CS711008Z Algorithm Design and Analysis

Lecture 10. Algorithm design technique: Network flow and its applications [1]

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

[1] The slides are made based on Chapter 7 of *Introduction to algorithms*, *Combinatorial optimization algorithm and complexity* by C. H. Papadimitriou and K. Steiglitz, the classical papers by Kuhn, Edmonds, etc. in the book *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art.*

Outline

- Extensions of MaximumFlow problem: undirected network; Circulation with multiple sources & multiple sinks; Circulation with lower bound of capacity; Minimum Cost Flow;
- Solving practical problems using network flow and primal_dual techniques:
  1. Partitioning a set: ImageSegmentation, ProjectSelection, ProteinDomainParsing;
  2. Finding paths: FlightScheduling, Disjoint Paths, BaseballElimination;
  3. Decomposing numbers: BaseballElimination;
  4. Constructing matches: BipartiteMatching, SurveyDesign;
- Extensions of matching: BipartiteMatching, WeightedBipartiteMatching, GeneralGraphMatching, WeightedGeneralGraphMatching;
- A brief history of network flow.

Extensions of MaximumFlow problem

Extensions

Four extensions of MAXIMUMFLOW problem:

1. MAXIMUMFLOW for undirected network;
2. CIRCULATION with multiple sources and multiple sinks;
3. CIRCULATION with lower bound for capacity;
4. MINIMUM COST FLOW;

Extension 1: MAXIMUM FLOW for undirected network

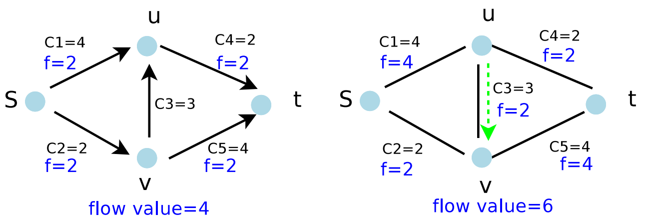
Extension 1: Maximum Flow for undirected network

INPUT:
an **undirected** network $G = < V, E >$, each edge e has a capacity $C(e) > 0$. Two special nodes: **source** s and **sink** t;
OUTPUT:
for each edge e, to assign a flow $f(e)$ to maximize the flow value $\sum_{e=(s,v)} f(e)$.

Flow properties:

1. (Capacity restriction): $0 \leq f(u,v) + f(v,u) \leq C(u,v)$ for any $(u,v) \in E$;
2. (Conservation restriction): $f^{in}(v) = f^{out}(v)$ for any node $v \in V$ except for s and t.

Example



Note: On the directed network, the maximum flow value is 4; in contrast, on the undirected network, the maximum flow value is 6.

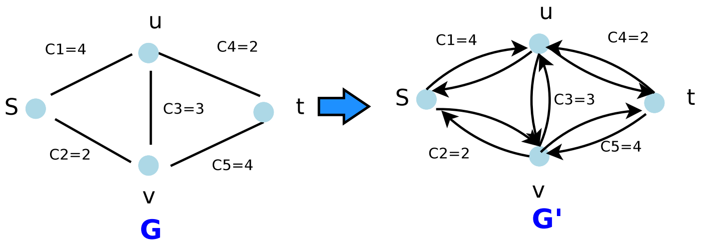
Algorithm

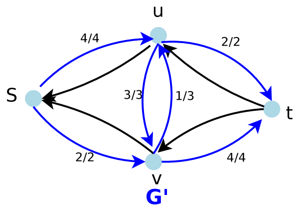
Maximum-flow algorithm for undirected network G

1: Transforming the undirected network G to a directed network G';
2: Calculating the maximum flow for G' by using Ford-Fulkerson algorithm;
3: Revising the flow to meet the capacity restrictions;

Step 1: changing undirected network to directed network

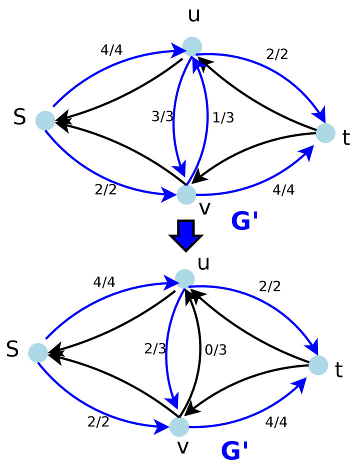
- Transformation: an undirected network G is transformed into a directed network G' through:
  1. adding edges: for each edge (u, v) of G, introducing two edges $e = (u, v)$ and $e' = (v, u)$ to G';
  2. setting capcities: setting $C(e') = C(e)$.





Note: the only trouble is the violation of capacity restriction: for edge $e = (u, v)$, $f(e) + f(e') = 4 > C(e) = 3$.

Step 3: revising flow to meet capacity restriction

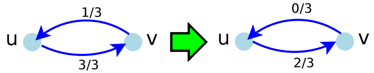


Note: for an edge violating capacity restriction, say $e = (u, v)$, the flow $f(e)$ and $f(e')$ were revised.

