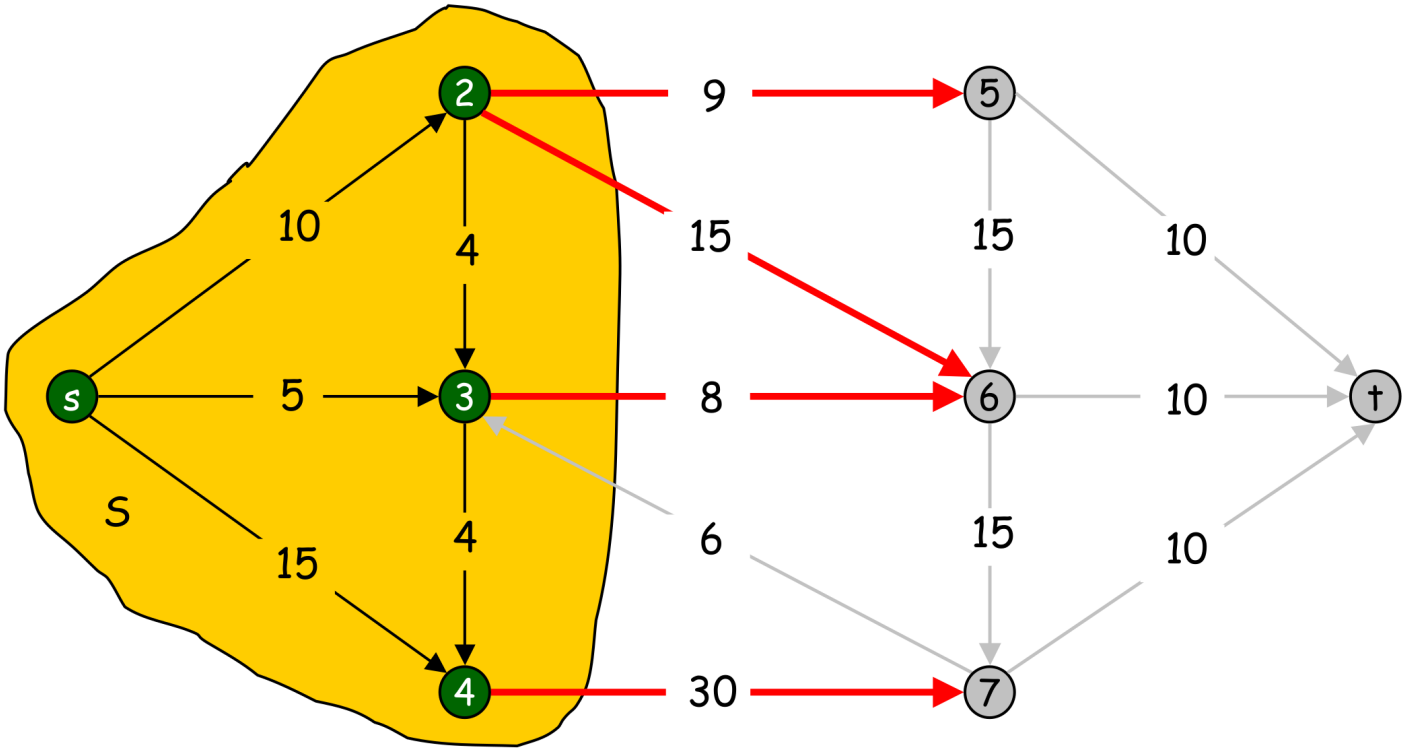
$s-t$ Cuts

A cut is a vertice partition $V = V_s \cup V_t$ with $s \in V_s$ and $t \in V_t$. Its capacity is $\text{cap}(V_s, V_t) = \sum_{u \in V_s, v \in \text{out}(u)} c((u,v))$ (below, V_s is denoted S).



