Network Optimization Models IV
Maximum Flow Problems

Class 24

April 17, 2023

Handouts

- Notes on Assignment 7
- Notes on Assignment 8
- Green Mountain Power
- Optimization in Machine Learning and Data Science
  link to full article

Newsjournal of the Society for Industrial and Applied Mathematics sinews.siam.org

siam news

Volume 56/ Issue 3
April 2023

Optimization in Machine Learning and Data Science

By Stephen J. Wright

Machine learning (ML) and artificial intelligence (AI) have burst into public consciousness in the last several years. While large language and multimodal models like GPT-4 have recently taken the excitement to a new level, developments in voice recognition software, novel recommendation systems for online retailers and streaming services, superhuman-level play by computers in Chess and Go, and unfulfilled promises in technologies like self-driving cars have been generating interest for more than a decade. Many research disciplines are feeling the profound effects of AI. For example, scientists can now utilize neural networks (NNs) to predict a protein's structure based on its amino acid sequence [3] — a problem that was identified decades ago as a grand challenge for computational science.

ML, AI, data science, data analysis, data mining, and statistical inference all have different but overlapping meanings; the term "data science" is perhaps the most general. The remainder of this article will refer to ML since the problems that we discuss tend to cluster in that area.

Modern ML rests on multidisciplinary foundations in statistics, mathematics, and computer science. Optimization plays a central role by providing tools that formulate and solve computational problems in ML. Optimization *formulations* encapsulate statistical principles and distill them into computational problems that are solvable with algorithms and software. They also enable tradeoffs between competing goals via the judicious use of constraints or weighting of terms in the objective. Additionally, recent years have seen an outburst of research around optimization *algorithms* that are suited to the structure, scale, and context of ML applications — often building upon long-established foundations with new perspectives and insights.

ML aims to extract meaning from data, learn important features and fundamental structures, and use this knowledge to make predictions about other similar data. For example, an image classification system can learn how to identify an object in an image by processing thousands of sample images, each of which is labeled with the object it contains (see Figure 1).

After preprocessing, the m items of data that are presented to a ML system each consist of a vector of *features* a_j, $j=1,2,\ldots,m$ and a *label* y_j that is associated with each feature. The fundamental task is to learn a function ϕ that (approximately) maps each a_j to its associated y_j. The process of determining ϕ is often known as *training*, and the data (a_j,y_j), $j=1,2,\ldots,m$ are called *training data*. Usually, ϕ is parametrized by some finite vector x and the training problem reduces to finding x such that $\phi(a_j;x)\approx y_j$ for all $j=1,2,\ldots,m$; in short, it becomes a data-fitting problem. In a NN, x can be the vector of weights on all of the connections in the network — a vector that is believed to have more than a trillion components in the GPT-4 NN.

To formulate the data-fitting problem as an optimization problem, we define an objective $f(x)=\frac{1}{m}\sum_{i=1}^{m}\ell(\phi(a_j;x),y_j)$, where the *loss function* ℓ measures the discrepancy between the prediction $\phi(a_j;x)$ and the true label y_j on a single data point.

See **Optimization** *on page 2*

Figure 1. *The ImageNet dataset contains more than a million photos of one thousand objects. Figure courtesy of the ImageNet database at Princeton University and Stanford University.*

Announcements

MATHEMATICS SEMINAR

Dynamic and adaptive information accumulation and exchange during foraging

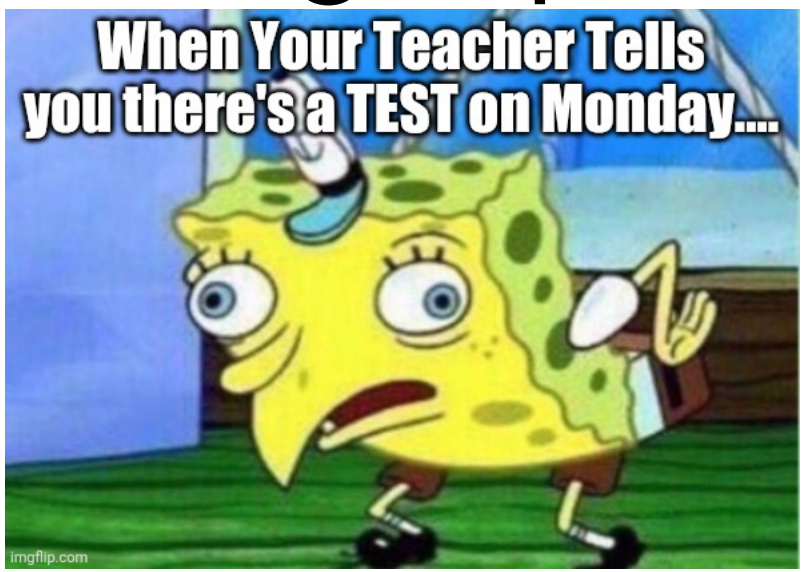
To effectively forage in natural environments, organisms must learn and adapt to changes in the availability of resources. Patch exploitation is a canonical foraging behavior, and the way in which animals account for environmental change and uncertainty should be captured more accurately by mathematical models. We first address this issue in a model describing agents that statistically and sequentially infer patch resource quality using Bayesian updating, based on their resource encounter history. Uncertainty leads patch-exploiting foragers to overharvest (underharvest) patches with initially low (high) resource yields. Social interactions that synchronize the times that foragers depart patches improve group foraging efficiency. We also address the problem of groups responding to rapid patch-quality changes. Using the example of honeybee swarms, we find social interactions that allow bees can directly switch the opinion of nest-mates foraging at lower-yielding feeders can quickly increase the fraction of the swarm at the correct feeder. Our mathematical framework allows us to compare the effects of a suite of mechanisms by which bees social inhibit the expressed opinions of their neighbors.

Zachary P Kilpatrick, PhD
Associate Professor
University of Colorado Boulder
Department of Applied Mathematics
https://www.colorado.edu/amath/zpkilpat

Friday, April 21 from 12:15 - 1:15pm in WNS 100

Announcements

Exam 2: Next Monday Evening, April 24



Network Optimization Problems

Minimal Spanning Tree

Shortest Path

Maximum Flow

Maximum Flow Problem


Paddington
Edgware Road
Baker Street
Great Portland Street
Euston
King's Cross St. Pancras
Angel
Old Street
Liverpool Street
Edgware Road
Marylebone
Warren Street
Euston Square
Farringdon
Regent's Park
Russell Square
Barbican
Bayswater
Notting Hill Gate
Lancaster Gate
Bond Street
Oxford Circus
Goodge Street
Chancery Lane
Moorgate
Holborn
Queensway
Marble Arch
Tottenham Court Road
St. Paul's
Bank
Aldgate
Covent Garden
High Street Kensington
Hyde Park Corner
Green Park
Cannon Street
Leicester Square
Mansion House
Monument
Tower Hill
Knightsbridge
Piccadilly Circus
Charing Cross
Gloucester Road
Sloane Square
St. James's Park
Blackfriars
Temple
River Thames
Earl's Court
South Kensington
Victoria
Westminster
Embankment
London Bridge
Waterloo


Oxygen, nitrogen, hydrogen and carbon monoxide pipelines
NORTHERN EUROPE networks
NETHERLANDS
BELGIUM
FRANCE
Rozenburg
Bergen-op-Zoom
Antwerpen
Terneuzen
Dunkerque
Mons
Feluy
Charleroi
Liège
Waziers
AIR LIQUIDE


GREEN
MOUNTAIN
POWER

Green Mountain Power (GMP) is a large electrical power generating company.

