Network Optimization: Quick Start

@luk036

2022-11-09

Introduction

Why and why not

- Algorithms are available for common network problems (Python: networkx, C++: Boost Graph Library (BGL)):
  - Explore the locality of network.
  - Explore associativity (things can be added up in any order)
- Be able to solve discrete problems optimally (e.g. matching/assignment problems)
- Bonus: gives you insight into the most critical parts of the network (critical cut/cycle)
- The theory is hard to understand.
- Algorithms are hard to understand (some algorithms do not allow users to have an input flow in reverse directions, but create edges internally for the reverse flows).
- There are too many algorithms available. You have to choose them wisely.

Flow and Potential

- Cut
- Current
- Flow x
- Sum of x_{ij} around a node = 0

- Cycle/Path
- Voltage
- Tension y
- Sum of y_{ij} around a cycle = 0

If you don't know more…

- For the min-cost linear flow problem, the best guess is to use the "network simplex algorithm".
- For the min-cost linear potential problem: formulate it as a dual (flow) problem.

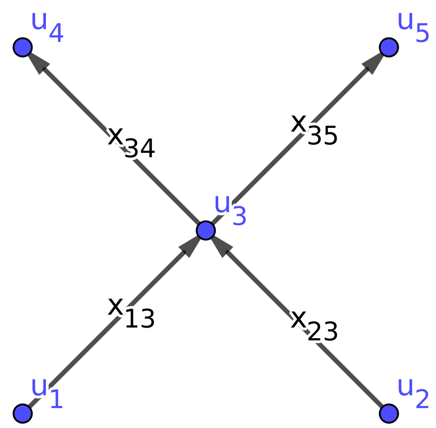

Figure 1: flow

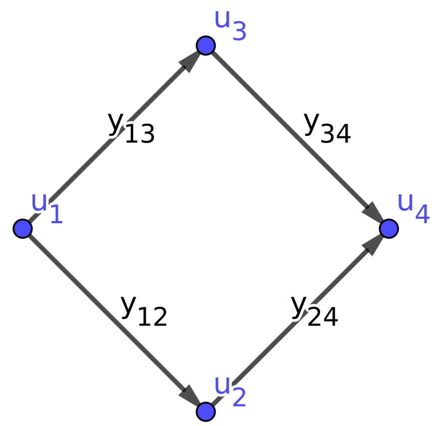

Figure 2: potential

- For the parametric potential problem (single parameter), the best guess is to use Howard's algorithm.
- All these algorithms are based on the idea of finding "negative cycle".
- You can apply the same principle to the nonlinear problems.

For dual problems…

- Dual problems can be solved by applying the same principle.
- Finding negative cycles is replaced by finding a negative "cuts", which is more difficult…
- …unless your network is a planar graph.

Guidelines for the average users

- Look for specialized algorithms for specialized problems. For example, for bipartite maximum cardinality matching, use the Hopcroft-Karp matching algorithm.
- Avoid creating edges with infinite costs. Delete them or reformulate your problem.

Guidelines for algorithm developers

- Make "negative cycles" as orthogonal to each other as possible.
- Reuse previous solutions as a new starting point for finding negative cycles.

Essential Concepts

Basic elements of a network

Definition (network) A *network* is a collection of finite-dimensional vector spaces, which includes *nodes* and *edges/arcs*:

- $V = \{v_1, v_2, \cdots, v_N\}$, where $|V| = N$
- $E = \{e_1, e_2, e_3, \cdots, e_M\}$ where $|E| = M$

which satisfies 2 requirements:

1. The boundary of each edge is comprised of the union of nodes
2. The intersection of any edges is either empty or the boundary node of both edges.

Network

- By this definition, a network can contain self-loops and multi-edges.

- A *graph* structure encodes the neighborhood information of nodes and edges.
- Note that Python's NetworkX requires special handling of multi-edges.
- The most efficient graph representation is an adjacency list.
- The concept of a graph can be generalized to *complex*: node, edge, face…

Types of graphs Bipartite graphs, trees, planar graphs, st-graphs, complete graphs.

Orientation

Definition (Orientation) An *orientation* of an edge is an ordering of its boundary node (s, t), where

- s is called a source/initial node
- t is called a target/terminal node

Note: orientation != direction

Definition (Coherent) Two orientations to be the same is called *coherent*

Node-edge Incidence Matrix (connect to algebra!)