Correctness of revising flow

Theorem

There exists a maximum flow f for network G, where $f(u,v) = 0$ or $f(v,u) = 0$.



Proof.

- Suppose f' is a maximum flow for undirected network G', where $f'(u,v) > 0$ and $f'(v,u) > 0$. We change f' to f as follows:
- Let $\delta = \min\{f'(u,v), f'(v,u)\}$.
- Define $f(u,v) = f'(u,v) - \delta$, and $f(v,u) = f'(v,u) - \delta$. We have $f(u,v) = 0$ or $f(v,u) = 0$.
- It is obvious that both capacity restrictions and conservation restrictions hold.
- f has the same value to f' and thus optimal.

Extension 2: CIRCULATION problem with multiple sources and multiple sinks

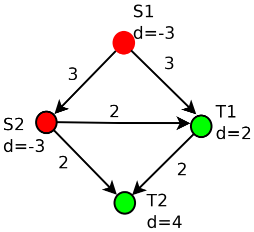
Extension 2: CIRCULATION problem with multiple sources and multiple sinks

INPUT:
a network $G =< V, E >$, where each edge e has a capacity $C(e) > 0$; multi sources s_i and sinks t_j. A sink t_j has demand $d_j > 0$, while a source s_i has supply d_i (described as a negative demand $d_i < 0$).
OUTPUT:
a **feasible circulation** f to satisfy all demand requirements using the available supply, i.e.,

1. Capacity restriction: $0 \leq f(e) \leq C(e)$;
2. Demand restriction: $f^{in}(v) - f^{out}(v) = d_v$;

Note: For the sake of simplicity, we define $d_v = 0$ for any node v except for s_i and t_j. Thus we have $\sum_i d_i = 0$, and denote $D = \sum_{d_v > 0} d_v$ as the **total demands** .

An example



Note: The differences between CIRCULATION and MULTICOMMODITIES problem:

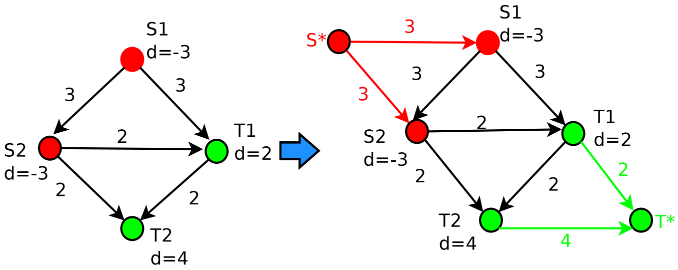
1. CIRCULATION problem: There is ONLY one type of commodity: a sink t_i can accept commodity from **any** source. In other words, the combination of commodities from all sources constitutes the demand of t_i.
2. MULTICOMMODITIES problem: There are multiple commodities, say transferring $food$ and oil in the same network. Here t_i (say demands $food$) accepts commodity k_i from s_i (say sending $food$) only. Linear programming is the only known polynomial-time algorithm for the MULTICOMMODITIES problem.

Algorithm for circulation:

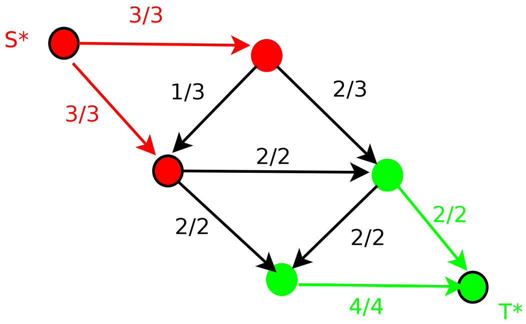
1: Constructing an expanded network G' via adding super source S^* and super sink T^*;
2: Calculating the maximum flow f for G' by using Ford-Fulkerson algorithm;
3: Return flow f if the maximum flow value is equal to $D = \sum_{v:d_v>0} d_v$.

Step 1: constructing an expanded network G'

Transformation: constructing a network G' through adding a super source s^* to connect each s_i with capacity $C(s^*, s_i) = -d_i$. Similarly, adding a super sink t^* to connect to each t_j with capacity $C(t_j, t^*) = d_j$.

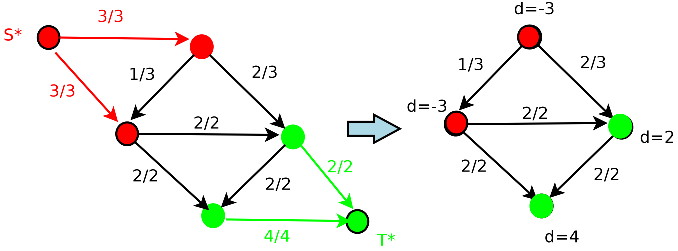


Step 2: calculating the maximum flow for G'



Note: a/b means f(e) = a, and capacity C(e)=b.

Step 3: checking the maximum flow for G'



Note: a/b means f(e) = a, and capacity C(e)=b.

The maximum flow value is $6 = \sum_{v, d_v > 0} d_v$. Thus, we obtained a feasible solution to the original circulation problem.

Correctness

Theorem

There is a feasible solution to CIRCULATION *problem iff the maximum $s^* - t^*$ flow in G' is D.*

Proof.