Study of future requirements for electricity in the region it serves:

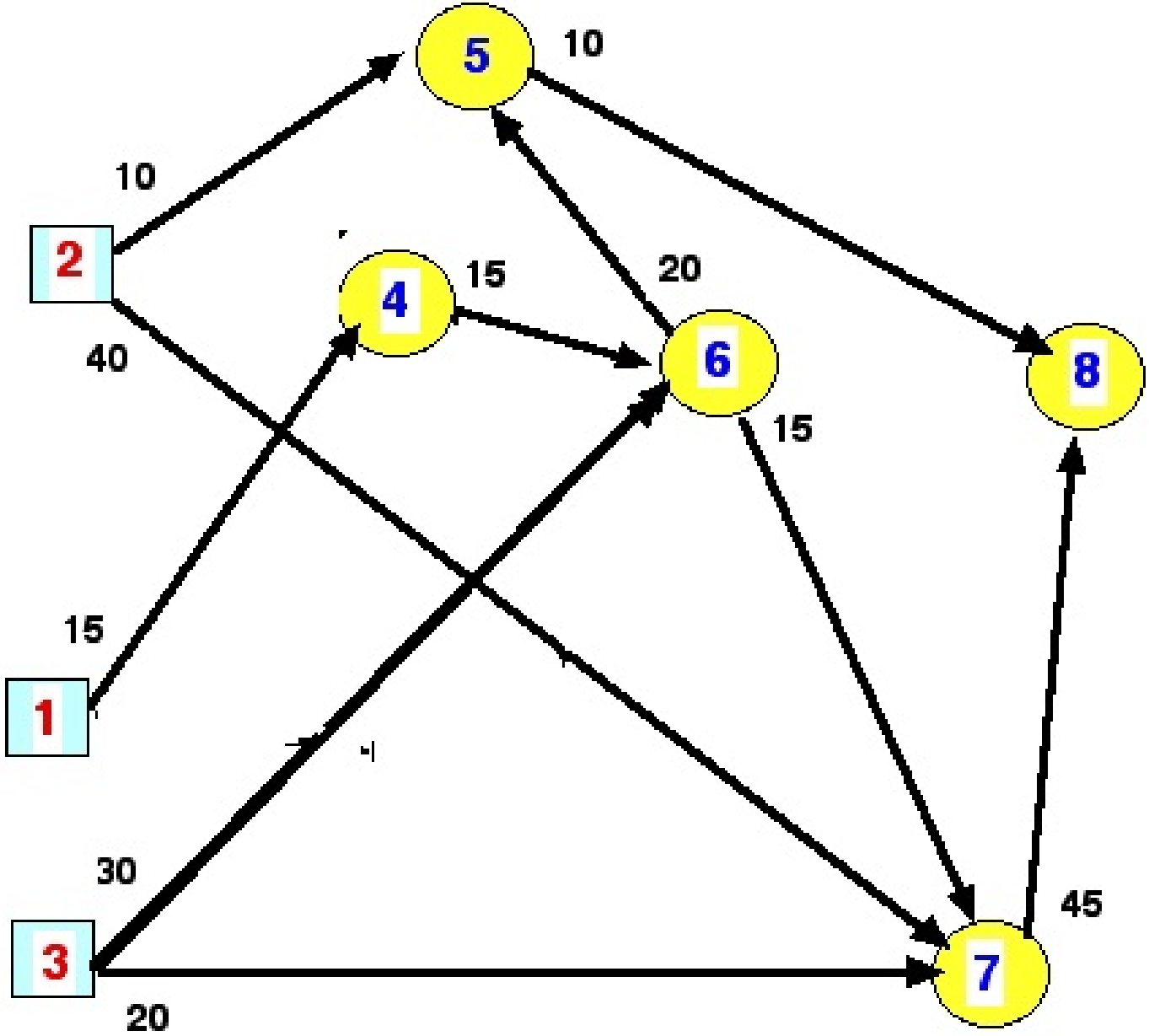
Are existing facilities (transmission lines, relay stations, etc.) adequate for transmitting the larger quantities of electricity required to accommodate increased future demands?

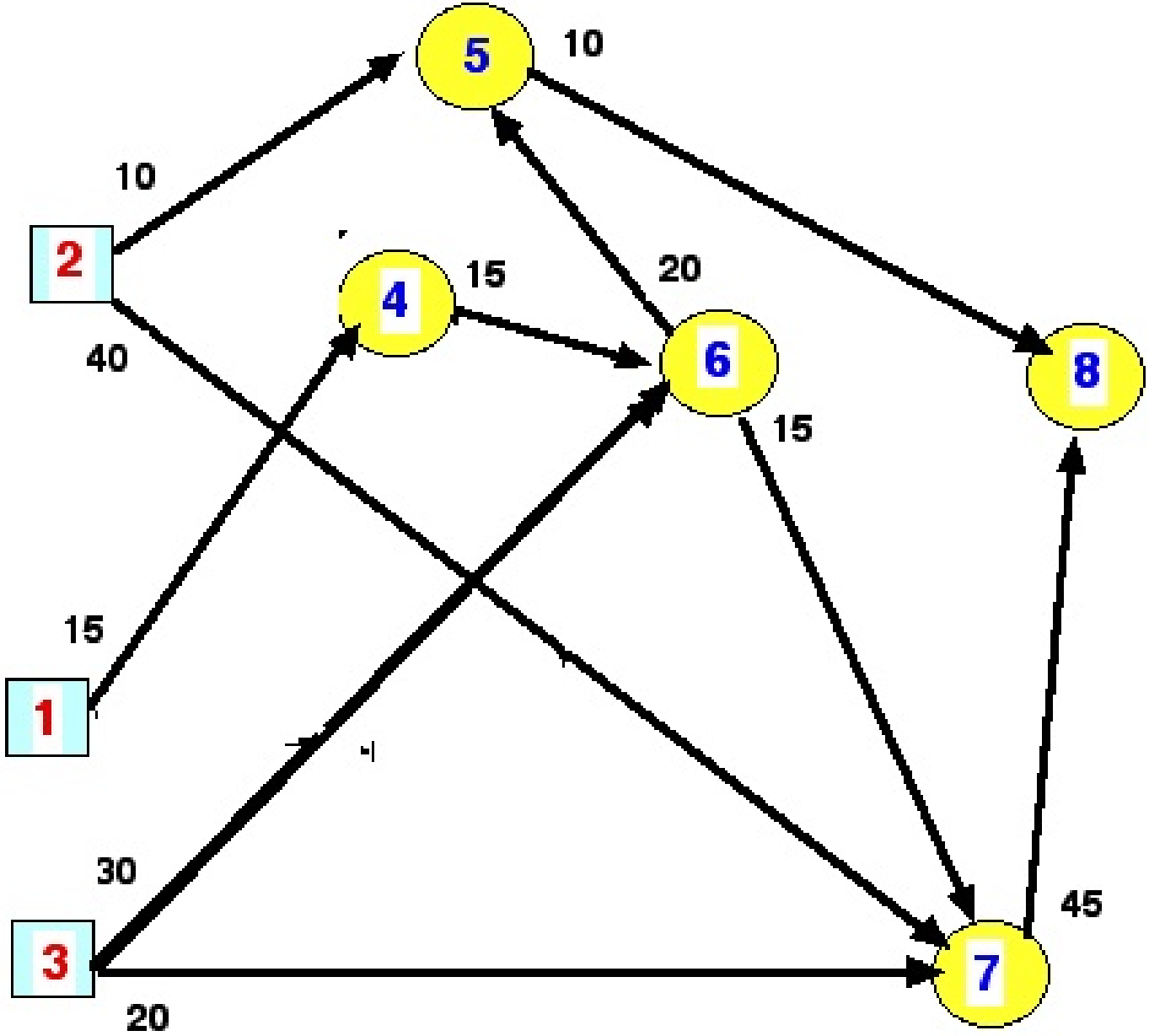
Suppose GMP has three generating stations, labeled $\boxed{1}$ $\boxed{,2}$ $\boxed{3}$, with respective capacities of 15, 10, and 40 megawatts; there are 5 cities, labeled $\boxed{4}$, $\boxed{5}$,$\boxed{6}$, $\boxed{7}$, and $\boxed{8}$, which form nodes on a graph.