Definition (Incidence Matrix) An $N \times M$ matrix $A^\intercal$ is a node-edge incidence matrix with entries:

$$A(i,j) = \begin{cases} +1 & \text{if } e_i \text{ is coherent with the orientation of node } v_j, \\ -1 & \text{if } e_i \text{ is not coherent with the orientation of node } v_j, \\ 0 & \text{otherwise.} \end{cases}$$

Example

$$A^\intercal = \begin{bmatrix} 0 & -1 & 1 & 1 & 0 \\ 1 & 1 & 0 & -1 & -1 \\ -1 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Chain

Definition (Chain τ) An edge/node *chain* τ is an M/N-tuple of scalar that assigns a coefficient to each edge/node, where M/N is the number of distinct edges/nodes in the network.

Remark (II) A chain may be viewed as an (oriented) indicator vector representing a set of edges/nodes.

Example (II) $[0, 0, 1, 1, 1]$, $[0, 0, 1, -1, 1]$

Discrete Boundary Operator

Definition (Boundary operator) The *boundary* operator $\partial = A^{\mathsf{T}}$.

Definition (Cycle) A chain is said to be a *cycle* if it is in the null-space of the boundary operator, i.e. $A^{\mathsf{T}}\tau = 0$.

Definition (Boundary) A chain β is said to be a *boundary* of τ if it is in the range of the boundary operator.

Co-boundary Operator d

Definition (Co-boundary operator) The *co-boundary* (or *differential*) operator $\mathrm{d} = \partial^* = (A^{\mathsf{T}})^* = A$

Note Null-space of A is #components of a graph

Discrete Stokes' Theorem

- Let
$$\tau_i = \begin{cases} 1 & \text{if } e_i \in S, \\ 0 & \text{otherwise.} \end{cases}$$
- Conventional (integration):
$$\int_S \mathrm{d}\tilde{\omega} = \oint_{\partial S} \tilde{\omega}$$
- Discrete (pairing):
$$[\tau, A\omega] = [A^{\mathsf{T}}\tau, \omega]$$

Fundamental Theorem of Calculus

- Conventional (integration): $\int_a^b f(t)dt = F(b) - F(a)$
- Discrete (pairing): $[\tau_1, Ac^0] = [A^{\mathsf{T}}\tau_1, c^0]$

Divergence and Flow

Definition (Divergence) $\operatorname{div} x = A^{\mathsf{T}} x$

Definition (Flow) x is called a *flow* if $\sum \operatorname{div} x = 0$, where all negative entries of (div x) are called *sources* and positive entries are called *sinks*.

Definition (Circulation) A network is called a *circulation* if there is no source or sink. In other words, $\operatorname{div} x = 0$

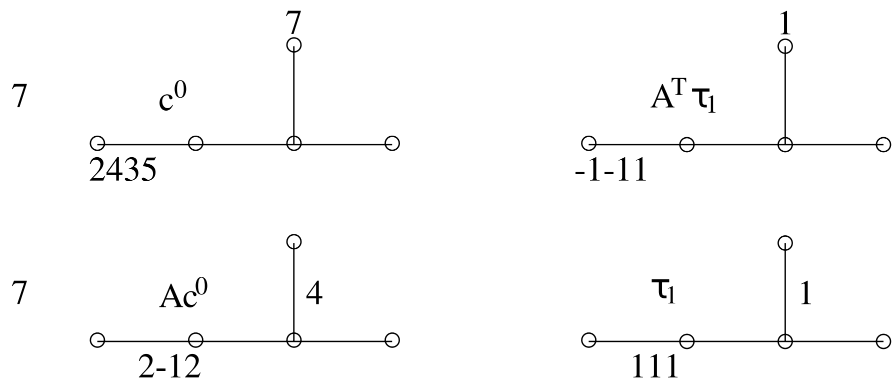

Figure 3: stokes

Tension and Potential

Definition (Tension) A tension (in co-domain) y is a *differential* of a *potential* u, i.e. $y = Au$.

Theorem (Tellgen's) Flow and tension are bi-orthogonal (isomorphic).