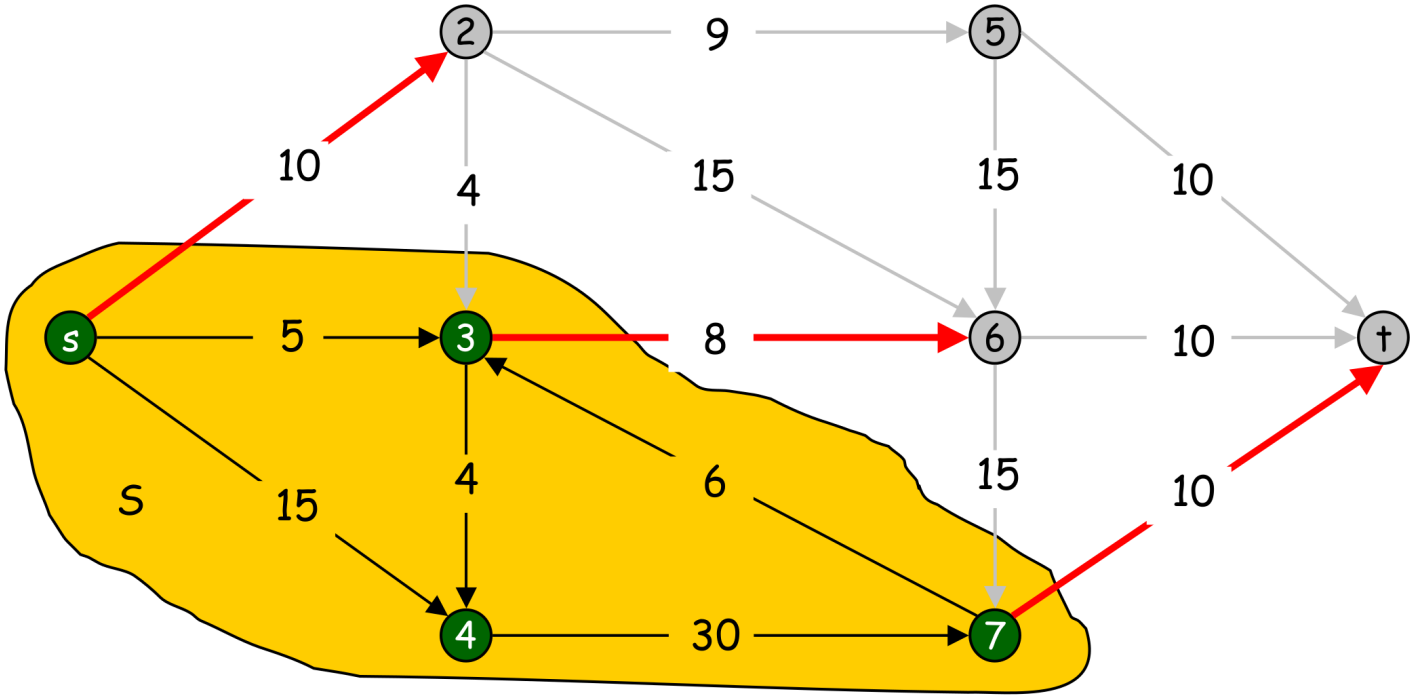
Capacity $= 10 + 5 + 15 = 30$

$s-t$ Cuts



$$\text{Capacity} = 30 + 8 + 15 + 9 = 62$$

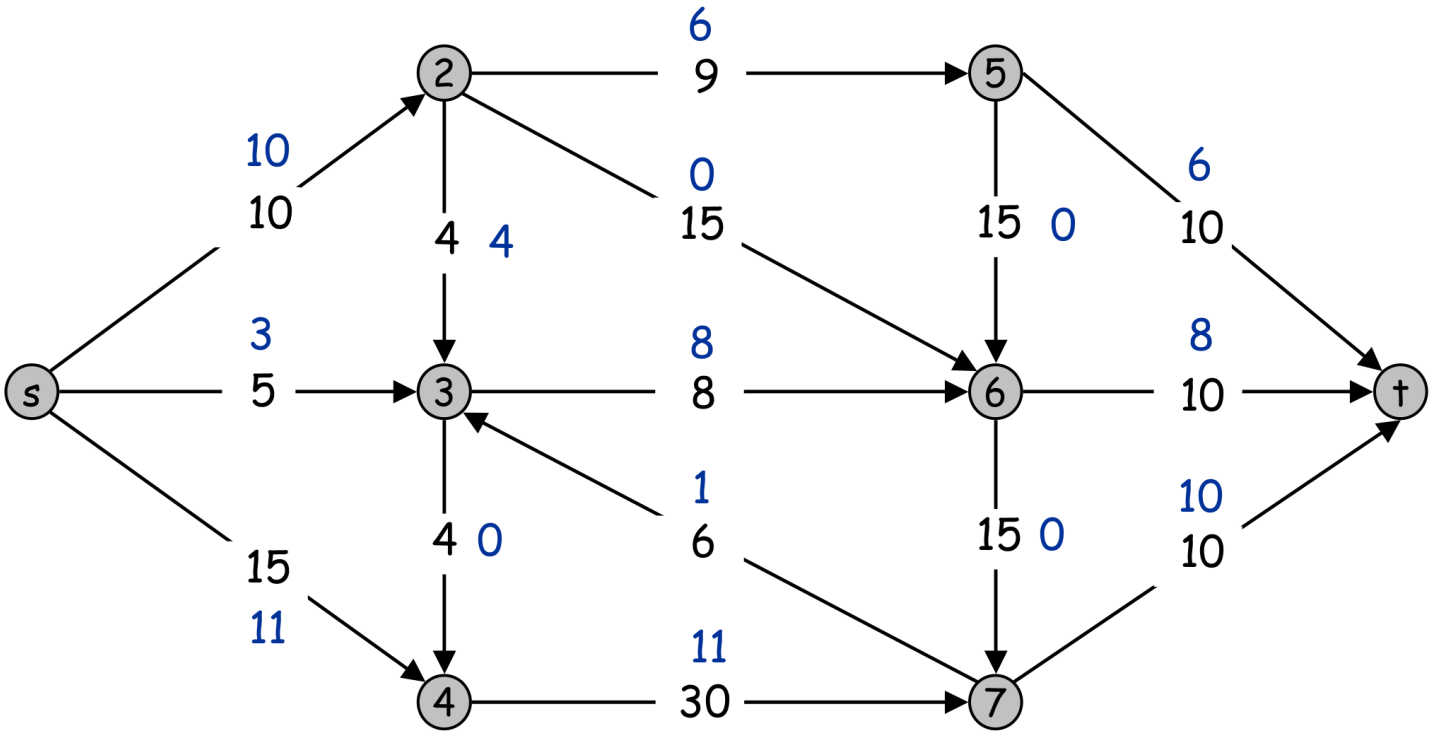
$s-t$ Cuts



$$\text{Capacity} = 10 + 5 + 15 = 30$$

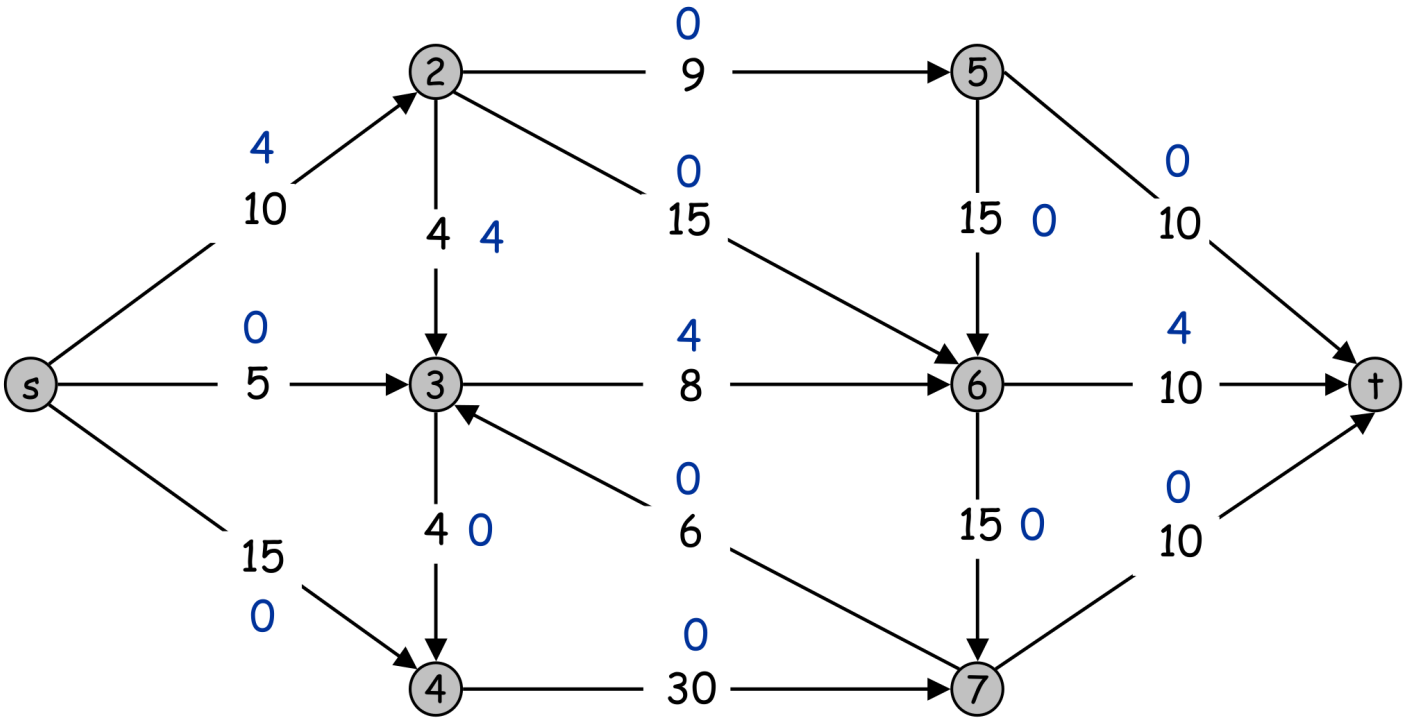
Flows

Keep in mind that we represent $f(e)$ and $c(e)$ on the same edge.

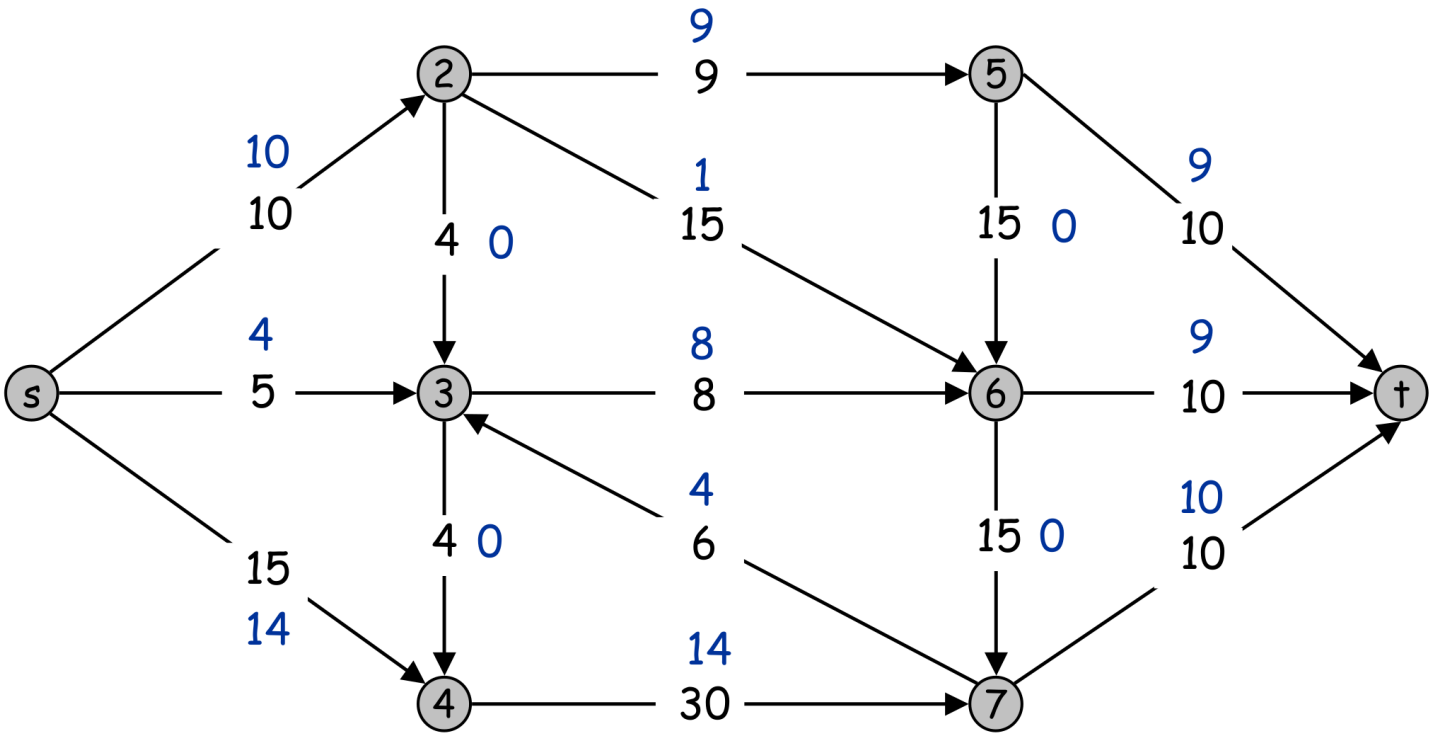


Flows

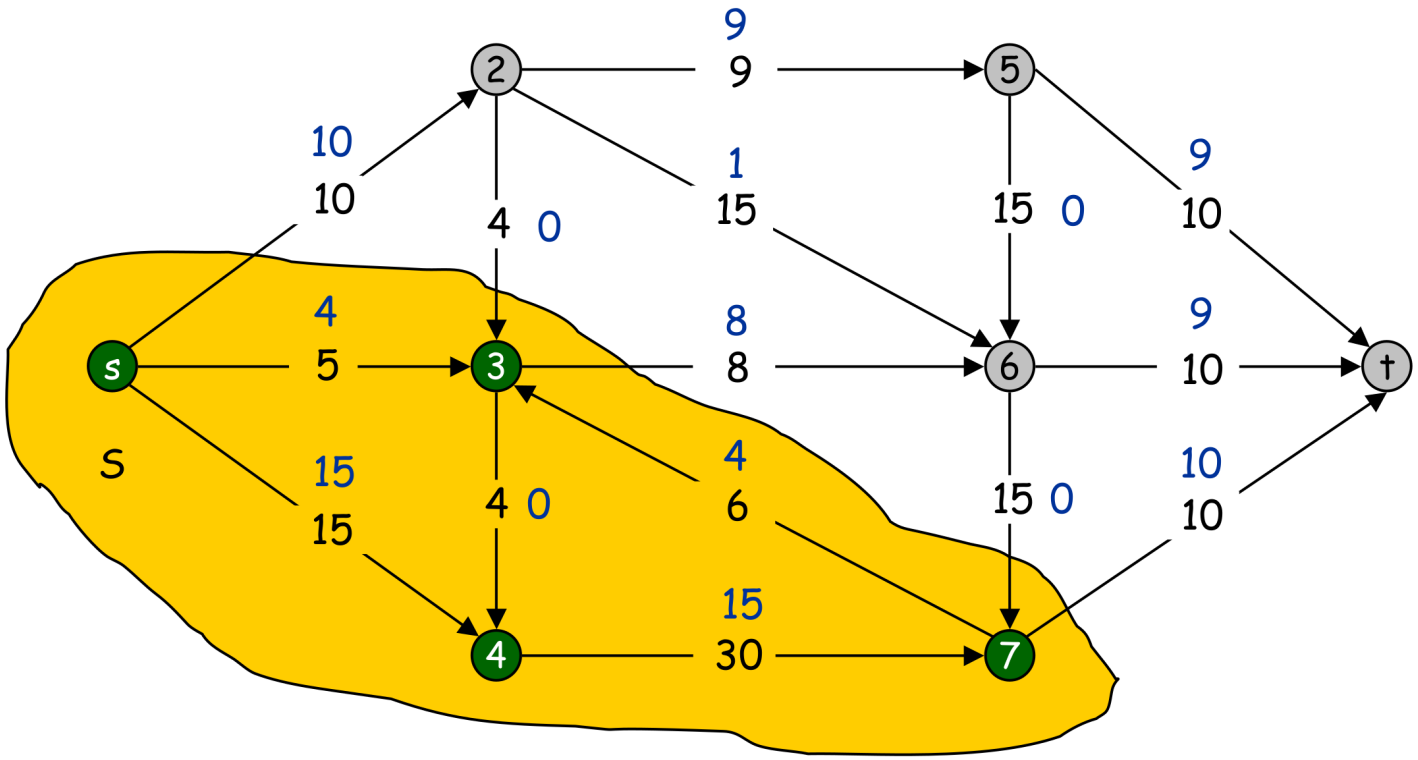
Below another feasable flow



Flows


9
9
2
5
10
10
1
15
4 0
15 0
9
10
4
5
s
3
8
8
6
9
10
t
4
6
4 0
15 0
10
10
15
14
14
30
4
7