There are transmission lines both between the generating stations and some cities and between some pairs of cities.

Each transmission line has a known **capacity** c_{ij}.

Major Question: What is the maximum number of megawatts that can be sent to each city from all three power stations?

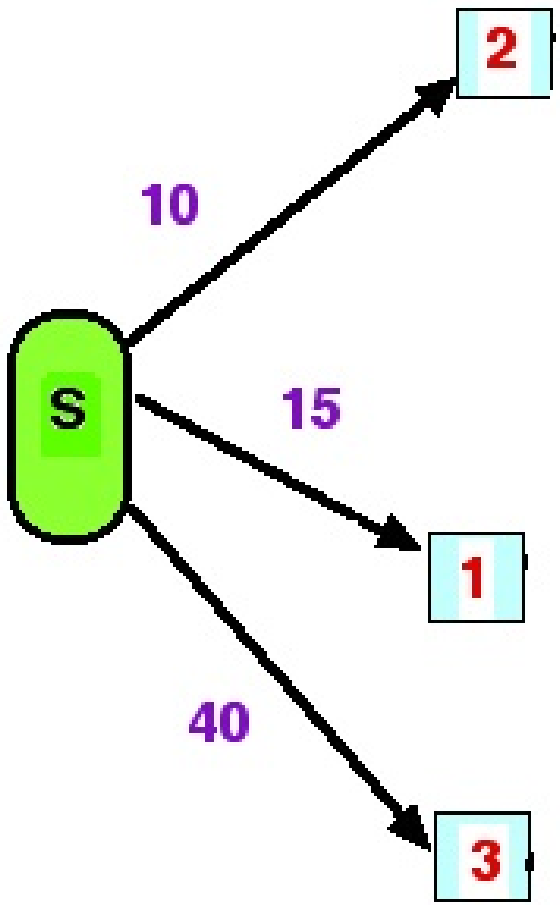

5
10
10
2
4
15
20
40
6
8
15
15
1
30
3
45
7
20

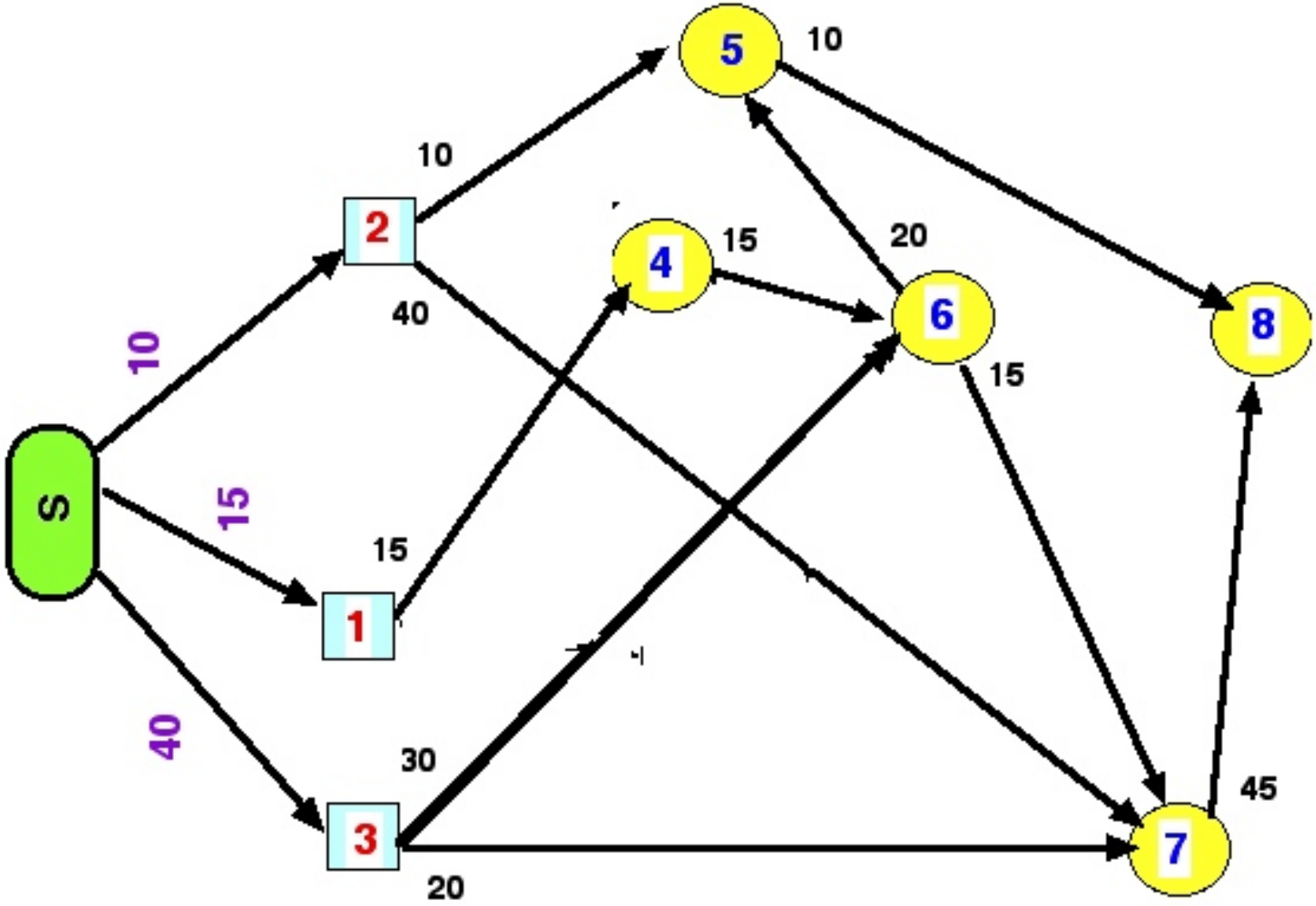

5
10
10
2
4
15
20
40
6
8
15
15
1
30
3
45
7
20

We have a "Multiple Source" Situation

It's convenient to convert to a network with a single **source** and a single **sink**

Add a new node, called a **Super Source S** and put a link from **S** to each original source and make the capacity of that link the number of megawatts that can be generated.