- $\Leftarrow$
  Simply removing all (s^*, s_i) and (t_j, t^*) edges. It is obvious that both capacity constraint and conservation constraint still hold for all s_i and t_j.
- $\Rightarrow$
  We construct a $s^* - t^*$ flow and prove that it is a maximum flow:
  1. Define a flow f as follows: $f(s^*, s_i) = -d_i$ and $f(t_j, t^*) = d_j$.
  2. Consider a special cut (A, B), where $A = \{s^*\}$, $B = V - A$.
  3. We have $C(A, B) = D$. Thus f is a maximum flow since it reaches the maximum value.

$\square$

Extension 3: CIRCULATION with lower bound for capacity

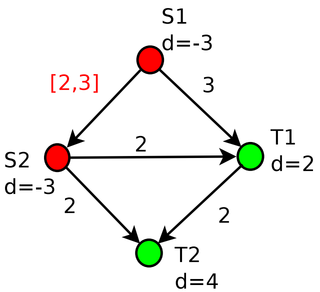
Extension 3: Circulation with lower bound of capacity

INPUT:
a network $G =< V, E >$, where each edge e has a capacity upper bound $C(e)$ and a lower bound $L(e)$; multi sources s_i and sinks t_j. A sink t_j has demand $d_j > 0$, while a source s_i has supply d_i (described as a negative demand $d_i < 0$).
OUTPUT:
a feasible circulation f to satisfy all demand requirements using the available supply, i.e.,

1. Capacity restriction: $L(e) \leq f(e) \leq C(e)$;
2. Conservation restriction: $f^{in}(v) - f^{out}(v) = d_v$;

Note: For the sake of simplicity, we define $d_v = 0$ for any node v except for s_i and t_j. Thus we have $\sum_i d_i = 0$, and define $D = \sum_{d_v > 0} d_v$ be the *total demands* .

An example



[a,b] denotes L(e)=a, and C(e) = b.

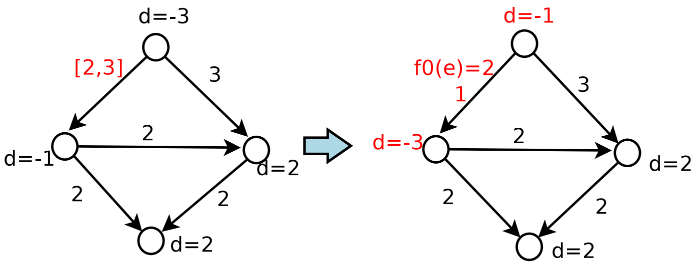
Advantages of lower bound: By setting lower bound $L(e) > 0$, we can force edge e to be used by flow, e.g. edge (s_1, s_2) should be used in the flow.

Algorithm for circulation with lower-bound for capacity

1: Building **an initial flow** f_0 by setting $f_0(e) = L(e)$ for $e = (u, v)$;
2: Solving a new circulation problem for G' without capacity lower bound. Specifically, G' was made by revising an edge $e = (u, v)$ with lower bound capacity:
   1. nodes: $d'_u = d_u + L(e)$, $d'_v = d'_v - L(e)$,
   2. edge: $L(e) = 0$, $C(e) = C(e) - L(e)$.

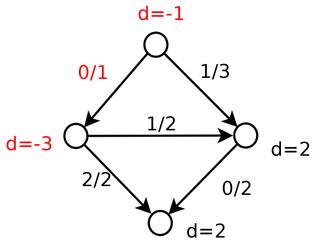
   Denote f' as a feasible circulation to G'.
3: Return $f = f' + f_0$.

Step 1: Building an initial flow f_0



Note: a/[l,b] means f(e) = a, and capacity L(e)=l, and C(e)=b.

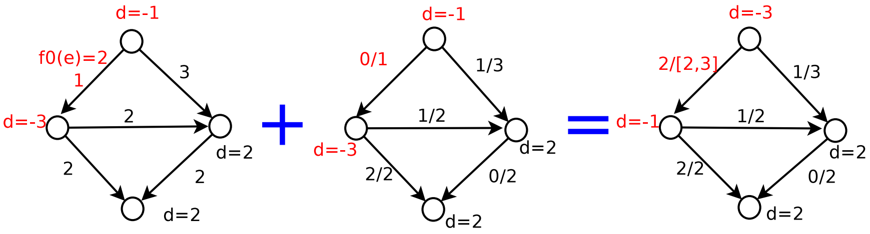
Step 2: Solving the new circulation problem



Note: a/[l,b] means f(e) = a, and capacity L(e)=l, and C(e)=b.

We found a feasible circulation f' for the network G'.

Step 3: Adding f_0 and f'



We get f to the original problem as: $f = f_0 + f'$.

Theorem

There is a circulation f to G (with lower bounds) iff there is a circulation f' to G' (without lower bounds).

Proof.

- Define $f'(e) = f(e) + L_e$.
- It is easy to verify both capacity constraints and conservation constraints hold.