Proof $0 = [A^\mathsf{T} x, u] = (A^\mathsf{T} x)^\mathsf{T} u = x^\mathsf{T}(Au) = x^\mathsf{T} y$

Path

A path indicator vector τ of P that

$$\tau_i = \begin{cases} 1 & \text{if } e_i \in P, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem [total tension y on P] = [total potential on the boundary of P].

Proof $y^\mathsf{T}\tau = (Au)^\mathsf{T}\tau = u^\mathsf{T}(A^\mathsf{T}\tau) = u^\mathsf{T}(\partial P)$.

Cut

Two node sets S and S' (the complement of S, i.e. $V - S$). A cut Q is an edge set, denoted by $[S, S']^-$. A cut indicator vector q (oriented) of Q is defined as Ac where

$$c_i = \begin{cases} 1 & \text{if } v_i \in S\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Theorem (Stokes' theorem!) [Total divergence of x on S] = [total x across Q].

Proof $(\operatorname{div} x)^\mathsf{T} c = (A^\mathsf{T} x)^\mathsf{T} c = x^\mathsf{T}(Ac) = x^\mathsf{T} q$.

Examples

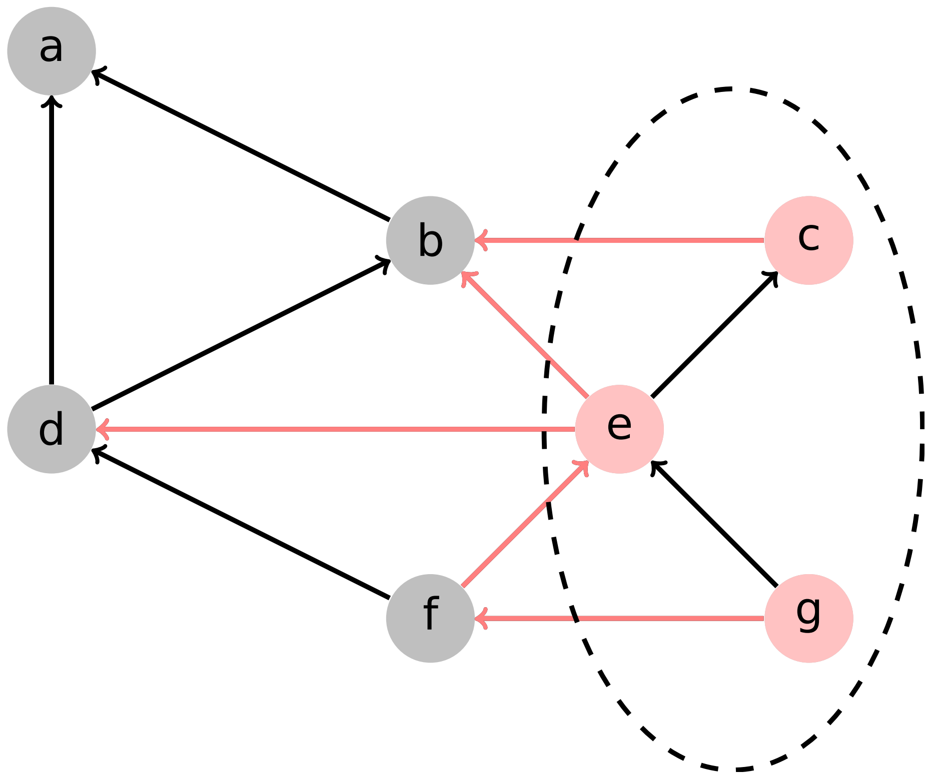

Figure 4: cut

Feasibility Problems

Feasible Flow/Potential Problem

Feasible Flow Problem

- Find a flow x such that:

$$\begin{aligned} & c^- \leq x \leq c^+, \\ & A^\mathsf{T} x = b, b(V) = 0. \end{aligned}$$

- Can be solved using:
  - Painted network algorithm
  - If no feasible solution, return a "negative cut".

Feasible Potential Problem:

- Find a potential u such that:

$$\begin{aligned} d^- &\leq y \leq d^+ \\ A \cdot u &= y. \end{aligned}$$

- Can be solved using:
  - Bellman-Ford algorithm
  - If no feasible solution, return a “negative cycle”.