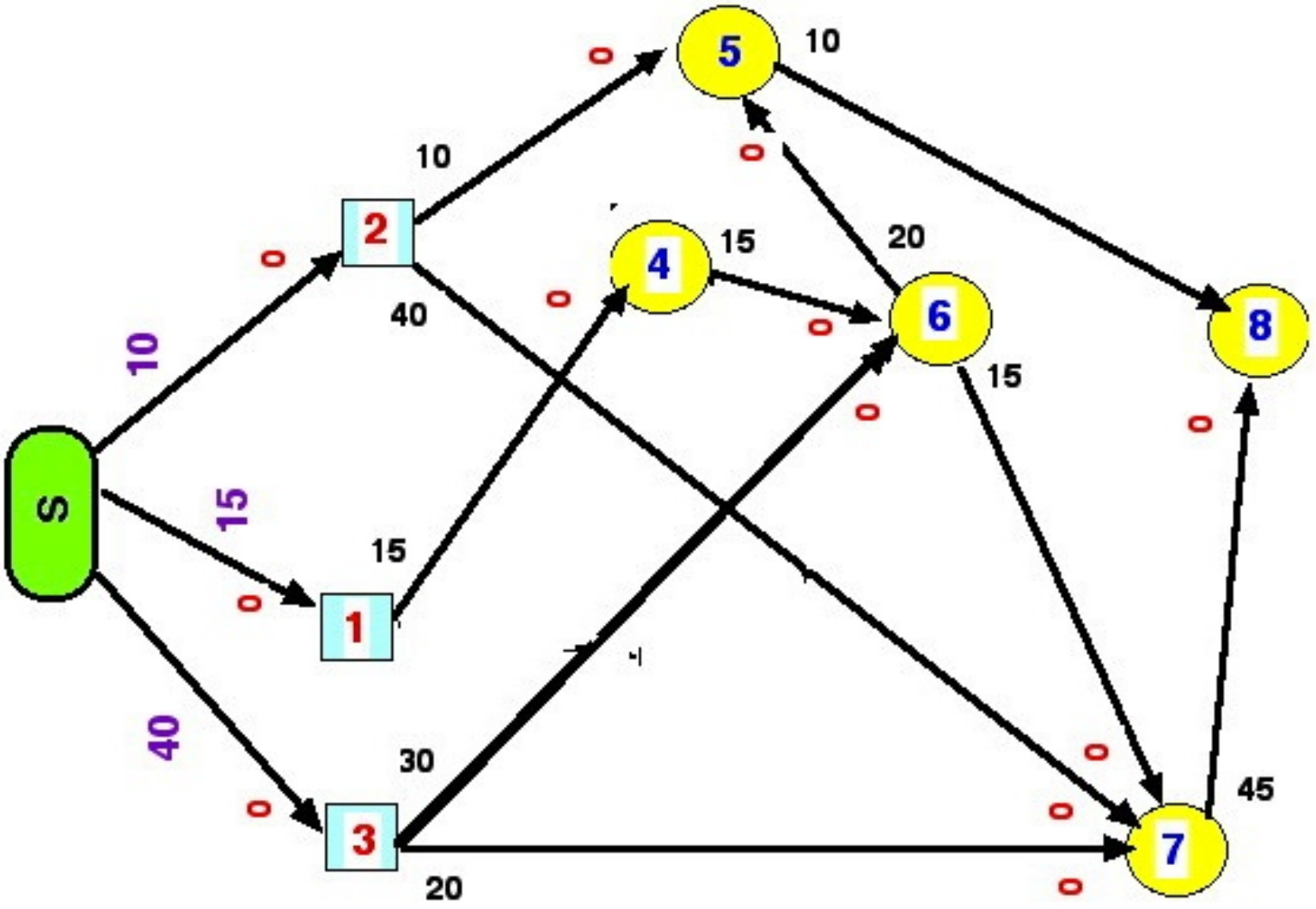



S
1
2
3
4
5
6
7
8
10
15
40
10
15
40
10
15
20
30
20
15
45

For this problem, we actually have 5 different maximum flow problems, one to each city.

We'll focus on node 8 as the destination.

Getting started:
Set all $x_{ij} = 0$.
Mark original capacities near start of each link
Let $c_{ji} = 0$ if there is no capacity in direction from j to i.


S
1
2
3
4
5
6
7
8
10
15
40
10
15
40
30
20
10
15
20
15
45
0
0
0
0
0
0
0
0
0
0
0
0
0
0

Original Capacities

S1 : 15	S2 : 10	S3: 40
14 : 15		
25 : 10	27:40	
37: 20	46 :15	58: 10
65: 20	67: 15	
78: 45		

Begin at the **source** and select any path to the destination which has positive capacity **c** where **capacity of a path = minimum of capacities of links on that path.**

Example: $\boxed{S} \xrightarrow{40} \boxed{3} \xrightarrow{20} \boxed{7} \xrightarrow{45} \boxed{8}$ has capacity $\mathbf{c} = 20$.

Augment Flows

$$x_{S3} \rightarrow x_{S3} + 20 = 0 + 20 = 20$$
$$x_{37} \rightarrow x_{37} + 20 = 0 + 20 = 20$$
$$x_{78} \rightarrow x_{78} + 20 = 0 + 20 = 20$$

Augment Flows And Update Capacities

$$x_{S3} \rightarrow x_{53} + 20 = 0 + 20 = 20$$
$$x_{37} \rightarrow x_{37} + 20 = 0 + 20 = 20$$
$$x_{78} \rightarrow x_{78} + 20 = 0 + 20 = 20$$

Modify Capacities of Arcs

$c_{ij} \rightarrow c_{ij} - \mathbf{c}$ for all arcs (i, j) on path

$c_{ji} \rightarrow c_{ji} + \mathbf{c}$ for all arcs (i, j) on path

This yields a **residual network** with **residual capacities** on arcs.

Then select new path from **S** to $\boxed{8}$, called **Augmenting Path** and continue as before.