Flows & Cuts

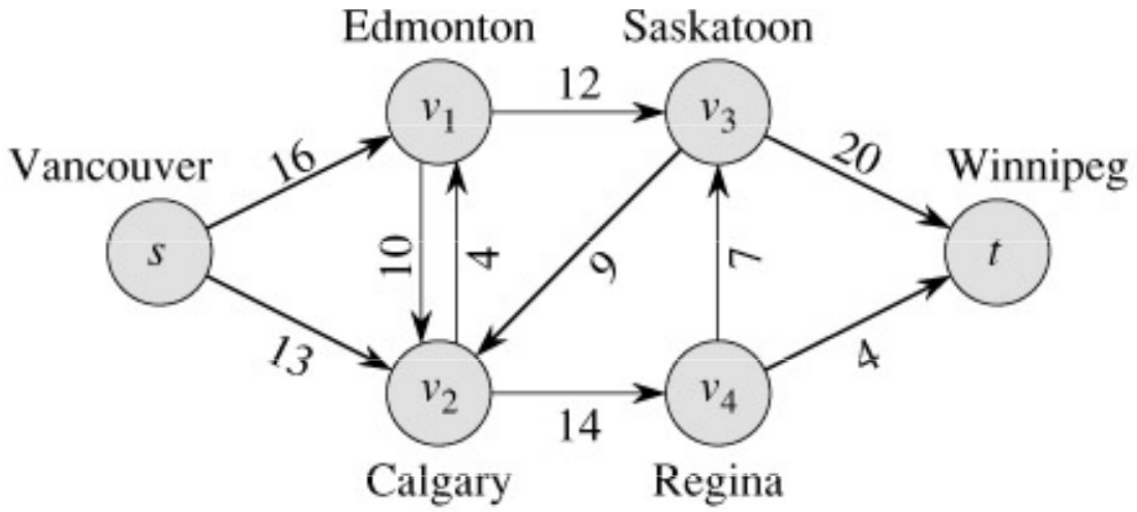

9
9
2
5
10
10
1
15
4 0
15 0
9
10
4
5
s
3
8
8
6
9
10
t
S
15
15
4 0
4
6
15 0
10
10
15
30
4
7

Electricity Flow, an Example

See electrical power distribution in Canada

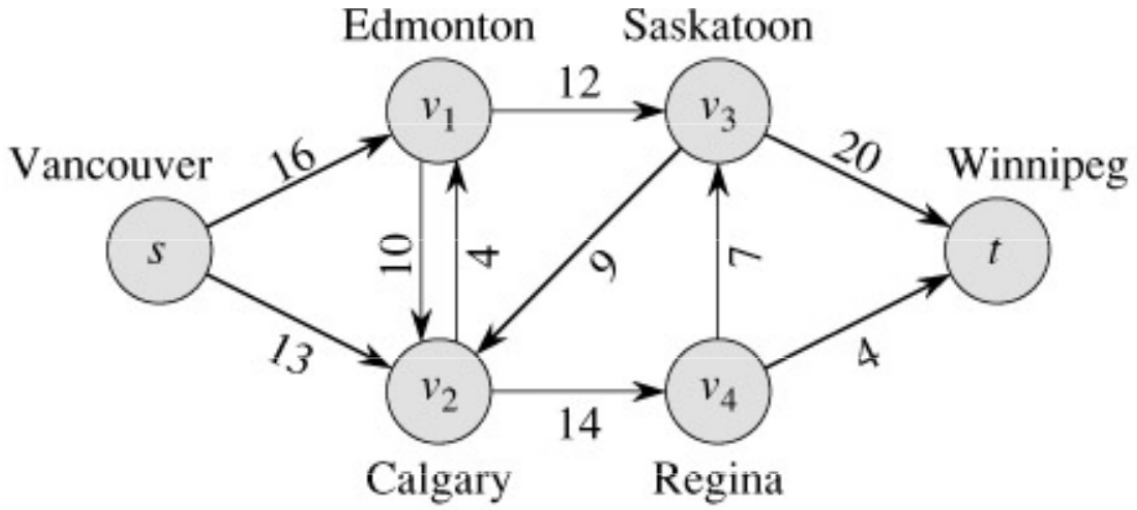
Power is generated in Vancouver (the source s) and all remaining power is transported to Winnipeg (the sink t).

Each high voltage cable has limited capacity (in MegaWatts).



Electricity Flow, an Example

There a two cables between Edmonton and Calgary, and we care about the net flow, Edmonton $(v_1) \to$ Calgary (v_2).



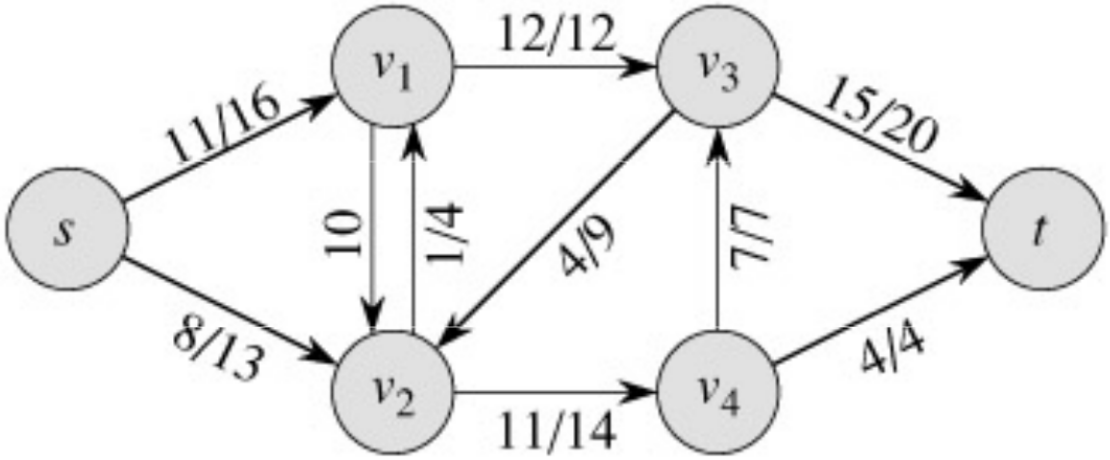
Electricity Flow, an Example

What could be the/a maximum flow $f^\star$?

The flow out of s is lower than the capacity out, i.e. $|f^\star| \leq 13 + 16 = 29$

The flow into t is lower than the capacity into, i.e. $|f^\star| \leq 20 + 4 = 24$

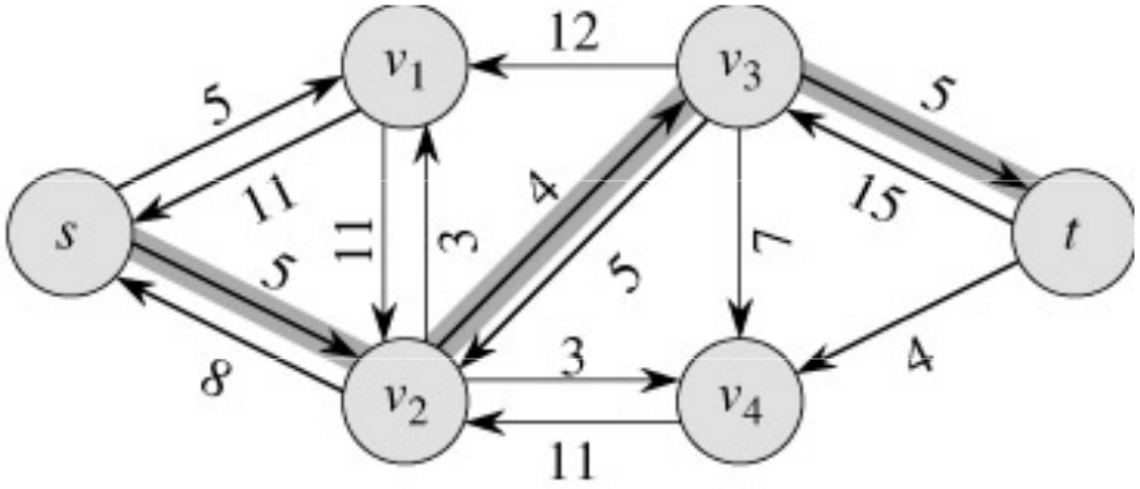
Is the maximum flow $|f^\star| = 24$? One can start with the feasible flow, for instance



i.e. $|f| = 19...$ can we go up to 24 ?

Electricity Flow, an Example

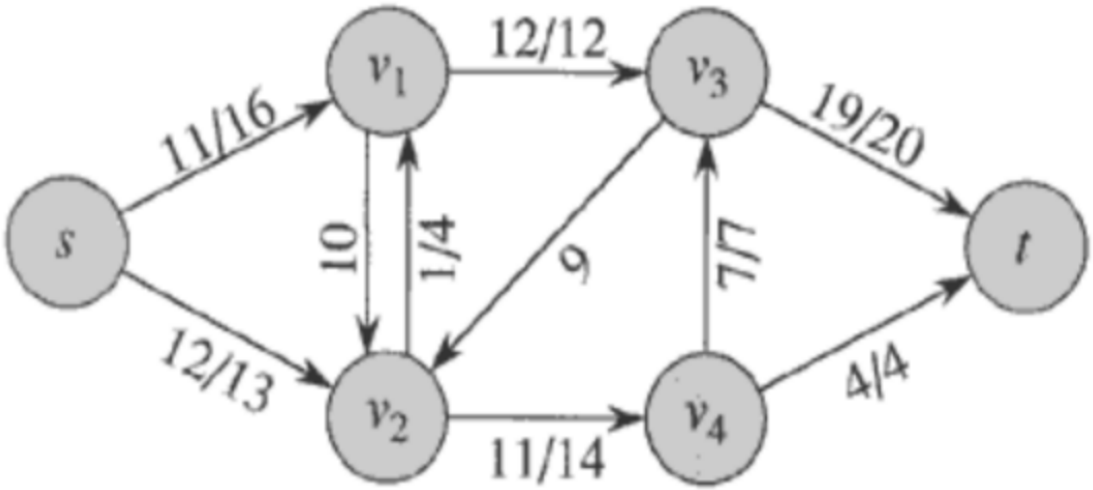
Define a so-called residual network and find an augmenting path from s to t.



The residual capacity of this path is 4.