Examples

Genome-scale reaction network (primal)

- A: Stoichiometric matrix S
- x: reactions between metabolites/proteins
- $c^- \leq x \leq c^+$: constraints on reaction rates

Timing constraints (co-domain)

- A^T: incidence matrix of timing constraint graph
- u: arrival time of clock
- y: clock skew
- $d^- \leq y \leq d^+$: setup- and hold-time constraints

Feasibility Flow Problem

Theorem (feasibility flow) The problem has a feasible solution if and only if $b(S) \leq c^+(Q)$ for all cuts $Q = [S, S']$ where $c^+(Q) =$ upper capacity [1, p. 56].

Proof (if-part)

Let $q = A \cdot k$ be a cut vector (oriented) of Q. Then

- $c^- \leq x \leq c^+$
- $q^\mathsf{T} x \leq c^+(Q)$
- $(A \cdot k)^\mathsf{T} x \leq c^+(Q)$
- $k^\mathsf{T} A^\mathsf{T} x \leq c^+(Q)$
- $k^\mathsf{T} b \leq c^+(Q)$
- $b(S) \leq c^+(Q)$

Feasibility Potential Problem

Theorem (feasibility potential) The problem has a feasible solution if and only if $d^+(P) \geq 0$ for all cycles P where $d^+(P) =$ upper span [1, p. ??].

Proof (if-part)

Let τ be a path indicator vector (oriented) of P. Then

- $d^- \leq y \leq d^+$
- $\tau^\mathsf{T} y \leq d^+(P)$
- $\tau^\mathsf{T}(A \cdot u) \leq d^+(P)$
- $(A^\mathsf{T}\tau)^\mathsf{T} u \leq d^+(P)$
- $(\partial P)^\mathsf{T} u \leq d^+(P)$
- $0 \leq d^+(P)$

Remarks

- The only-if part of the proof is constructive. It can be done by constructing an algorithm to obtain the feasible solution.
- d^+ could be ∞ or zero, etc.
- d^- could be $-\infty$ or zero, etc.
- c^+ could be ∞ or zero, etc.
- c^- could be $-\infty$ or zero, etc.

Note: most tools require that c^- must be zero such that the solution flow x is always positive.

Convert to the elementary problem

- By splitting every edge into two, the feasibility flow problem can reduce to an elementary one:
  - Find a flow x such that

$$
\begin{aligned}
&c \leq x, \\
&A_1^\mathsf{T} x = b_1, \\
&b_1(V_1) = 0.
\end{aligned}
$$

  where A_1 is the incident matrix of the modified network.

Original:

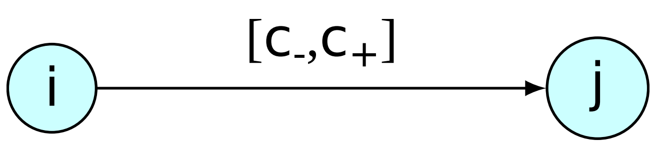

Figure 5: original

Modified:

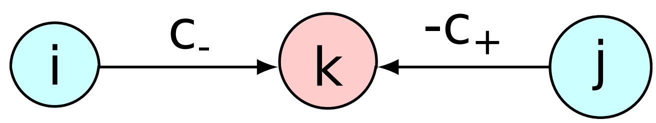

Figure 6: modified

Convert to the elementary problem

- By adding a reverse edge for every edge, the feasibility potential problem can reduce to an elementary one:
  - Find a potential u such that

$$
\begin{array}{l}
y_2 \leq d, \\
A_2 u = y_2
\end{array}
$$

  where A_2 is the incident matrix of the modified network.