$\square$

Extension 4: MINIMUM COST FLOW problem

Extension 4: Minimum Cost Flow

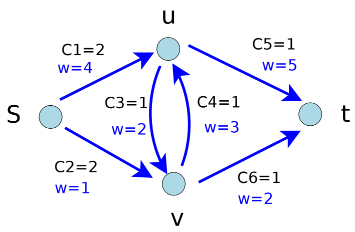
INPUT:

a network $G =< V, E >$, where each edge e has a capacity $C(e) > 0$, and a cost $w(e)$ for transferring a unit through edge e. Two special node: source s and sink t. A flow value v_0.

OUTPUT:

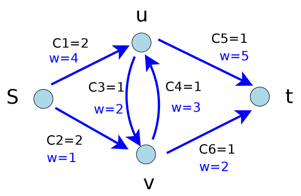
to find a circulation f with flow value v_0 and the cost is minimized.

An example



- Objective: how to transfer $v_0 = 2$ units commodity from s to t with the minimal cost?
- Basic idea: the cost w_e makes it difficult to find the minimal cost flow by simply expanding G to G' as we did for the CIRCULATION problem. Then we return to the primal_dual idea.

Primal_Dual technique: LP formulation



$$
\begin{array}{lcccccccl}
\color{red}{\min} & 4y_1 & +y_2 & +2y_3 & +3y_4 & +5y_5 & +2y_6 & & \\
s.t. & y_1 & +y_2 & & & & & \color{red}{=}2 & \text{node } s \\
 & & & & & -y_5 & -y_6 & \color{red}{=}-2 & \text{node } t \\
 & -y_1 & & +y_3 & -y_4 & +y_5 & & \color{red}{=}0 & \text{node } u \\
 & & -y_2 & -y_3 & +y_4 & & +y_6 & \color{red}{=}0 & \text{node } v \\
 & & & & & & y_i & \leq C_i & \\
 & & & & & & y_i & \geq 0 &
\end{array}
$$

Intuition: y_i denotes the flow on edge i.

Primal_Dual technique: Dual form D

$$
\begin{array}{lrrrrrrll}
\max & -4y_1 & -y_2 & -2y_3 & -3y_4 & -5y_5 & -2y_6 & & \\
s.t. & y_1 & +y_2 & & & & & \leq 2 & \text{node } s \\
 & & & & & -y_5 & -y_6 & \leq -2 & \text{node } t \\
 & -y_1 & & +y_3 & -y_4 & +y_5 & & \leq 0 & \text{node } u \\
 & & -y_2 & -y_3 & +y_4 & & +y_6 & \leq 0 & \text{node } v \\
 & & & & & & y_i & \leq C_i & \\
 & & & & & & y_i & \geq 0 &
\end{array}
$$

Rewrite the LP into standard DUAL form via:

- Objective function: using $\max$ instead of $\min$.
- Constraints: Simply replacing "$=$" with "$\leq$". (Why? Notice that if all inequalities were satisfied, they should be equalities. For example, inequalities (2), (3) and (4) force $y_1 + y_2 \geq 2$, thus change $\leq$ into $=$ for inequality (1). So do other inequalities.

Finding a valid circulation with value v_0 first.

- We need to find a valid circulation with value $v_0 = 2$ first.
- This is easy: $\textsc{Circulation}$ problem.
- Thus we have a feasible solution to D.

Primal_Dual technique: DRP

$$
\begin{array}{lrrrrrrll}
\max & -4y_1 & -y_2 & -2y_3 & -3y_4 & -5y_5 & -2y_6 & & \\
s.t. & y_1 & +y_2 & & & & & \leq 0 & \text{node } s \\
 & & & & & y_5 & +y_6 & \leq 0 & \text{node } t \\
 & y_1 & & -y_3 & +y_4 & -y_5 & & \leq 0 & \text{node } u \\
 & & y_2 & +y_3 & -y_4 & & -y_6 & \leq 0 & \text{node } v \\
 & & & & & & y_i & \leq 0 & \text{for full arc} \\
 & & & & & & -y_i & \leq 0 & \text{for empty arc} \\
 & & & & & & y_i & \leq 1 & \text{for any arc}
\end{array}
$$

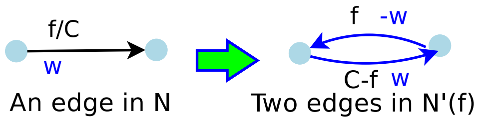
Recall the rules to construct DRP from D:

- Replacing the right hand items with 0.
- Removing the constraints not in J (J contains the constraints in D where $=$ holds).
- Adding constraints $y_i \geq -1$ for any arcs.

Definition (Cycle flow)

A flow f is called **cycle flow** if input equal output for any node (including s and t).

- Suppose we have already obtained a flow for network N.
- Solving the corresponding DRP is essentially finding a cycle in a new network $N'(f)$, which is constructed as follows:
  1. For each edge $e = (u, v)$ in N, two edges $e = (u, v)$ and $e' = (v, u)$ were introduced to $N'(f)$;
  2. The capacities for e and e' in $N'(f)$ are set as $C(e) - f(e)$ and $-f(e)$, respectively;
  3. The costs are set as $w(e') = -w(e)$;



Minimum cost flow algorithm [M. Klein 1967]

Theorem

f is the minimum cost flow in network N $\Leftrightarrow$ network $N'(f)$ contains no cycle with negative cost.