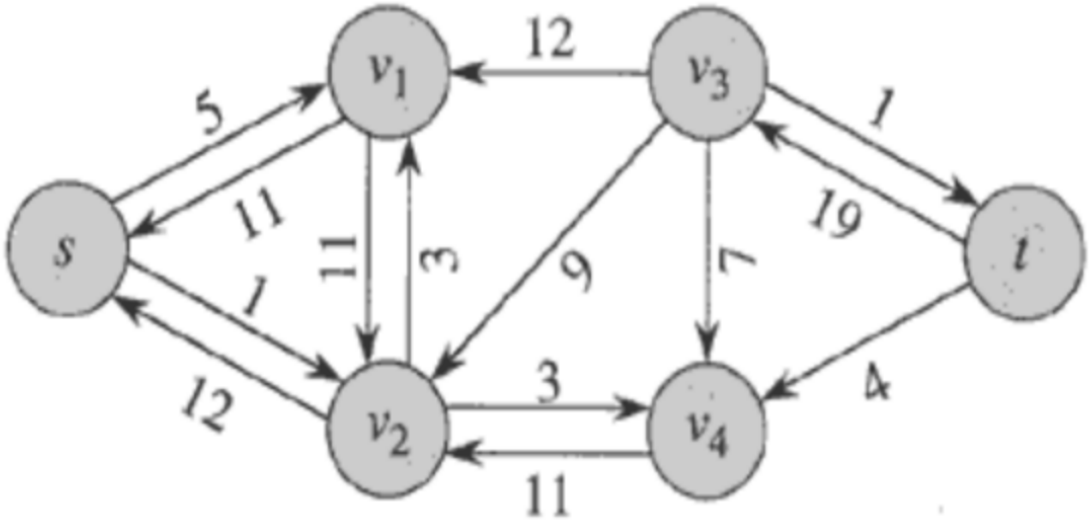
Electricity Flow, an Example

The new flow is the following



Electricity Flow, an Example

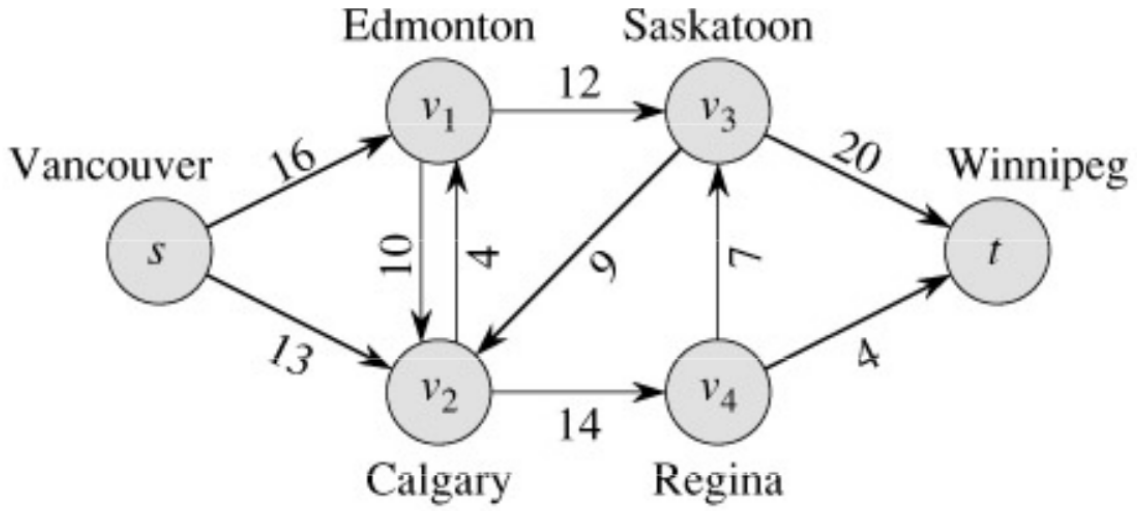
The associated residual network is the following



There are no augmenting path...

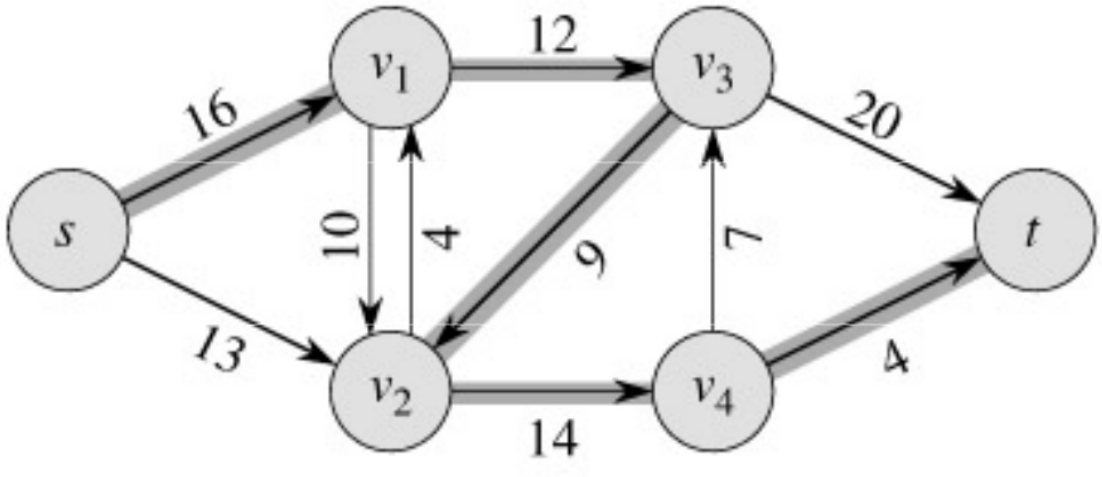
Electricity Flow, an Example

Let us start again with the same network, with capacities



Electricity Flow, an Example

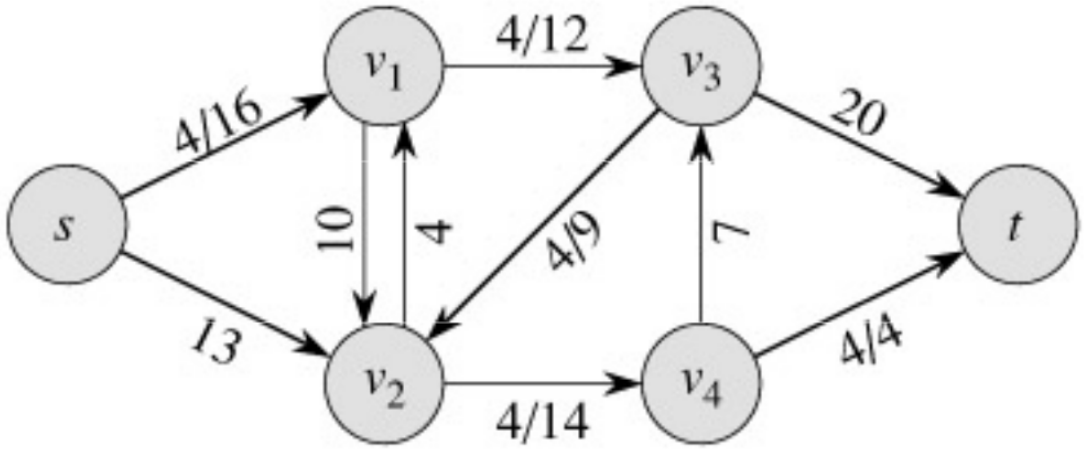
We can start with a path, from s to t



The maximal capactity of this flow is 4

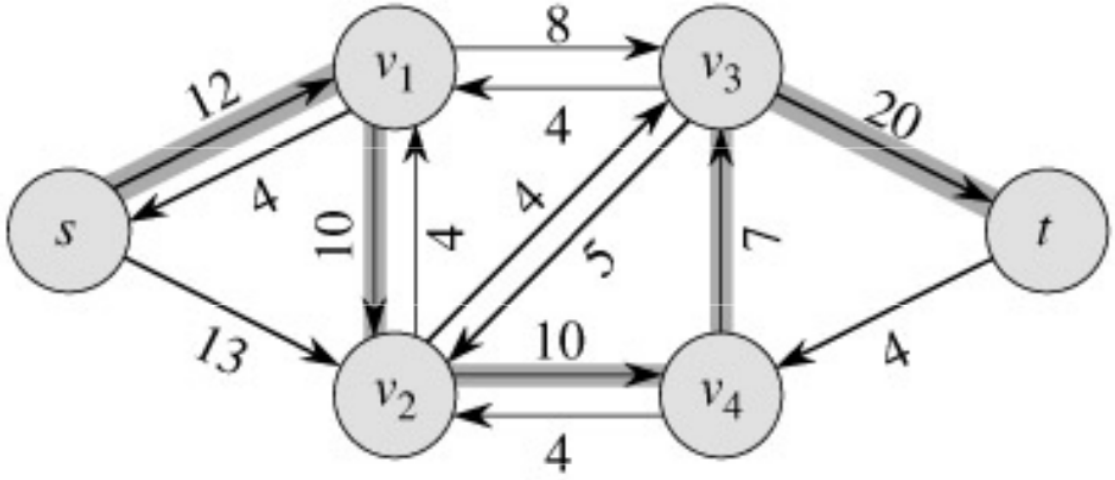
Electricity Flow, an Example

The flow network f is here, and $|f| = 4$



Electricity Flow, an Example

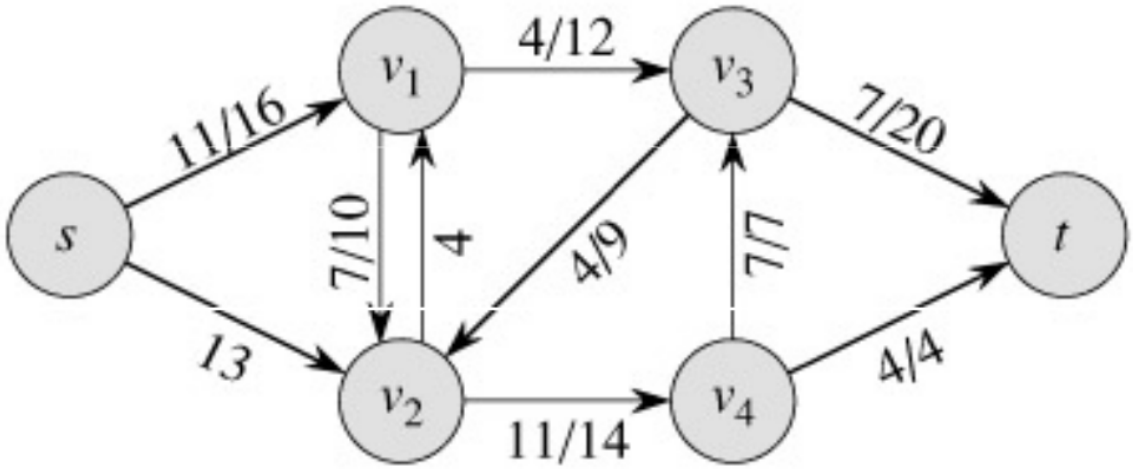
Consider the residual network, and pick an augmenting path from s to t



The maximal capactity of this flow is 7.