Original:

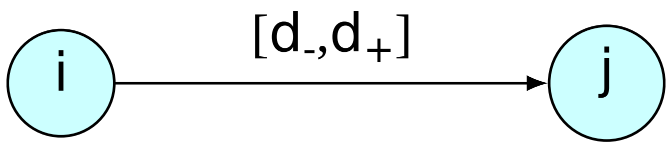

Figure 7: original2

Modified:

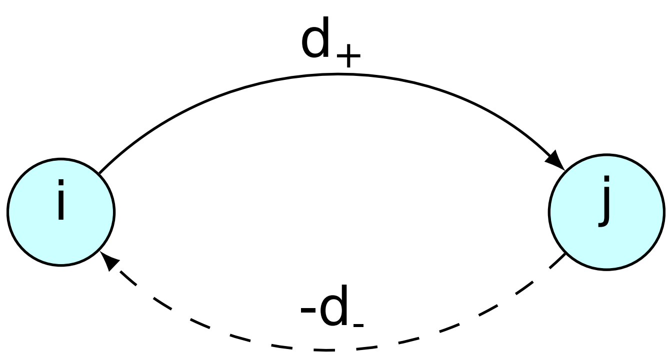

Figure 8: modified2

]

Basic Bellman-Ford Algorithm

```
function BellmanFord(list vertices, list edges, vertex source)
   // Step 1: initialize graph
   for each vertex i in vertices:
       if i is source then u[i] := 0
       else u[i] := inf
       predecessor[i] := null

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge (i, j) with weight d in edges:
           if u[j] > u[i] + d[i,j]:
               u[j] := u[i] + d[i,j]
               predecessor[j] := i

   // Step 3: check for negative-weight cycles
   for each edge (i, j) with weight d in edges:
       if u[j] > u[i] + d[i,j]:
           error "Graph contains a negative-weight cycle"
   return u[], predecessor[]
```

Example 1 : Clock skew scheduling

- Goal: intentionally assign an arrival time u_i to each register so that the setup and hold time constraints are satisfied.
- Note: the clock skew $s_{ij} = u_i - u_j$ is more important than the arrival time u itself, because the clock runs periodically.
- In the early stages, fixing the timing violation could be done as soon as a negative cycle is detected. A complete timing analysis is unnecessary at this stage.

Example 2 : Delay padding + clock skew scheduling

- Goal: intentionally "insert" a delay p so that the setup and hold time constraints are satisfied.
- Note that a delay can be "inserted" by swapping a fast transistor into a slower transistor.
- Traditional problem formulation: Find p and u such that

$$
\begin{aligned}
&y \leq d + p, \\
&Au = y, p \geq 0
\end{aligned}
$$

- Note 1: Inserting delays into some local paths may not be allowed.

- Note 2: The problem can be reduced to the standard form by modifying the network (timing constraint graph)

Four possible ways to insert delay

- No delay:

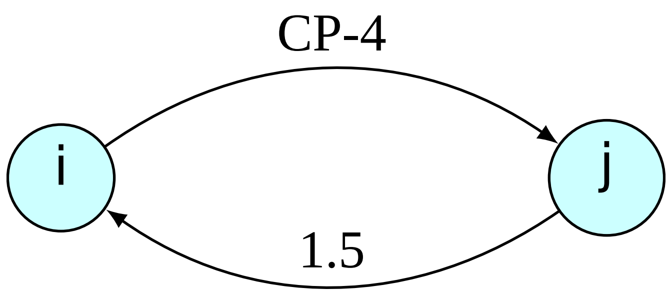

Figure 9: no_delay

- $p_s = p_h$:

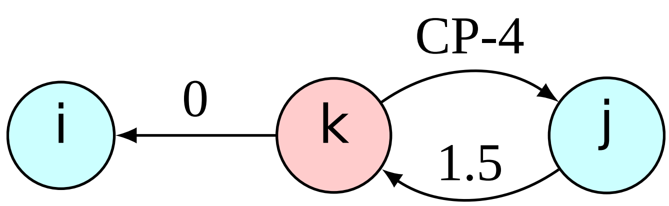

Figure 10: same_delay

- Independent:
- $p_s \geq p_h$:

Remarks (III)

- If there exists a negative cycle, it means that timing cannot be fixed using simply this technique.
- Additional constraints, such as $p_s \leq p_{\max}$, can be imposed.