Proof.

f is the minimum cost flow in network N
$\Leftrightarrow$ The optimal solution to DRP is 0.
$\Leftrightarrow$ $N'(f)$ has no cycle flow with negative cost.
$\Leftrightarrow$ $N'(f)$ has no cycle with negative cost. $\square$

Intuition: Suppose that we have obtained a cycle in $N'(f)$. Pushing a unit flow along the cycle leads to a cycle flow (denoted as $\overline{f}$). Then $f + \overline{f}$ is also a flow for the original network N.

Minimum cost flow algorithm

Klein algorithm

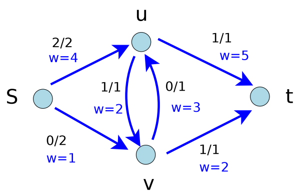
1: Finding a flow f with value v_0 using maximum-flow algorithm, say Ford-Fulkerson;
2: **while** $N'(f)$ contains a cycle C with negative cost **do**
3: Denote b as the bottleneck of cycle C.
4: Define $\overline{f}$ as the unit flow along C.
5: $f = f + b\overline{f}$;
6: **end while**
7: **return** f.

Note:

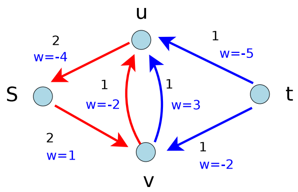
1. The cost of flow decreases as iteration proceeds, while the flow value keeps constant.
2. The cycle with negative cost can be found using Bellman-Ford algorithm.

Example: Step 1

Initial flow f_0: flow value 2, flow cost: 17.



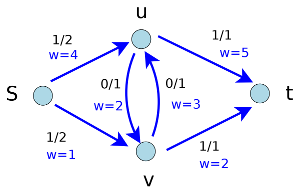
New network $N'(f)$:



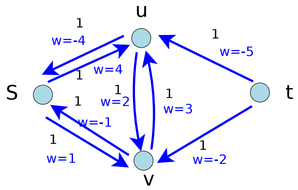
Negative cost cycle: $s \to v \to u \to s$ (in red). Cost: -5.

Example: Step 2

$f = f + \overline{f}$: flow value $2 - 0 = 2$, flow cost: $17 - 5 = 12$.



New network $N'(f)$:



Negative cost cycle: cannot find. Done!

Extension: Hitchcock TRANSPORTATION problem 1941

INPUT: n sources $s_1, s_2, ..., s_n$ and n sinks $t_1, t_2, ..., t_n$. Source s_i has supply a_i, and a sink t_j has demand b_j. The cost from s_i to t_j is c_{ij}.
OUTPUT: arrange a schedule to minimize cost.

Note:

1. Frank L. Hitchcock formulated theTRANSPORTATION problem in 1941. This problem is equivalent to MINIMUM COST FLOW PROBLEM [Wagner, 1959].
2. In 1956, L. R. Ford Jr. and D. R. Fulkerson proposed a "labeling" technique to solve the transportation problem. This algorithm is considerably more efficient than simplex algorithm. See "Solving the Transportation Problem" by L. R. Ford Jr. and D. R. Fulkerson.
3. If $c_{ij} = 0/1$, then Hitchcock problem turns into assignment problem.

Applications of MaximumFlow problem

Formulating a problem into MaximumFlow problem:

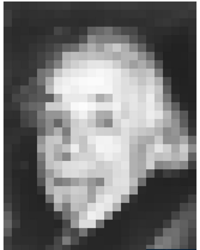
1. We should define a **network** first. Sometimes we need to construct a graph from the very scratch.
2. Then we need to define **weight for edges**. Sometimes we need to move the weight on nodes to edges.
3. How to define **source** s **and sink** t ? Sometimes super source s^* and t^* are needed.
4. Finally we need to prove that **max-flow** (finding paths, matching) or **min-cut** (partition nodes) is what we wanted.

Note: most problems utilize the property that there exists a maximum integer-valued flow iff there exists a maximum flow.

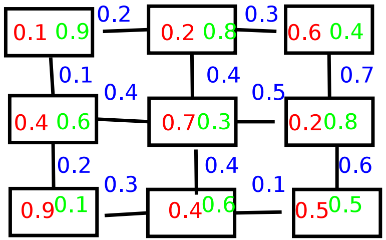
Paradigm 1: Partition a set

Problem 1: ImageSegmentation problem

INPUT:
Given an image in pixel map format. The pixel $i, i \in P$ has a probability to be foreground f_i and the probability to be background b_i; in addition, the likelihood that two neighboring pixels i and j are similar is l_{ij};
GOAL:
to identify foreground out of background. Mathematically, we want a partition $P = F \cup B$, such that $Q(F, B) = \sum_{i\in F} f_i + \sum_{j\in B} b_i + \sum_{i\in F}\sum_{j\in N(i)\cap F} l_{ij} + \sum_{i\in B}\sum_{j\in N(i)\cap B} l_{ij}$ is maximized.