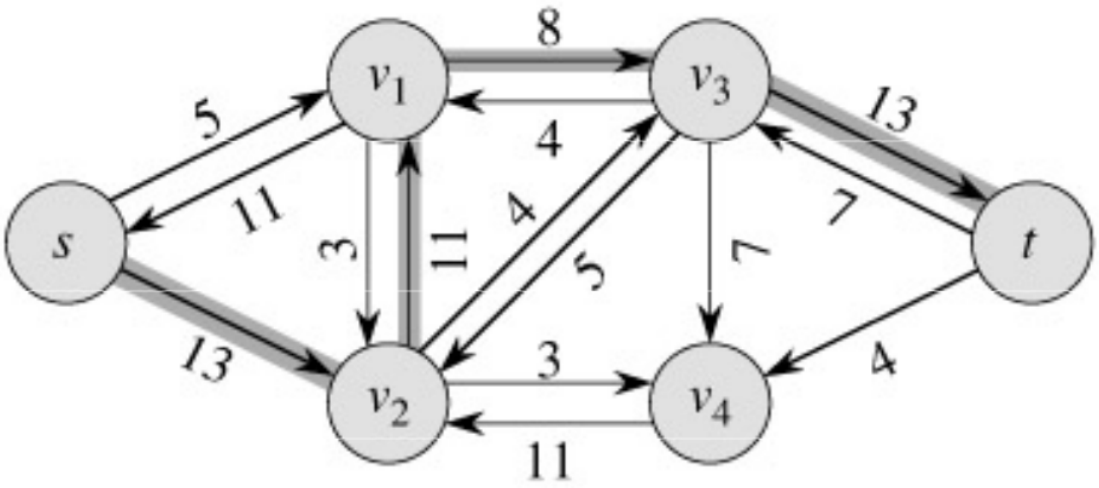
Electricity Flow, an Example

The flow network f is here, and $|f| = 4 + 7 = 11$



Electricity Flow, an Example

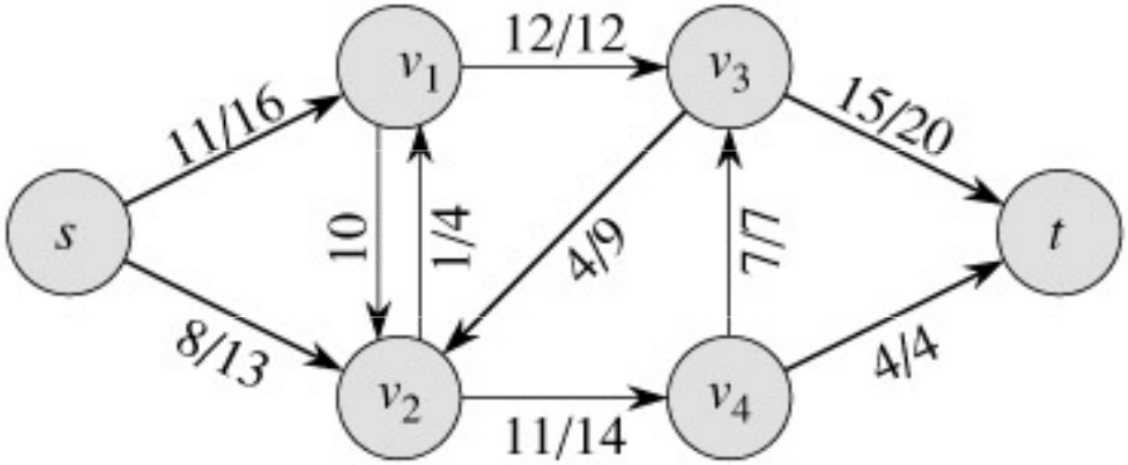
Consider the residual network, and pick an augmenting path from s to t



The maximal capactity of this flow is 8.

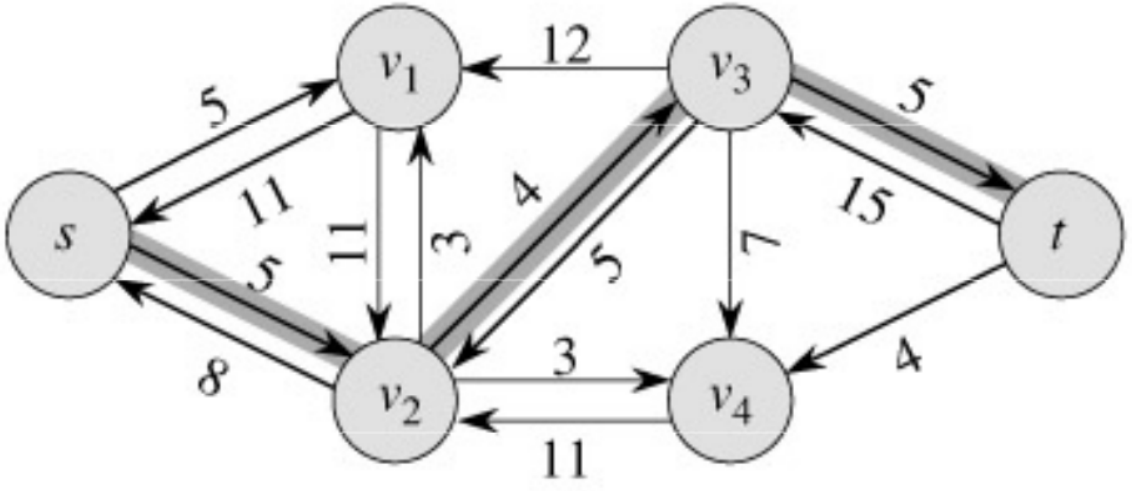
Electricity Flow, an Example

The flow network f is below, and $|f| = 4 + 7 + 8 = 19$



Electricity Flow, an Example

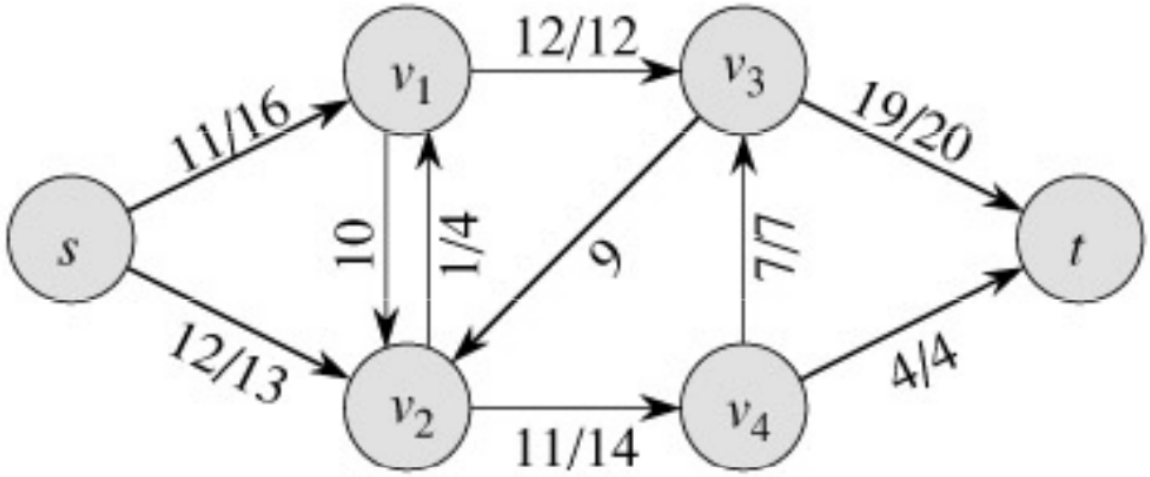
Consider the residual network, and pick an augmenting path from s to t



The maximal capactity of this flow is 4.

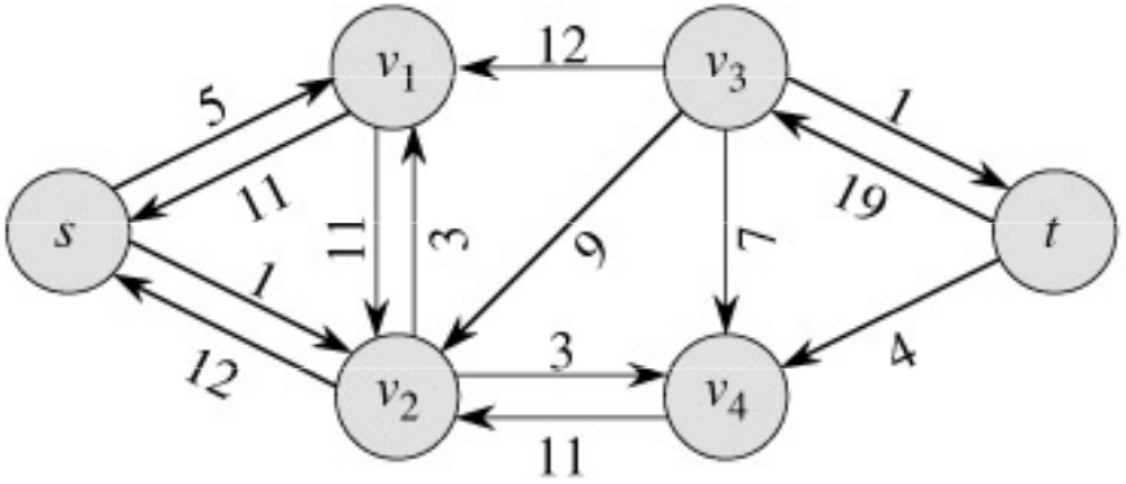
Electricity Flow, an Example

The flow network f is below, and $|f| = 4 + 7 + 8 + 4 = 23$



Electricity Flow, an Example

Consider the residual network, and there is no augmenting path from s to t



Max-flow min-cut

Max-flow min-cut theorem: If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

- f is a maximum flow in G
- residual network G_f contains no augmenting path
- there is a cut (V_s, V_t) such that $c(V_s, V_t) = |f|$

How to compute the maximum flow, in practice ?

Ford-Fulkerson

A first idea is the Ford-Fulkerson method, from Flows in Networks

- start with some feasible flow
- get the residual flow network (a graph that shows where extra capacity is available)
- augment paths (where extra capacity is available)
- look also at cuts, that provide an upper bound for the maximum flow

The residual flow network is the network flow $F = ((V, E), c_f)$ where $c_f(e) = c(e) - f(e), \forall e \in E$.

Ford-Fulkerson

Define the residual capacity between u and v, with respect to some flow f as

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) \text{ if } (u,v) \in E \\ f(v,u) \text{ if } (v,u) \in E \\ 0 \text{ otherwise} \end{cases}$$

because the flow either goes from u to v, or from v to u.

The residual network of f has the same vertices as G, but u and v are connected (edge $u \to v$) if $c_f(u,v) > 0$, and the weight of that edge is $c_f(u,v)$.

We can actually restrict ourself to E_f where $E_f = \{e \in E, c_f(e) > 0\}$.