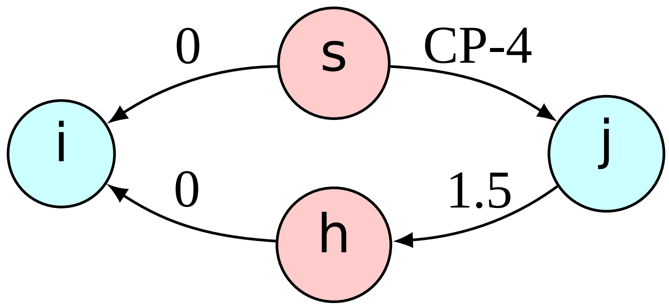

Figure 11: independent

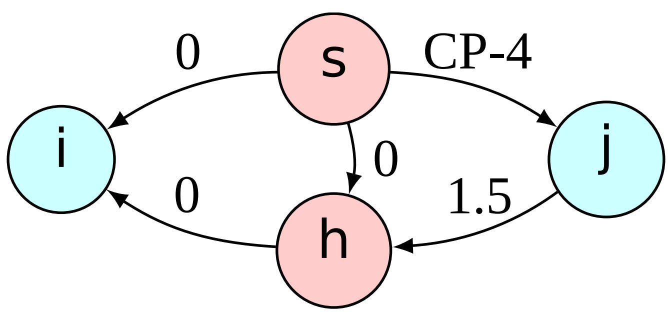

Figure 12: setup_greater

Parametric Problems

Parametric Potential Problem (PPP)

- Consider a parameter potential problem:

$$\begin{array}{ll} \text{maximize} & \beta \\ \text{subject to} & y \leq d(\beta), \\ & A \cdot u = y \end{array}$$

  where $d(\beta)$ is a *monotonic decreasing* function.

- If $d(\beta)$ is a linear function $(m - s\beta)$ where s is non-negative, the problem reduces to the well-known *minimum cost-to-time ratio problem*.

- If s = constant, it further reduces to the *minimum mean cycle problem*.

Note: Parametric flow problem can be defined similarly.

Examples (III)

- $d(\beta)$ is linear $(m - s\beta)$:
  - Optimal clock period scheduling problem
  - Slack maximization problem
  - Yield-driven clock skew scheduling (Gaussian)
- $d(\beta)$ is non-linear:
  - Yield-driven clock skew scheduling (non-Gaussian)
  - Multi-domain clock skew scheduling

Examples (IV)

- Lawler's algorithm (binary search based)
- Howard's algorithm (cycle cancellation)
- Young's algorithm (path based)
- Burns' algorithm (path based)
  - for clock period optimization problem (all elements of s are either 0 or 1)
- Several hybrid methods have also been proposed

Remarks (IV)

- Need to solve feasibility problems many times.
- Data structures, such as Fibonacci heap or spanning tree/forest, can be used to improve efficiency

- For multi-parameter problems, the *ellipsoid method* can be used.
- Example 1: yield-driven clock skew scheduling (c.f. lecture 5)

Example 2: yield-driven delay padding

- The problem can be reduced to the standard form by modifying the underlying constraint graph.

Four possible way to insert delay

- No delay:

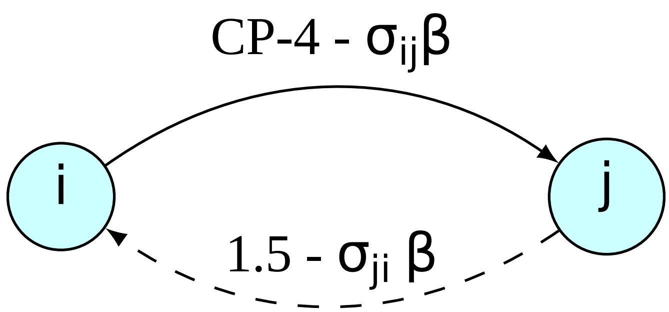

Figure 13: no_delay_s

- $p_s = p_h$:

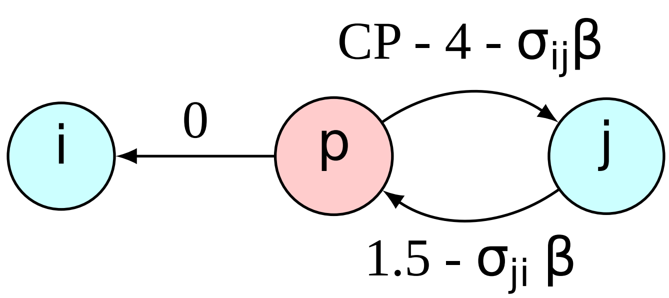

Figure 14: same_delay_s

- Independent:
- $p_s \geq p_h$:

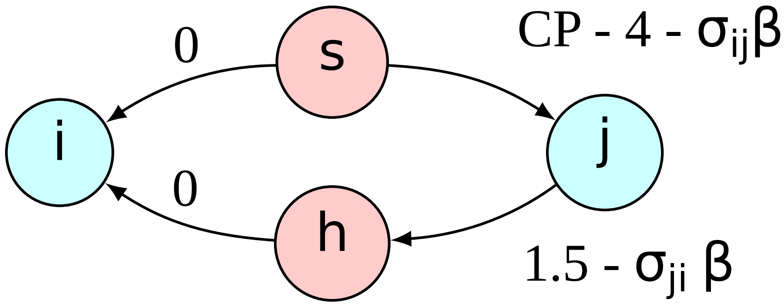

Figure 15: independent_s

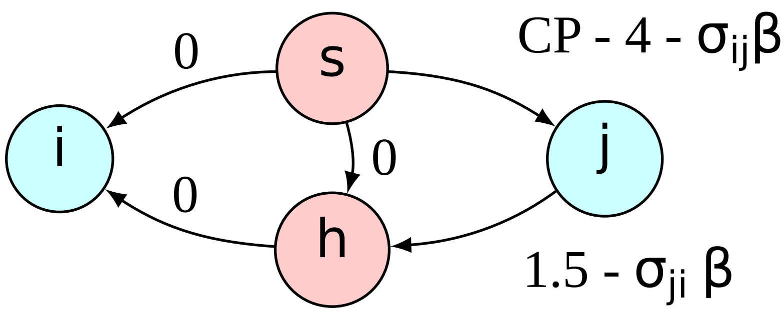

Figure 16: setup_greater_s

Min-cost Flow/Potenial Problem

Elementary Optimal Problems

- Elementary Flow Problem:

$$\begin{array}{ll} \min & d^\mathsf{T} x + p \\ \text{s. t.} & c \le x, \\ & A^\mathsf{T} x = b,\ b(V) = 0 \end{array}$$

- Elementary Potential Problem:

$$\begin{array}{ll} \max & b^\mathsf{T} u - (c^\mathsf{T} y + q) \\ \text{s. t.} & y \le d, \\ & Au = y \end{array}$$

Elementary Optimal Problems (Cont'd)

- The problems are dual to each other if $p + q = -c^\mathsf{T} d, (x - c)^\mathsf{T}(d - y) = 0, c \le x, y \le d$
- Since $b^\mathsf{T} u = (A^\mathsf{T} x)^\mathsf{T} u = x^\mathsf{T} Au = x^\mathsf{T} y$, $[\min] - [\max] = (d^\mathsf{T} x + p) - (b^\mathsf{T} u - [c^\mathsf{T} y + q]) = d^\mathsf{T} x + c^\mathsf{T} y - x^\mathsf{T} y + p + q = (x - c)^\mathsf{T}(d - y) \ge 0$
- $[\min] - [\max]$ when equality holds.

Remark (V)

- We can formulate a linear problem in primal or dual form, depending on which solution method is more appropriate:
  - Incremental improvement of feasible solutions
  - Design variables are in the integral domain:
    * The max-flow problem (i.e. $d^\mathsf{T} = [-1, -1, \cdots, -1]^\mathsf{T}$) may be better solved by the dual method.

Linear Optimal Problems

- Optimal Flow Problem:

$$\begin{array}{ll} \min & d^\mathsf{T} x + p \\ \text{s. t.} & c^- \le x \le c^+, \\ & A^\mathsf{T} x = b,\ b(V) = 0 \end{array}$$

- Optimal Potential Problem:

$$\begin{array}{ll} \max & b^\mathsf{T} u - (c^\mathsf{T} y + q) \\ \text{s. t.} & d^- \le y \le d^+, \\ & Au = y \end{array}$$

Linear Optimal Problems (II)

By modifying the network:

- The problem can be reduced to the elementary case [pp.275-276]

piece of cake

- Piece-wise linear convex cost can be reduced to this linear problem [p.239,p.260]

The problem has been extensively studied and has numerous applications.

Remark (VI)

- We can transform the cost function to be non-negative by reversing the orientation of the negative cost edges.
- Then reduce the problem to the elementary case (or should we???)