Augmenting Paths for GMP Problem

Iteration 1: $\boxed{S} \rightarrow \boxed{3} \rightarrow \boxed{7} \rightarrow \boxed{8}$

Iteration 2: $\boxed{S} \rightarrow \boxed{2} \rightarrow \boxed{5} \rightarrow \boxed{8}$

Iteration 3: $\boxed{S} \rightarrow \boxed{1} \rightarrow \boxed{4} \rightarrow \boxed{6} \rightarrow \boxed{7} \rightarrow \boxed{8}$

Iteration 4: $\boxed{S} \rightarrow \boxed{3} \rightarrow \boxed{6} \rightarrow \boxed{5} \rightarrow \boxed{2} \rightarrow \boxed{7} \rightarrow \boxed{8}$

Capacities

Iteration 1: 20

Iteration 2: 10

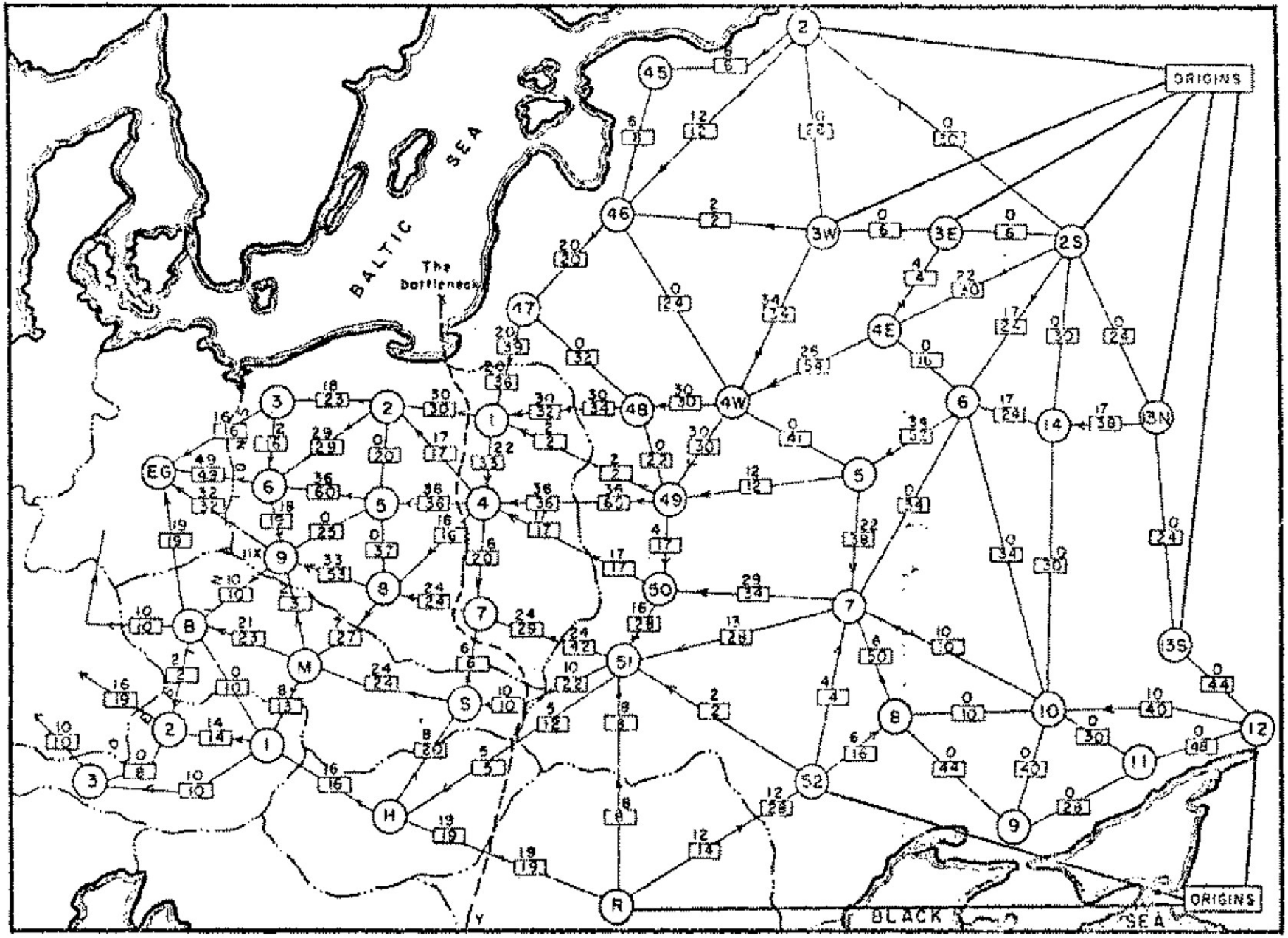
Iteration 3: 15

Iteration 4: 10

Final Flow Assignments

$$
\begin{array}{llll}
x_{S3} = 30 & x_{14} = 15 & x_{25} = 0 & x_{36} = 10 \\
x_{S2} = 10 & & x_{27} = 10 & x_{37} = 20 \\
x_{S1} = 30 & & & \\
 & & & \\
x_{46} = 15 & x_{58} = 10 & x_{65} = 10 & x_{78} = 45 \\
 & & x_{67} = 15 &
\end{array}
$$

Maximum Flow is 55 megawatts


BALTIC SEA
The Bottleneck
ORIGINS
ORIGINS
BLACK SEA

Schematic diagram of the railway network of the Western Soviet Union and Eastern European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as "The bottleneck".

T.E. Harris, F.S. Ross, *Fundamentals of a Method for Evaluating Rail Net Capacities*, Research Memorandum RM-1573, The RAND Corporation, Santa Monica, California, 1955.

The Maximum Flow Problem

1) All flow through a directed and connected network originates at one node, called the **source** and terminates at one other node, called the **sink**.

2) All the remaining nodes are *transshipment nodes*.

3) Flow through an arc is allowed only in the direction indicated by the arrowhead, where the maximum amount of flow is given by the *capacity* of that arc. At the **source**, all arcs point away from the node. At the **sink**, all arcs point into the node.

4) The objective is to maximize the total amount of flow from the **source** to the **sink**. This amount is measured in either of two equivalent ways, namely, either the amount leaving the source or the amount entering the sink.

The Ford-Fulkerson Algorithm

After some flows have been assigned to the arcs, the residual network shows the remaining arc capacities (called residual capacities) for assigning additional flows.

Change each directed arc to an undirected arc. The arc capacity in the original direction remains the same and the arc capacity in the opposite direction is zero, so the constraints on flows are unchanged.

Subsequently, whenever some amount of flow is assigned to an arc, that amount is subtracted from the residual capacity in the same direction and added to the residual capacity in the opposite direction.

An **augmenting path** is a directed path from the source to the sink in the residual network such that every arc on this path has strictly positive residual capacity. The minimum of these residual capacities is called the **residual capacity** of the augmenting path because it represents the amount of flow that can feasibly be added to the entire path. Therefore, each augmenting path provides an opportunity to further augment the now through the original network.

The augmenting path algorithm repeatedly selects some augmenting path and adds a flow equal to its residual capacity to that path in the original network. This process continues until there are no more augmenting paths, so the flow from the source to the sink cannot be increased further. **The key to ensuring that the final solution necessarily is optimal is the fact that augmenting paths can cancel some previously assigned flows in the original network**, so an indiscriminate selection of paths for assigning flows cannot prevent the use of a better combination of flow assignments.