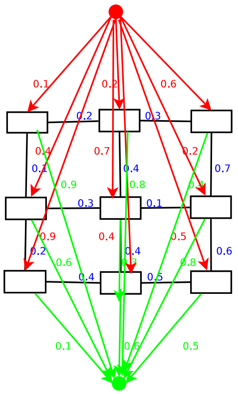


An example



- Red: the probability f_i for pixel i to be foreground;
- Green: the probability b_i for pixel i to be background;
- Blue: the likelihood that pixel i and j are in the same category;

Converting to network-flow problem



1. Network: Adding two nodes source s and sink t with connections to all nodes;
2. Capacity: $C(s,v) = f_v$, $C(v,t) = b_v$; $C(u,v) = l_{uv}$;
3. Cut: a partition. Cut capacity $C(F,B) = M - Q(F,B)$, where $M = \sum_i (b_i + f_i) + \sum_i \sum_j l_{ij}$ is a constant.
4. MinCut: the optimal solution to the original problem

Problem 2: PROJECT SELECTION

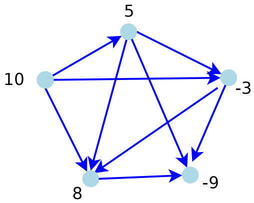
INPUT:

Given a directed acyclic graph (DAG). A node represents a project associated with a profit (denoted as $p_i > 0$) or a cost (denoted as $p_i < 0$), and directed edge $u \to v$ represent the prerequisite relationship, i.e. v should be finished before u.

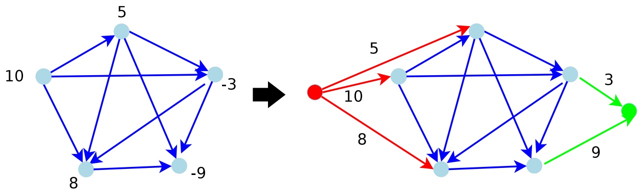
GOAL:

to choose a subset A of projects such that:

1. Feasible: if a project was selected, all its prerequisites should also be selected;
2. Optimal: to maximize profits $\sum_{v \in A} p_v$;



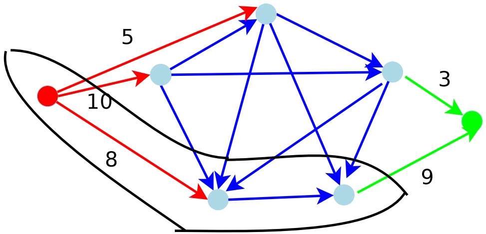
Network construction



1. Network: introducing two nodes: s and t, s connecting the nodes with $p_i > 0$, and t connecting the nodes with $p_i < 0$;
2. Capacity: moving weights from nodes to edges, and set $C(u, v) = \infty$ for $< u, v > \in E$.
3. Cut: a partition of nodes.

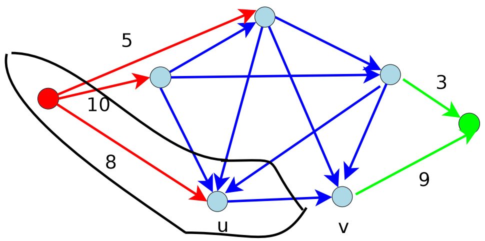
Minimum cut corresponds to maximum profit

1. Cut capacity: $C(A,B) = C - \sum_{i \in A} p_i$, where $C = \sum_{v \in V} p_v$ ($p_v > 0$) is a constant.



2. In the example, $C(A,B) = 5 + 10 + 9$, $\sum_{i \in A} p_i = 8 - 9$, and $C = 5 + 10 + 8$.
3. Min-Cut: corresponding to the maximum profit since the sum of cut capacity and profit is a constant.

Feasibility

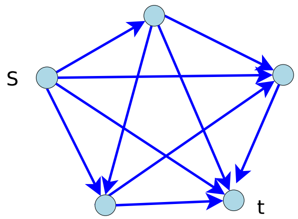


- Feasible: The feasibility is implied by the infinite weights on edges, i.e. an invalid selection corresponds to a cut with infinite capacity.
- For example, if a project u was selected while its precursor v was not selected, then the edge $< u, v >$ is a cut edge, leading to an infinite cut.

Paradigm 2: Finding paths

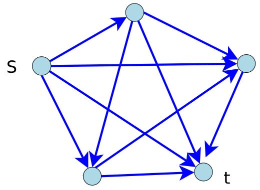
Problem 3: Disjoint paths

INPUT:
Given a graph $G =< V, E >$, two nodes s and t, an integer k.
GOAL:
to identify k $s - t$ paths whose edges are disjoint;

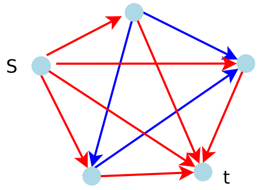


Related problem: graph connectivity

Network construction



1. Edges: the same to the original graph;
2. Capacity: $C(u, v) = 1$;
3. Flow: (See extra slides)



Theorem

k disjoint paths in G $\Leftrightarrow$ the maximum $s-t$ flow value is at least k.