Residual Capacity

The residual capacity $c_p(f)$ of an augmenting path p is the maximum additional flow we can allow along that path,

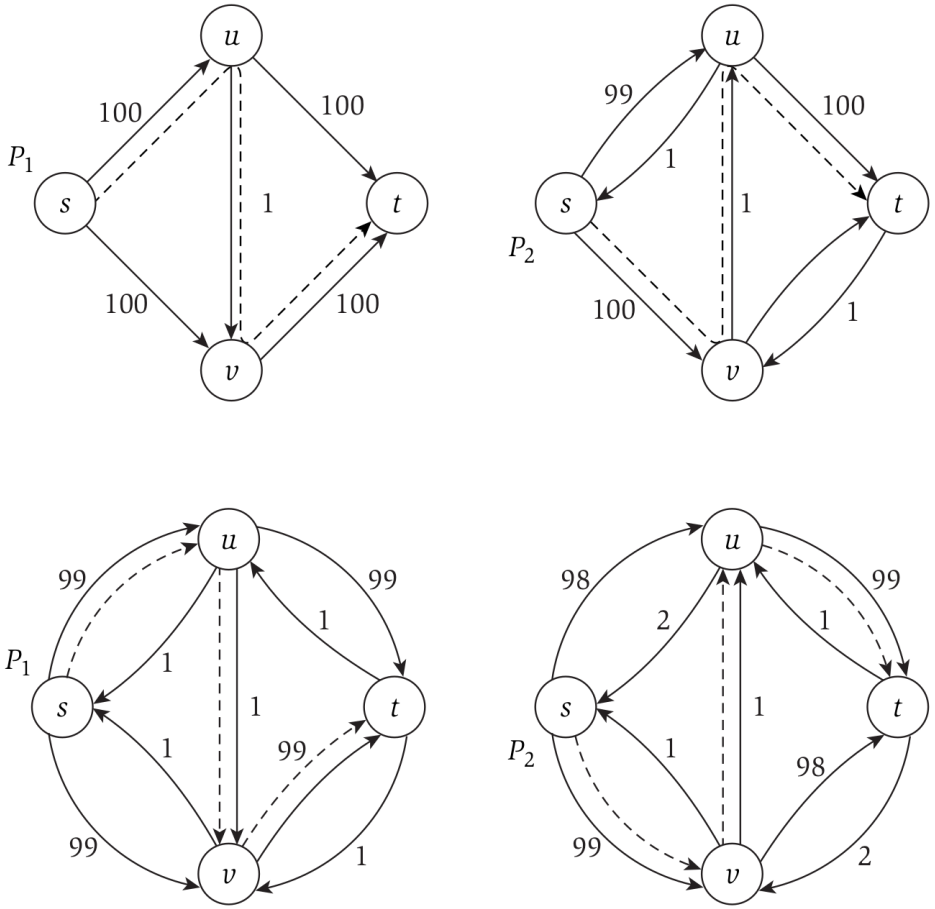
$$c_p(f) = \min\{c_f(e), e \in \text{ path } p\}.$$

The Ford-Fulkerson algorithm is the following. Consider a network G with nodes s, t and capacity c.

- for each edge $(u, v) \in E$, set $f(u, v) = f(v, u) = 0$
- while there is a path p from s to t in the residual network G_f do
- set $c_f(p) = \min\{c_f(u, v) \text{ for } (u, v) \in p\}$
- for each edge $(u, v) \in p$ do
- $f(u, v) = f(u, v) + c_f(p)$ while $f(v, u) = -f(u, v)$

Ford-Fulkerson

If chosen poorly, algorithm can be extremely long, see



Edmonds-Karp to improve Ford-Fulkerson

The Ford-Fulkerson algorithm is a greedy algorithm that computes the maximum flow in a flow network, also called Edmonds–Karp algorithm.

See Ford & Fulkerson (1962) Flows in Networks

The Ford–Fulkerson algorithm can be implemented to solve the bipartite matching problem in $O(mn)$ time.

The Hopcroft–Karp 1973 algorithm solves in $O(mn^{1/2})$ time, as the Even–Tarjan 1975, see also wikipedia

The Mądry 2013 solves in $O(m^{10/7})$ time.

For non-bipartite graph matching, Blossom algorithm from (Edmonds 1965) is : $O(n^4)$ but the Micali–Vazirani 1980 algorithm solves in in $O(mn^{1/2})$ time (see wikipedia)

Linear Programming

A linear program is a collection of linear constraints on some real-valued variables, with come linear objective function.

A polytope is an n-dimensional body, with flat faces

A polytope is a convex polytope if for any point in the polytope, the line segment joining the two points lies within the polytope.

The feaseable region of a linear program is a convex polytope.

An extremal point is a corner point of the feasible region. Note that it cannot be expressed as the convex co,bination of two (or more) in the polytope

A general program is either

$$\begin{cases} \max\limits_{\boldsymbol{x}\in\mathbb{R}_+^d} \{\boldsymbol{c}^\top\boldsymbol{x}\} \\ \text{s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \end{cases}$$

for some $k \times d$ matrix $\boldsymbol{A}$,

Linear Programming

or

$$\begin{cases} \min\limits_{\boldsymbol{x}\in\mathbb{R}_+^d} \{\boldsymbol{c}^\top\boldsymbol{x}\} \\ \text{s.t. } \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \end{cases}$$

There are three possible cases

- the feasible region is empty: there are no optimal values
- the feasible region is unbounded: the optimal value is unbounded
- the feasible region is a bounded convex polytope: an optimal value exist

Linear Programming

Maximal Flow problems can be expressed as a Linear Programming problem

$$\begin{cases} \max_E \left\{ \sum_{(u,v)\in E} f(u,v) \right\} \\ \text{s.t.} \sum_{(u,v)\in E} f(u,v) = \sum_{(u,v)\in E} f(v,u) \ \forall u,v \in V \ \text{(conservation)} \\ f(u,v) \leq c(u,v) \ \forall (u,v) \in E \ \text{(capacity constraint)} \\ f(u,v) \geq 0 \ \forall (u,v) \in E \end{cases}$$

Simplex & Interior Point Methods

Simplex Method (see wikipedia)

Consider a standard linear program

$$\max_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\mathsf{T} \cdot \boldsymbol{x} \right\} \text{ subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{x} \geq \boldsymbol{0}$$

The linear program can be represented as a tableau of the form

$$\begin{bmatrix} 1 & -\boldsymbol{c}^\mathsf{T} & 0 \\ 0 & \boldsymbol{A} & \boldsymbol{b} \end{bmatrix}$$

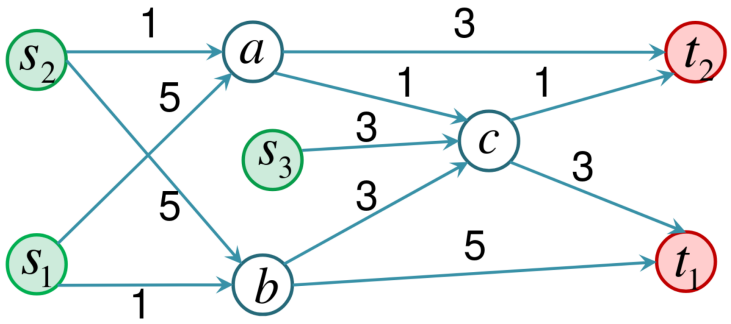
Interior Point Method (see wikipedia)

In contrast to the simplex algorithm (which finds an optimal solution by traversing the edges between vertices on a polyhedral set) interior-point methods move through the interior of the feasible region.

Networks & Flows : Extensions

Let us get back to the initial Soviet Railway example.

Note that there are several sources and several sinks.



How to find the maximum flow in the standard network ?

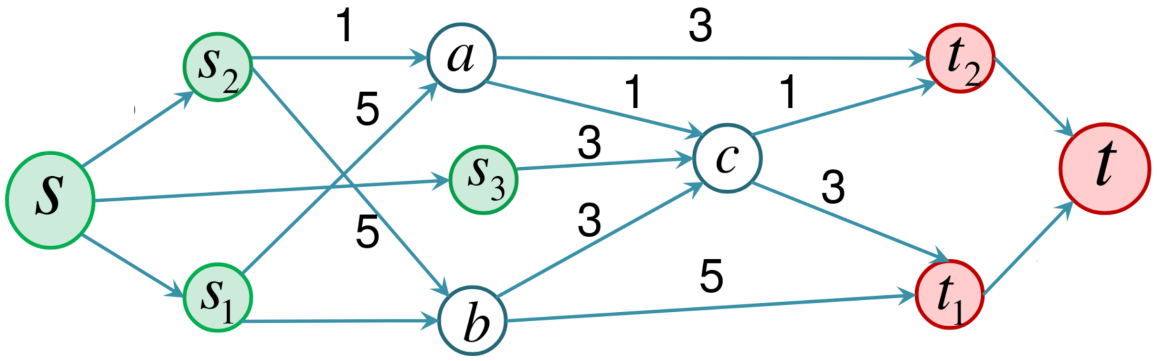
Consider a super source S that is connected to all sources, with infinite capacities

Consider a super sink T that is connected to all sinks, with infinite capacities

Networks & Flows : Extensions

Consider a super source S that is connected to all sources, with infinite capacities

Consider a super sink T that is connected to all sinks, with infinite capacities



Networks & Flows : Extensions

Other problems were mentioned in that example, like find edge-disjoint paths, or vertice-disjoint paths: what is the minimum number of train tracks needed to be destroyed to prevent any transportation from the Asian side to the European side?

Given a graph $G = (V, E)$, with a source s and a sink t, what is the maximum number of edge-disjoint paths from s to t ?

This can be solve easily with flow networks, defining a flow network with unit capacities, for every edges. It is a 0-1 network (here, maximum flow can be computed efficiently)

Networks & Flows : Extensions

Theorem Given a 0-1 flow network with a maximum flow $f^\star$ and maximum number of edge-disjoint paths $k^\star$, then $|f^\star| = k^\star$.

Proof: (1) By having a flow of 1 through every disjoint path, we obtain a flow of size $k^\star$ so $|f^\star| \geq k^\star$.

(2) The converse is obtained by induction (starting with the case $|f^\star| = 0$)

Consider a maximum flow (of size m).

Remove every edge with 0 flow through it.

Find a path from s to t (it exists since $|f^\star| > 0$).

Remove the edges of the path, to obtain a network with a flow of size $m - 1$

By the induction hypothesis , there are $m - 1$ edge-disjoint paths in this network.

Bring back the removed path, to obtain at least m edge-disjoint paths

Networks & Flows : Extensions

Given a graph $G = (V, E)$, with a source s and a sink t, what is the minimum number of edges needed to be removed so that no path from s to t remains

Again, consider some $0 - 1$ network, and find a maximum flow in this network.

Theorem Given a 0-1 flow network with a maximum flow $f^\star$ and minimum number of edges needed to disconnect s from t $k^\star$, then $|f^\star| = k^\star$.

Proof We have seen that $f^\star$ was actually the maximum number of edge-disjoint paths

$|f^\star| \leq k^\star$ since there are $|f^\star|$ edge-disjoint paths, and we need to remove at least one edge from each

$|f^\star| \geq k^\star$ since the minum cut is a set of disconnecting edges. And by the max-flow/ min-cut theorem, there are $|f^\star|$ edges in the min-cut.

Menger Theorem (1927) the minimum number of disconnecting edges is equal to the maximum number of edge-disjoint paths

Optimal Matching

Consider a transport company, with three plants $\{A, B, C\}$ and three construction sites $\{L, M, N\}$

Plants can supply (capacity), per week $\{s_A, s_B, s_C\}$, $\{300, 300, 100\}$

Site can demand (requirement), per week $\{d_L, d_M, d_N\}$, $\{200, 200, 300\}$

and costs of transportation are given in the following matrix

	L	M	N
A	4	3	8
B	7	5	9
C	4	5	5

Let $x(i, j)$ denote the number of units to be transported from source i to destination j

Optimal Matching

Let s_i denote the supply and d_j denote the demand, and $c(i,j)$ the cost per unit, from source i to destination j

Here total supply = total demand, i.e. $\sum_i s_i = \sum_j d_j$

Hence, it is called a balanced problem.