The Augmenting Path Algorithm for the Maximum Flow Problem

1) Identify an augmenting path by finding some directed path from the source to the sink in the residual network such that every arc on this path has strictly positive residual capacity. (If no augmenting path exists, the net flows already assigned constitute an optimal flow pattern.)

2) Identify the residual capacity c* of this augmenting path by finding the minimum of the residual capacities of the arcs on this path. **Increase** the flow in this path by c*.

3) **Decrease by c*** the residual capacity of each arc on this augmenting path. Increase by c* the residual capacity of each arc in the opposite direction on this path. Return to step 1.

- adapted from Hillier and Lieberman

Finding An Augmented Path

1) Begin by determining all nodes that can reached from the source with a single arc with positive residual capacity.

2) For each of these nodes, determine all new nodes that can be reached with a single arc with positive residual capacity.

3) Repeat Step 2

This procedure yields a tree of all nodes that can be reached from the source.

The Max-Flow Min-Cut Theorem

A **cut** is a set of directed arcs containing at least one arc from every directed path from the source to the sink.

The **cut value** of a cut in the sum of the arc capacities of the arcs in the cut.

The Max-Flow Min-Cut Theorem: For any network with a single source and single sink, the maximum feasible flow from source to sink equals the minimum cut value for all cuts of the network.

Delbert Ray Fulkerson
Born: August 14, 1924
Died: January 10, 1976
Biography

Lester Randolph Ford, Jr.
Born: September 23, 1927
Died: February 26, 2017)
Biography

Important Works by Ford and Fulkerson

"Maximal Flow Through a Network," *Canadian Journal of Mathematics*, 8:399 - 404, 1956.

"A Simple Algorithm for Finding Maximal Network Flows," *Canadian Journal of Mathematics*, 9: 210 - 218, 1957.

Flows in Networks, Princeton University Press, 1962.
New Edition in 2010 with a new foreword by Robert G. Bland and James B. Orlin.

Complexity

By adding the flow augmenting path to the flow already established in the graph, the maximum flow will be reached when no more flow augmenting paths can be found.
There is no certainty that this situation will ever be reached. The best that can be guaranteed is that the answer will be correct if the algorithm terminates. If the algorithm runs forever, the flow might not even converge towards the maximum flow.
However, this situation only occurs with irrational flow values.
When the capacities are integers, the runtime of Ford–Fulkerson is bounded by $O(Ef)$ where E is the number of edges in the graph and f is the maximum flow in the graph.
This is because each augmenting path can be found in $O(E)$ time and increases the flow by an integer amount of at least 1 with the upper bound f

A variation of the Ford – Fulkerson algorithm with guaranteed termination and a runtime independent of the maximum flow value is the Edmonds – Karp algorithm, which runs in $O(VE^2)$

Yefim Dinitz
Ben Gurion University

Jack Edmonds
Waterloo

Richard Karp
Berkeley

Dinic, E. A. (1970). "Algorithm for solution of a problem of maximum flow in a network with power estimation". *Soviet Math. Doklady.* 11: 1277 –1280.
Edmonds, Jack; Karp, Richard M. (1972). "Theoretical improvements in algorithmic efficiency for network flow problems". *Journal of the ACM*. Association for Computing Machinery. 19 (2): 248 – 264.

Shortest Path as LP Problem

The shortest-path problem is a minimum cost flow problem with a unit supply at the origin and a unit demand at the Destination.
Label the Origin as node 1 and the Destination as node n.
Then Linear Programming problem is
Minimize

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^{n} x_{1j} - \sum_{j=1}^{n} x_{j1} = 1$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{jn} = 0, \text{for } 2 \leq i \leq n-1$$

$$\sum_{j=1}^{n} x_{nj} - \sum_{j=1}^{n} x_{jn} = -1$$

$$0 \leq x_{ij} \leq 1, \text{for } 1 \leq i, j \leq n$$

Next :

An Application to Baseball

An Application to Baseball

Baseball Elimination Via Max Flow

"See that thing in the paper last week about Einstein? ...
Some reporter asked him to figure out the mathematics
of the pennant race.
You know, one team wins so many of their remaining games,
the other teams win this number or that number.
What are the myriad possibilities?
Who's got the edge?"
"The hell does he know?"
"Apparently not much.
He picked the Dodgers to eliminate the Giants last Friday."
Don DeLillo, *Underworld*

Team	Wins	Games To Play	Against Atlanta	Against Phily	Against NY	Against Miami
Atlanta	83	8	-	1	6	1
Philadelphia	80	3	1	-	0	2
New York	78	6	6	0	-	0
Miami	77	3	1	2	0	-

Which teams have a chance of finishing the season with the most wins?

Miami is eliminated: it can finish with at most 80 wins, but Atlanta already has 83.

Sportswriters use *The Magic Number*.

Magic Number

Another Example: Can Boston finish with at least as many wins as every other team in the division?

Team	Wins	Games To Play	Against New York	Against Baltimore	Against Toronto	Against Boston
New York	92	2	-	1	1	0
Baltimore	91	3	1	-	1	1
Toronto	91	3	1	1	-	1
Boston	90	2	0	1	1	-

First Glance: Yes. Boston wins both its remaining games, Baltimore and Toronto win exactly 1, and New York loses both its games.

Team	Wins	Games To Play	Against New York	Against Baltimore	Against Toronto	Against Boston
New York	92	2	-	1	1	0
Baltimore	91	3	1	-	1	1
Toronto	91	3	1	1	-	1
Boston	90	2	0	1	1	-

Second Thought: No. If New York loses both of its games, then Baltimore and Toronto each pick up a win; the winner of the Baltimore -Toronto game finishes with 93 victories.

Team	Wins	Games To Play	Against New York	Against Baltimore	Against Toronto	Against Boston
New York	92	2	-	1	1	0
Baltimore	91	3	1	-	1	1
Toronto	91	3	1	1	-	1
Boston	90	2	0	1	1	-

A Different Analysis: Boston can win at most 92 games. The other three teams have a cumulative total of $92 + 91 + 91 = 274$ wins. Their three games against each other will produce an additional 3 wins for a final total of 277 wins. One of the teams must end up with more than 92 wins since the average number of wins is $277/3 > 92$.

- Is there an efficient algorithm to determine whether a given team has been eliminated from first place?
- When a team has been eliminated, is there an averaging argument that proves it?

A Mathematical Formulation

We have a set S of teams.

For each team x in S, let w_x = its current number of wins.

For each pair of teams x, y, let g_{xy} = the number of games they will play against each other.

Let z represent the team in S whose fate we wish to examine.

if T is a subset of the set of teams, then $|T|$ denotes the number of teams in T.

Theorem *Characterization Theorem for Baseball Elimination*): Suppose team z has been eliminated. Then there exists a proof of this fact of the following form:

- z can finish with at most m wins.
- There is a subset T of S teams such that

$$\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} > m|T|$$

Another Example

Team	Wins	Games To Play	Against New York	Against Baltimore	Against Toronto	Against Boston
New York	90	7	-	1	6	-
Baltimore	88	2	1	-	1	-
Toronto	87	7	6	1	-	-
Boston	79	12	-	-	-	-

Claim: Boston has been eliminated.
Boston can finish with at most $m = 79 + 12 = 91$ wins.
Let $T = \{\text{New York, Toronto}\}$. Then

$$\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} = 90 + 87 + 6 = 183 > 91 \cdot 2 = 182.$$

One of New York or Toronto will finish with at least 92 wins.