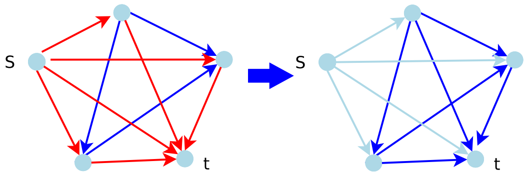
Proof.

1. Note: maximum $s-t$ flow value is k implies an INTEGRAL flow with value k.
2. Simply selecting the edges with $f(e)=1$.

$\square$

Time-complexity: $O(mn)$.

Menger theorem 1927



Theorem

The number of maximum disjoint paths is equal to the number of minimal edge removement to separate s from t.

Menger theorem

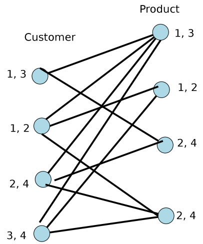
Proof.

1. The number of maximum disjoint paths is equal to the maximum flow;
2. Then there is a cut (A, B) such that $C(A, B)$ is the number of disjoint paths;
3. The cut edges are what we wanted.

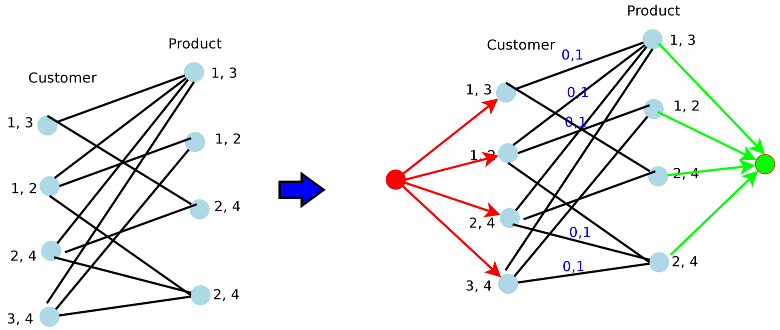
$\square$

Problem 4: Survey design

INPUT:
A set of customers A, and a set of products P. Let $B(i) \subseteq P$ denote the products that customer i bought. An integer k.
GOAL:
to design a survey with k questions such that for customer i, the number of questions is at least c_i but at most c'_i. On the other hand, for each product, the number of questions is at least p_i but at most p'_i.



Network construction

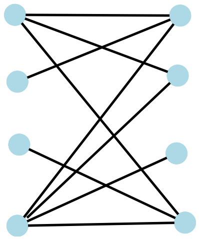


1. Edges: introducing two nodes s and t. Connecting customers with s and products with t.
2. Capacity: moving weights from nodes to edges; setting $C(i, j) = 1$;
3. Circulation: is a feasible solution to the original problem.

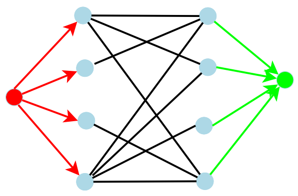
Paradigm 3: Matching

Problem 5: Matching

INPUT:
A bipartite $G = < V, E >$;
GOAL:
to identify the maximal matching;

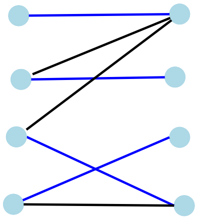


Constructing a network



1. Edges: adding two nodes s and t; connecting s with U and t with V;
2. Capacity: $C(e) = 1$ for all $e \in E$;
3. Flow: the maximal flow corresponds to a maximal matching;

Time-complexity: $O(mn)$



Perfect match

Definition (Perfect match)

Given a bipartite $G =< V, E >$, where $V = X \cup Y$, $X \cap Y = \phi$, $|X| = |Y| = n$. A match M is a perfect match iff $|M| = n$.

Theorem

A bipartite has a perfect matching $\Leftrightarrow$ *for any* $A \subseteq X$, $|\Gamma(A)| \geq |A|$, *where* $\Gamma(A)$ *denotes the partners of nodes in* A.



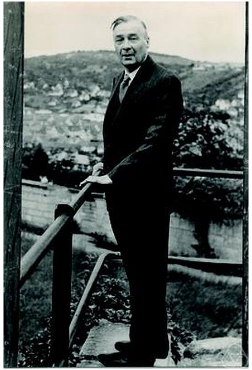
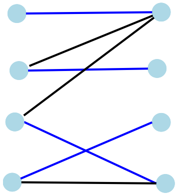
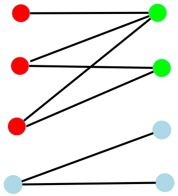


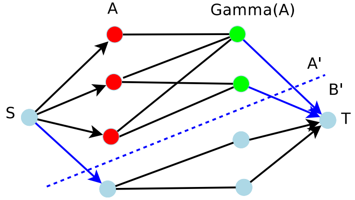
Figure: Konig, Egervary, and Philip Hall



Perfect match



No perfect match



No perfect match

Proof.

Here we only show that if there is no perfect matching, then $|\Gamma(A)| < |A|$.