Optimisation program is

$$\min\left\{\sum_i\sum_j x(i,j)c(i,j)\right\}$$

subject to

$$\sum_j x(i,j) = s_i, \forall i,\ \sum_i x(i,j) = d_j, \forall j,\ x(i,j) \geq 0, \forall i,j.$$

Optimal Matching

Initial solution - **Northwest Corner Rule**

1. Start by selecting the cell in the most "North-West" corner of the table.
2. Assign the maximum amount to this cell that is allowable based on the requirements and the capacity constraints.
3. Exhaust the capacity from each row before moving down to another row.
4. Exhaust the requirement from each column before moving right to another column.
5. Check to make sure that the capacity and requirements are met

Optimal Matching

Stepping Stone Method

1. Select an unused square to be evaluated.

2. Beginning at this square, trace a closed path back to the original square via squares that are currently being used (only horizontal or vertical moves allowed). You can only change directions at occupied cells.

3. Beginning with a plus (+) sign at the unused square, place alternative minus (-) signs and plus signs on each corner square of the closed path just traced.

4. Calculate an improvement index, I_ij by adding together the unit cost figures found in each square containing a plus sign and then subtracting the unit costs in each square containing a minus sign.

5. Repeat steps 1 to 4 until an improvement index has been calculated for all unused squares.

Optimal Matching

• If all indices computed are greater than or equal to zero, an optimal solution has been reached.

• If not, it is possible to improve the current solution and decrease total shipping costs.

Optimal Matching with Ranks

A matching is stable if no unmatched man and woman each prefers the other to his or her spouse.

The problem is to find a stable matching for any dating pool.

Iterative Process

(0) each individual ranks the opposite sex

(1) each man proposes to his top choice

each woman rejects all but her top suitor

(k) each man rejected on day $(k-1)$ proposes to his top remaining choice

each woman rejects all but her top remaining suitor

Optimal Matching with Ranks

It is a deferred-acceptance algorithm.

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

Run the algorithm

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

[(1.1)] $\{M, N\} \mapsto A$ and $L \mapsto B$

Optimal Matching with Ranks

Run the algorithm

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

$[(1.2)]$ $A \mapsto M$ and $B \mapsto L$: couples (A, M) and (B, L) are engaged

Run the algorithm

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

$[(2.1)]$ $N \mapsto B$

Optimal Matching with Ranks

Run the algorithm

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

[(2.2)] $B \mapsto N$: couples (A, M) and (B, N) are engaged

Run the algorithm

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

[(3.1)] $L \mapsto C$

Optimal Matching with Ranks

Run the algorithm

Ana	Luke	Matthew	Norman
Beatrice	Matthew	Norman	Luke
Clara	Norman	Matthew	Luke

Luke	Beatrice	Clara	Ana
Matthew	Ana	Clara	Beatrice
Norman	Ana	Beatrice	Clara

[(3.2)] $C \mapsto L$: couples (A, M), (B, N) and (C, L) are engaged

Optimal Matching with Ranks

Theorem The deferred-acceptance algorithm arranges stable marriages.

Heuristically, each of the women that a given man prefers to his wife rejected him in favor of a suitor she preferred.

Theorem (Gale-Shapley) In the solution found by the deferred-acceptance algorithm, every man gets his best possible match

Theorem (Conway) In the stable matching found by the male-proposing algorithm, every woman gets her worst possible match

Corrolary Waiting to receive proposals is a bad strategy.

Should people lie to improve the result ?

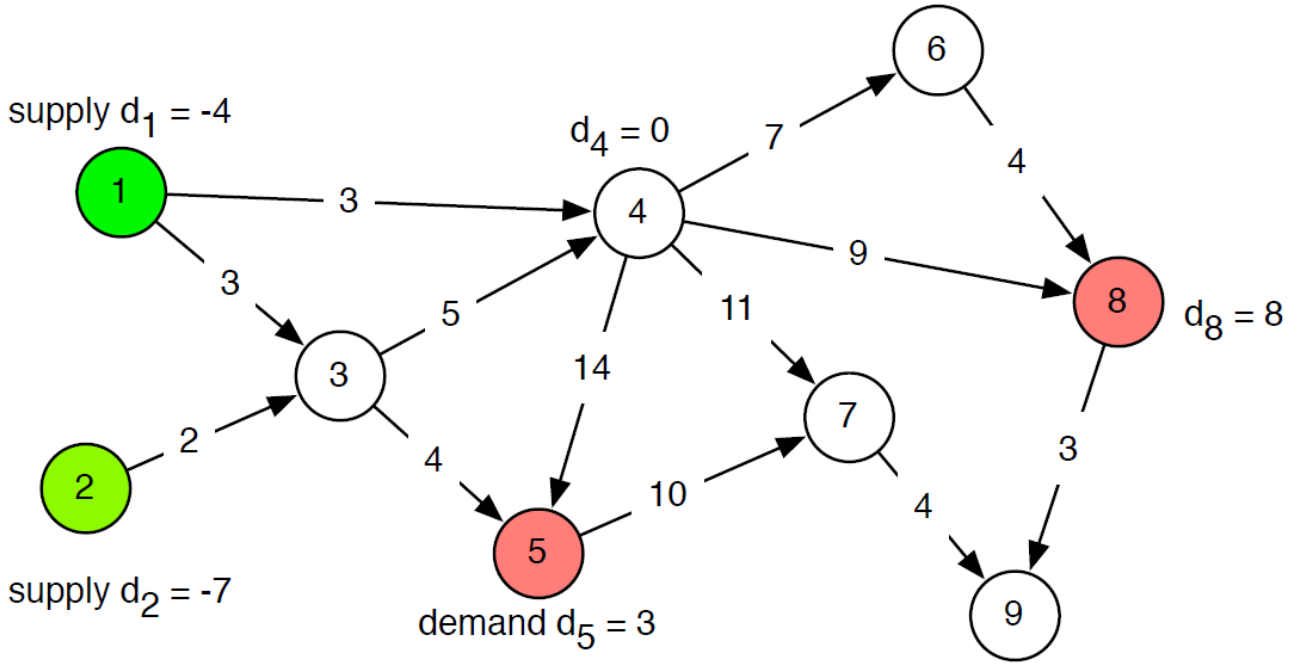
Theorem (Dubins-Freedman) No man (or group of men) can improve their results in the male-proposing algorithm by submitting false preferences.

Circulation with Supply and Demand

Flows with supply and demand corresponds to multiple sources and sinks

Each source has a certain amount of flow to give (its supply).

Each sink wants to get a certain amount of flow (its demand).



Circulation with Supply and Demand

We want to find a flow f such that $\forall e \in E$, $f(e) \in [0, c(e)]$

and for each $v \in V$, $\sum_{uin} f(u,v) - \sum_{uout} f(u,v) = d(v)$

The demand $d(v)$ is the excess flow that should come into node.

Let S be the set of nodes with negative demands (supply). Let T be the set of nodes with positive demands (demand).

In order for there to be a feasible ow, we must have $\sum_{s \in S} d(s) + \sum_{t \in T} d(t) = 0$

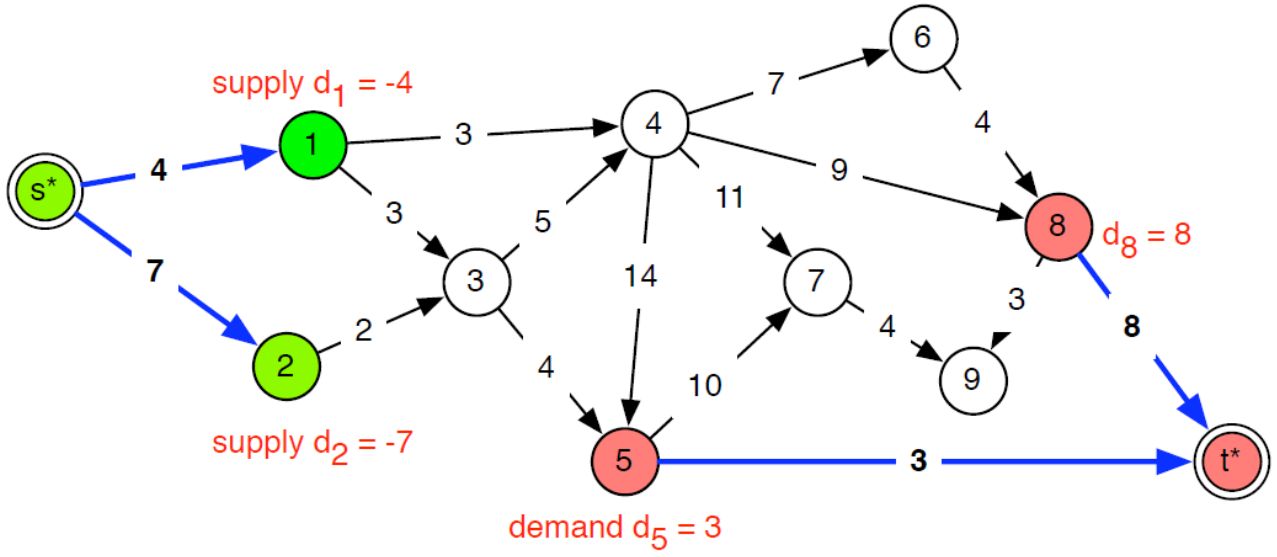
Define $D = \sum_{t \in T} d(t)$

Circulation with Supply and Demand

To get back to a maximum flow problem, introduce a super source $s^\star$ with edges to every vertice in S and a super sink $t^\star$ with edges to every vertice in T

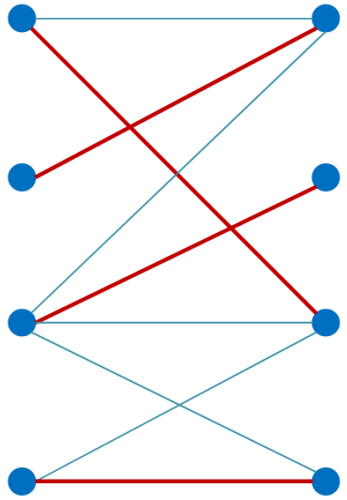
Capacity of edge $(s^\star, s)$ is $-d(s)$ while capacity of edge $(t, t^\star)$ is $d(t)$.

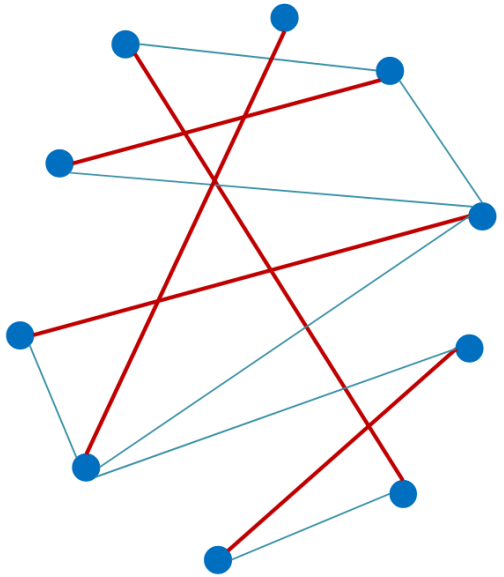
Now we can use nax-flow algorithms to find a flow f.