Team	Wins	Games To Play	Against New York	Against Baltimore	Against Toronto	Against Boston
New York	90	7	-	1	6	-
Baltimore	88	2	1	-	1	-
Toronto	87	7	6	1	-	-
Boston	79	12	-	-	-	-

Note: The set $T =$ New York, Toronto, Baltimore doesn't work: Here

$$\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} = 265 + 8 = 273$$

with average $273/3 = 91$.

Designing and Analyzing the Algorithm

Suppose there's a way for z to end up in first place with m wins. We now want to allocate wins for all remaining games so no other team finishes with more than m wins.

We'll allocate wins using a maximum flow computation.
We have a source s from which all wins emanate.
The ith win can pass through one of the two teams involved in the ith game.
We will then impose a capacity constraint: at most $m - w_x$ wins can pass through team x.

Construct a flow network G.
Let $S' = S - \{z\}$ (The set of other teams).
Let $g^* = \sum_{x,y \in S'} g_{xy}$ (total number of games left between all pairs.

Nodes

- s a source and t a sink.
- a node v for each team in S'.
- a node u_{xy} for each pair of teams in S' with a nonzero number of games left to play against each other.

Edges

- (s, u_{xy}): wins emanate from s.
- (v_x, t): wins are absorbed at t.
- (u_{xy}, v_x) and (u_{xy}, v_y): only x or y can win a game that they play against each other.

Capacities:

- Capacity of (s, u_{xy}) should be g_{xy}.
- Capacity of (v_x, t) should be $m - w_x$ (Ensure that team x cannot win more than $m - w_x$ games).
- Capacity of each (u_{xy}, v_x) will be infinite.

Team	Wins	Games To Play	Against New York	Against Baltimore	Against Toronto	Against Boston
New York	90	7	-	1	6	-
Baltimore	88	2	1	-	1	-
Toronto	87	7	6	1	-	-
Boston	79	12	-	-	-	-

Is Boston eliminated?

$m = 91$

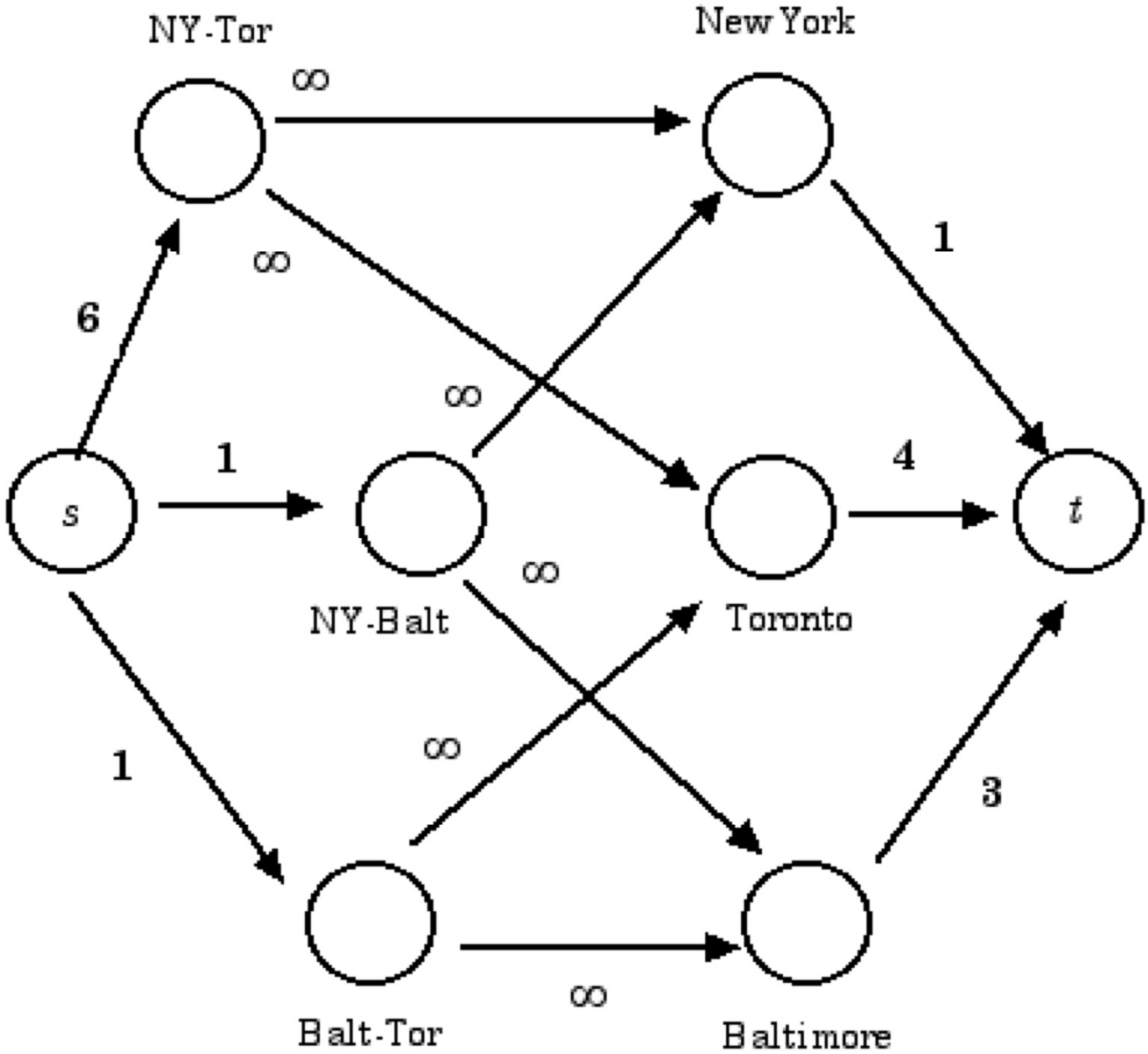
$S' = S - \{z\} = \{$ New York, Baltimore, Toronto$\}$