1. Suppose there is no perfect matching, i.e., the maximal match is M, $|M| < n$;

Paradigm 4: Decomposing numbers

INPUT:
n teams $T_1, T_2, ..., T_n$. A team T_i has already won w_i games, and for team T_i and T_j, there are g_{ij} games left.
GOAL:
Can we determine whether a team, say T_i, has already been eliminated from the first place? If yes, can we give an evidence?

An example

Four teams: *New York, Baltimore, Toronto, Boston*

1. w_i: NY (90), Balt (88), Tor (87), Bos (79).
2. g_{ij}: NY:Balt 1, NY:Tor 6, Balt:Tor 1, Balt:Bos 4, Tor:Bos 4, NY:Bos 4.

It is safe to say that $Boston$ has already been eliminated from the first place since:

1. $Boston$ can finish with at most $79 + 12 = 91$ wins.
2. We can find a subset of teams, e.g. $\{NY, Tor\}$, with the total number of wins of 90+87+6 $=$ 183, thus at least a team finish with $\frac{183}{2} = 91.5 > 91$ wins.

Note that $\{NY, Tor, Balt\}$ cannot serve as an evidence that Bos has already been eliminated.

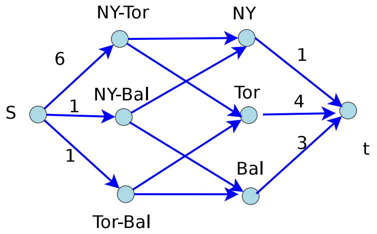
Question: For a specific team z. Can we determine whether there exists a subset of teams $S \subseteq T - \{z\}$ such that

1. z can finish with at most m wins;
2. $\frac{1}{|S|}(\sum_{x \in S} w_x + \sum_{x,y \in S} g_{xy}) > m$.

In other word, at least one of the teams in S will have more wins than z.

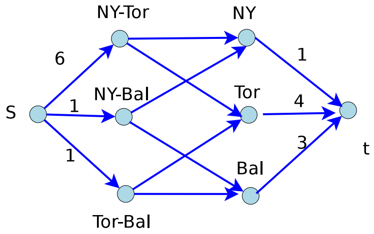
Network construction: taking $z = Boston$ as an example

- We define $m = w_z + \sum_{x \in T} g_{xz} = 91$, i.e. the total number of possible wins for team z.
- A network is constructed as follows:
  1. Define $S = T - \{z\}$, and $g^* = \sum_{x,y \in S} g_{xy} = 8$.
  2. Nodes: For each pair of teams, constructing a node $x : y$, and for each team x, constructing a node x.
  3. Edges:
     - For edge $s - x : y$, set capacity as $g_{x,y}$.
     - For edge $x : y - x$ and $x : y - y$, set capacity as $g_{x,y}$.
     - For edge $x - t$, set capacity as $m - w_x$.

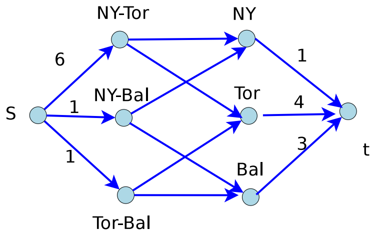


Intuition: number decomposition

Intuition: along edge $s - x : y$, we send $g_{x,y}$ wins, and at node $x : y$, this number is decomposed into two numbers, i.e. the number of wins of each team.

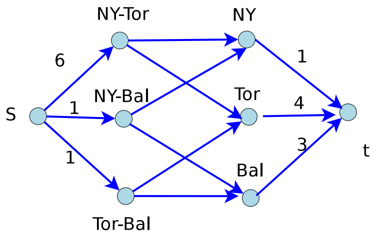


Case 1: the maximum flow value is $g* = 8$



Theorem

There exist a flow with value $g^* = 8$ iff there is still possibility that $z = Boston$ wins the championship.

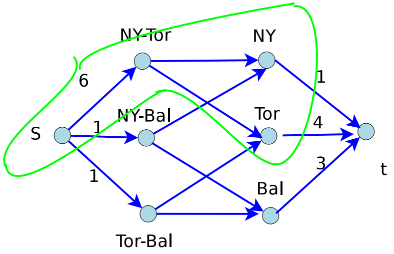


Proof.

- $\Leftarrow$
  - If there is a flow with value g^*, then the capacities on edges $x - t$ guarantees that no team can finish with over m wins.
  - Therefore, z still have chance to win the championship (if z wins all remaining games).
- $\Leftarrow$
  - If there is possibility for z to win the championship
  - we can define a flow with value g^*.

□

Case 2: the maximum flow value is less than $g* = 8$



Theorem

If the maximum flow value is strictly smaller than g^*, the minimum cut describes a subset $S \subseteq T - \{z\}$ such that $\frac{1}{|S|}(\sum_{x \in S} w_x + \sum_{x,y \in S} g_{xy}) > m$.

Proof.

(See extra slides) $\square$

Extensions of matching: ASSIGNMENT problem, Hungarian algorithm for WEIGHTED ASSIGNMENT problem, Blossom algorithm.