More on Matching

Consider a set of vertices V, a matching is a set of edges E that do not have a set of common vertices : a matching is a graph (V, E) where each node has either zero or one edge incident to it (see brilliant.org)





More on Matching

A maximal matching is a matching to which no more edges can be added without increasing the degree of one vertices to two (it is locally optimal)

A maximum matching is a matching with the largest possible number of edges (it is globally optimal)

A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets V_1 and V_2, with $V_1 \cap V_2 = \emptyset$, such that every edge connects a vertex in V_1 to one in V_2 (called A and B in the pictures).

Consider directed graphs, with edges from V_1 to V_2.

Assume further that all edges have unit capacity.

More on Matching

For a bipartite matching, with value k

- there are no node in V_1 which has more than one outgoing edge, where there is a flow
- there are no node in V_2 which has more than one incoming edge, where there is a flow
- the number of edges between V_1 and V_2 carrying flow is equal to k

A perfect matching is a matching in which each node has exactly one edge incident on it

König's theorem states that, in bipartite graphs, the maximum matching is equal in size to the minimum vertex cover.

Hall's marriage theorem provides a characterization of bipartite graphs which have a perfect matching and the Tutte theorem provides a characterization for arbitrary graphs.

Bipartite Matching (0/1 weights)

See the Task Assignment Problem

A set of tasks that need to be complete.

A set of people , each qualified to do a different subset of tasks.

Each person can perform at most one task. Each task is performed by at most one person.

Assign as many tasks as possible to people

Consider the bipartite graph, (V, E), where $V = V_1 \cup \text{Var}_2$, with V_1 is the set of people, V_2 is the set of tasks. And there is an edge between every person and every task that the person is qualified to do

A matching in an undirected graph is a set of vertice - disjoint edges.

The size of a matching is the number of edges in it.

A matching is possible on any kind of graph

A maximum matching of G is a matching of maximum size.

Bipartite Matching ($0/1$ weights)

How can we find such a matching?

Recall that on $G = (V_1 \cup V_2, E)$,

$$\sum_{v \in V_1} d(v) = \sum_{v \in V_2} d(v) = n_E = |E|,$$

since every edge of E contributes 1 to the sum of the degrees in each side

Simple case: assume that

- Each person is qualified to do exactly k of the jobs.
- Every job has exactly k people that are qualified for it

Bipartite Matching ($0/1$ weights)

Since each person is qualified to do exactly k of the tasks $\sum_{v\in V_1} d(v) = k|V_1|$

Since each task has exactly k people that are qualified to do it $\sum_{v\in V_2} d(v) = k|V_2|$

Thus $k|V_1| = k|V_2|$, i.e. necessarily $|V_1| = |V_2|$

A perfect matching of a graph $G = (V, E)$ is a matching of size $|V|/2$.

So, a perfect matching for a bipartite graph can be obtained only when both sides have the same size.

Consider a bipartite graph $(V_1 \cup V_2, E)$, and a vertice $v \in V_1$, then the neighbor set of v is

$$N(v) = \{u \in V_2 \text{ such that } (v, u) \in E\}.$$

and for a subset $A_1 \in V_1$,

$$N(A_1) = \{u \in V_2 \text{ such that } (v, u) \in E \text{ for some } v \in A_1\}.$$

If there is a subset $A_1 \subset V_1$ such that $|A_1| > |N(A_1)|$ ten there is no perfect matching in G.

Bipartite Matching ($0/1$ weights)

Hall's (Marriage) Theorem Consider a bipartite graph $(V_1 \cup V_2, E)$ with $|V_1| = |V_2|$, there exists a perfect matching in G if and only if for every $A_1 \subset V_1$, $|A_1| \leq |N(A_1)|$

Hall's (Marriage) Theorem Consider a bipartite graph $(V_1 \cup V_2, E)$, there exists a perfect matching in G if and only if for every $A_1 \subset V_1$, $|A_1| \leq |N(A_1)|$

Consider a bipartite graph $G = (V_1 \cup V_2, E)$, the deficiency of G is

$$\text{def}(G) = \max_{A_1 \subset V_1} \left\{ |A_1| - |N(A_1)| \right\}$$

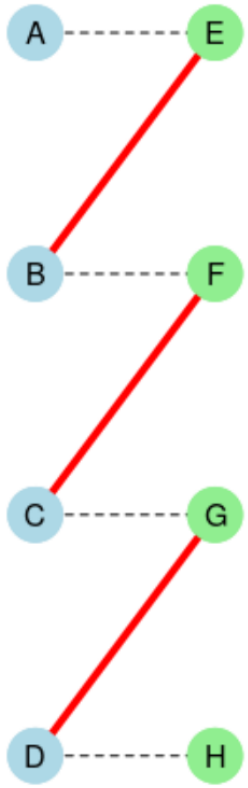
E.g. on the graph on the right $\text{def}(G) = 1$

Note that $\text{def}(G) \geq 0$

Bipartite Matching (0/1 weights)

Theorem Consider a bipartite graph $G = (V_1 \cup V_2, E)$, then the size of the maximum matching is $V_1 - \text{def}(G)$.

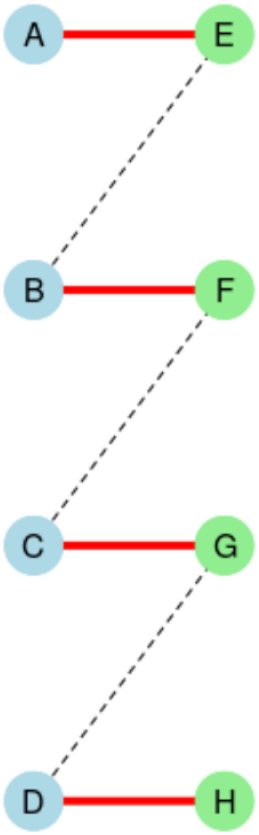
In a bipartite graph $G = (V_1 \cup V_2, E)$, and let M denote a matchin. A path is alternating for M if it starts with an unmatched vertice of V_1 and every other edge of it is in M.



Bipartite Matching ($0/1$ weights)

An alternating path is augmenting for M if it also ends with an unmatched vertice.
Let M denote an an augmenting path, then by switching the edges that are in M with the edges that are not, we obtain a larger matching.

Theorem If a matching M in a bipartite graph $G = (V_1 \cup V_2, E)$ is not a maximum matching, then there exists an augmenting path for M.



Bipartite Matching ($0/1$ weights)

This can be used to set up an algorithm to find a maximum matching

Consider a bipartite graph $G = (V_1 \cup V_2, E)$, start with any maching M (even a single edge)

Repeatedly find an augmenting path for M, and use it to find a larger matching

Bipartite Matchings and Network Flows

Given a bipartite graph $G = (V_1 \cup V_2, E)$, why not use a max-flow algorithm to find a maximum matching of G ?

Add a source s and edges to every vertice of V_1, and a sink t and edges to every vertice of V_2

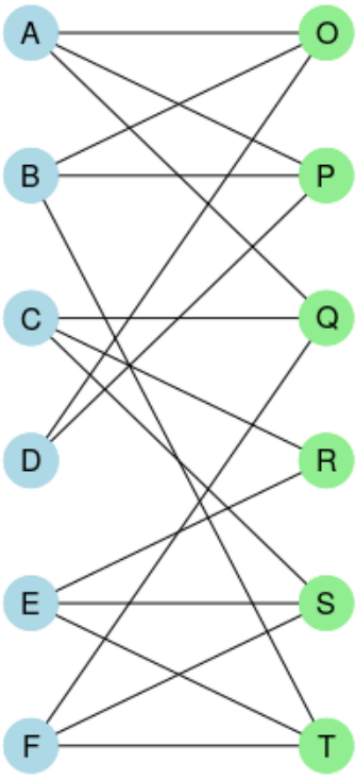
Direct edges from s to vertices of V_1, then to vertices of V_2 and finally to t.

Set capacities to 1.

There is a bijection between the matchings of G and the flows of the network.

Theorem A max-matching corresponds to a max-flow.

Try the following one



	O	P	Q	R	S	T
A	■	■	■	□	□	□
B	■	■	□	□	□	■
C	□	□	■	■	■	□
D	■	■	□	□	□	□
E	□	□	□	■	■	■
F	□	□	■	□	■	■

Bipartite Matching (general weights)

Consider now the case where non-negative weights $w(e)$ are assigned to each edge $e \in E$, see as a cost matrix $\boldsymbol{w}$

Example : you want to send three sales managers to three cities. They are incities $\{A, B, C\}$ and they need to visit $\{D, E, F\}$

Here are the cost of airplane ticksts

	D	E	F
A	250	400	350
B	400	600	350
C	200	400	250

Where should you send each of your salespeople in order to minimize airfare?

Bipartite Matching (general weights)

One possible assignment/matching is $\{A, D\}$, $\{B, E\}$,$\{C, F\}$, and the total cost is 1100

One possible assignment/matching is $\{A, D\}$, $\{B, F\}$,$\{C, E\}$, and the total cost is 1000

There are $n!$ ways of assigning n resources to n tasks.

Theorem: If a number is added to or subtracted from all of the entries of any one row or column of a cost matrix, then on optimal assignment for the resulting cost matrix is also an optimal assignment for the original cost matrix.

Hungarian Algorithm (Kuhn-Munkres)

Step 1. Subtract the smallest entry in each row from all entries of its row.

Step 2. Subtract the smallest entry in each column from all entries of its column.

Step 3. Draw lines through appropriate rows and columns so that all zero entries of the cost matrix are covered and minimum number of such lines is used.

Step 4. Test for Optimality:

(i) If the minimum number of covering lines is n, an optimal assignment of zeros is possible and we are finished.

(ii) If the minimum number of covering lines is less than n, an optimal assignment of zeros is not yet possible. In that case, proceed to Step 5.

Step 5. Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row, and then add it to each covered column. Return to Step 3 (see hrefhttps://brilliant.org/wiki/hungarian-matching/brilliant.org)

Hungarian Algorithm (Kuhn-Munkres)

In our previous example

Step 1:

	D	E	F
A	250	400	350
B	400	600	350
C	200	400	250

$\sim$

	D	E	F
A	0	150	100
B	50	250	0
C	0	200	50

Step 2:

	D	E	F
A	0	150	100
B	50	250	0
C	0	200	50

$\sim$

	D	E	F
A	0	0	100
B	50	100	0
C	0	50	50

Hungarian Algorithm (Kuhn-Munkres)

Step 3. Cover all the zeros of the matrix with the minimum number of horizontal or vertical lines

	D	E	F
A	0	0	100
B	50	100	0
C	0	50	50

Hungarian Algorithm (Kuhn-Munkres)

Step 4. Since the minimal number of lines is 3, an optimal assignment of zeros is possible and we are finished

Since the total cost for this assignment is 0, it must be an optimal assignment.

	D	E	F
A	0	0	100
B	50	100	0
C	0	50	50

$\sim$

	D	E	F
A	250	400	350
B	400	600	350
C	200	400	250

(we keep the same assignment to the original cost matrix)