$g^* = 8$ games left

Capacity of $(v_{NewYork}, t) = m - w_{NewYork} = 91 - 90 = 1$

Capacity of $(v_{Baltmore}, t) = m - w_{Baltimore} = 91 - 88 = 3$

Capacity of $(v_{Toronto}, t) = m - w_{Toronto} = 91 - 87 = 4$

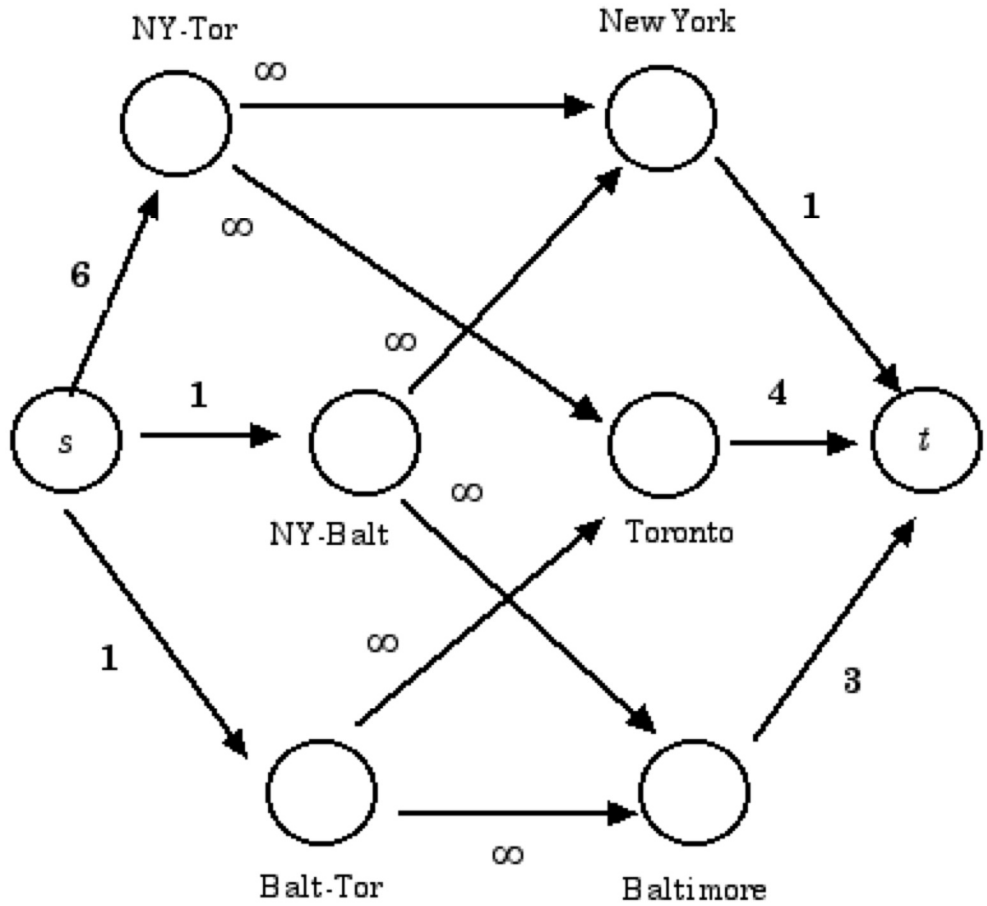

NY-Tor
New York
∞
∞
6
1
∞
1
4
s
t
NY-Balt
Toronto
∞
1
∞
3
Balt-Tor
∞
Baltimore

If there is a flow of value g^*, then it is possible for the outcomes of all remaining games to yield a situation where no team has more than m wins.

Hence if z wins all its remaining games, it can still achieve at least a tie for first place.

Conversely, if there are outcomes for the remaining games in which z does achieve at least a tie, we can use these outcomes to define a flow of value g^*.

Boston has a chance if and only if the maximum flow in the network is at least g* = 8 .



The maximum flow in this network is only 7
A Minimum cut is { $s \to$ Bal-Tor, $s \to$ NY-Bal, Toronto $\to t$, New York $\to t$. }

We have shown: **Team z has been eliminated if and only if the maximum flow in G has value strictly less than g^*.**

Characterizing When a Team is Eliminated

Theorem (*Characterization Theorem for Baseball Elimination*): Suppose team z has been eliminated. Then there exists a proof of this fact of the following form:

- z can finish with at most m wins.
- There is a subset T of S such that

$$\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} > m|T|$$

Proof of Theorem: I. Suppose z has been eliminated.
The maximum $s - t$ flow in G has value $g' < g^*$

There is an $s - t$ minimum cut (A, B) of capacity g'

Let T be the set of teams x for which v_x is in A.

Claim: We can use T in the "averaging argument."

First, consider the node u_{xy} and suppose one of x or y is not in T, but u_{xy} is in A. Then the edge (u_{xy}, v_x) would cross from A to B, and hence the cut (A, B) would have infinite capacity.

But this contradicts the assumption that (A, B) is a minimum cut of capacity less than g^*.

Thus, if x or y is not in T, then u_{xy} is in B.

On the other hand, suppose x and y both belong to T, but u_{xy} is in B. Consider the cut (A', B') obtained by adding u_{xy} to the set A and deleting it from the set B.

The capacity of (A', B') is simply the capacity of (A, B) minus the capacity g_{xy} of the edge (s, u_{xy}) for this edge (s, u_{xy}) used to cross from A to B, but now does not cross from A' to B'.

Since $g_{xy} > 0$, the cut (A', B') has smaller capacity than the cut (A, B) , contradicting the minimality assumption on (A, B).

Thus, if x and y belong to T, then u_{xy} is in A.

We have established: u_{xy} is in A if and only if both x and y are in T.

Now let's determine the cut-value $c(A, B)$. The edges crossing from A to B have two possible forms:

- edges of the form (v_x, t) where x is in T, and
- edges of the form (s, u_{xy}) where at least one of x or y does not belong to T. (ie, $\{x, y\}$ is not a subset of T.)

$$\text{Thus } c(A, B) = \sum_{x \in T} (m - w_x) + \sum_{x,y \in T} g_{xy}$$

$$= m|T| - \sum_{x \in T} w_x + (g^* - \sum_{x,y \in T} g_{xy})$$

but we know that $c(A, B) = g' < g^*$ so

$$m|T| - \sum_{x \in T} w_x + (g^* - \sum_{x,y \in T} g_{xy}) < g^*$$

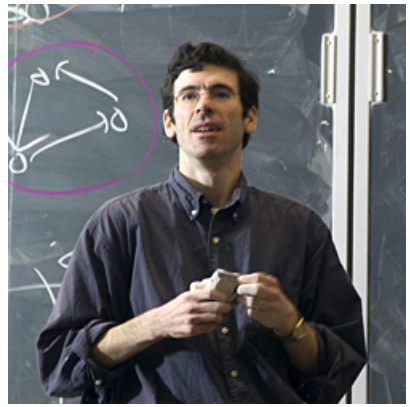
$$\text{implying } m|T| - \sum_{x \in T} w_x - \sum_{x,y \in T} g_{xy} < 0$$

or

$$\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} > m|T|$$

Adapted from Jon Kleinberg and Éva Tardos, *Algorithm Design*, Boston: Pearson Addison-Wesley, 2006.

Éva Tardos
October 1, 1957
Tardos Home Page



Jon Kleinberg
October 16, 1971
Kleinberg Home Page

Maximum Flow as LP Problem

Label the Source as node 1 and the Sink as node N.
Then the Linear Programming problem is

$$\text{Maximize } z = \sum_{j=2}^{N} x_{1j}$$

subject to

$$\sum_{j=2, j\neq i}^{N} x_{ij} - \sum_{j=2, j\neq i}^{N} x_{ji} = 0, \text{ for } i = 2, 3, ..., N-1$$

$$0 \leq x_{ij} \leq c_{ij}, \text{ where } c_{ij} = 0 \text{ if } (i, j) \text{ is not a branch}$$

Next Week:

Dynamic Programming