Algorithms for Optimal Flow Problems

- Successive shortest path algorithm
- Cycle cancellation method
  - Iteratively insert additional minimal flows according to a negative cycle of the residual network until no negative cycles are found.
- Scaling method

For Special Cases

- Max-flow problem ($d = -[1, \cdots, 1]$)
  - Ford-Fulkerson algorithm: iteratively insert additional minimal flows according to an augmented path of the residual network, until no augmented paths of the residual network are found.
  - Pre-flow Push-Relabel algorithm (dual method???)
- Matching problems ($[c^-, c^+] = [0, 1]$)
  - Edmond's blossom algorithm

Min-Cost Flow Problem (MCFP)

- Problem Formulation:

$$\begin{array}{ll} \min & d^\mathsf{T} x \\ \text{s. t.} & 0 \leq x \leq c, \\ & A^\mathsf{T} x = b,\ b(V) = 0 \end{array}$$

- Algorithm idea: descent method: given a feasible x_0, find a better solution $x_1 = x_0 + \alpha p$, where α is positive.

General Descent Method

- **Input**: $f(x)$, initial x
- **Output**: optimal opt x^*
- **while** not converged,
    1. Choose descent direction p;
    2. Choose the step size α;
    3. $x := x + \alpha p$;

Some Common Descent Directions

- Gradient descent: $p = -\nabla f(x)^\mathsf{T}$
- Steepest descent:
    - $\triangle x_{nsd} = \text{argmin}\{\nabla f(x)^\mathsf{T} v \mid \|v\| = 1\}$
    - $\triangle x_{sd} = \|\nabla f(x)\| \triangle x_{nsd}$ (un-normalized)
- Newton's method: $p = -\nabla^2 f(x)^{-1} \nabla f(x)$
- For convex problems, must satisfy $\nabla f(x)^\mathsf{T} p < 0$.

Note: Here, there is a natural way to choose p!

Min-Cost Flow Problem (II)

- Let $x_1 = x_0 + \alpha p$, then we have:

$$\begin{array}{lll} \min & d^\mathsf{T} x_0 + \alpha d^\mathsf{T} p & \Rightarrow d^\mathsf{T} p < 0 \\ \text{s. t.} & -x_0 \le \alpha p \le c - x_0 & \Rightarrow \text{residual graph} \\ & A^\mathsf{T} p = 0 & \Rightarrow p \text{ is a cycle!} \end{array}$$

- In other words, choose p to be a negative cycle!
    - Simple negative cycle, or
    - Minimum mean cycle

Primal Method for MCFP

- **Input**: $G(V, E), [c^-, c^+], d$
- **Output**: optimal opt x^*
- Initialize a feasible x and certain data structure
- **while** a negative cycle p found in $G(x)$,
    1. Choose a step size α;
    2. **If** α is unbounded, **return** UNBOUNDED;
    3. **If** $\alpha = 0$, **break**;
    4. $x := x + \alpha p$;
    5. Update corresponding data structures
- **return** OPTIMAL

Remarks (VI)

- In Step 4, negative cycle can be found using Bellman-Ford algorithm.
- In the cycle cancelling algorithm, p is:
  - a simple negative cycle, or
  - a minimum mean cycle
- A heap or other data structures are used for finding negative cycles efficiently.
- Usually α is chosen such that one constraint is tight.

Min-Cost Potential Problem (MCPP)

- Problem Formulation:
$$\begin{array}{ll} \min & c^{\mathsf{T}} y \\ \text{s. t.} & y \leq d, \\ & Au = y \end{array}$$
  where c is assumed to be non-negative.
- Algorithm: given an initial feasible u_0, find a better solution $u_1 = u_0 + \beta q$, where β is positive:
$$\begin{array}{lll} \min & c^{\mathsf{T}} y_0 + c^{\mathsf{T}} y & \Rightarrow c^{\mathsf{T}} y < 0 \\ \text{s. t.} & y \leq d - Au_0 & \Rightarrow \text{residual graph} \\ & \beta A q = y & \Rightarrow q \text{ is a ``cut''!} \end{array}$$

Method for MCPP

- **Input**: $G(V, E), c, d$
- **Output**: optimal opt u^*
- Initialize a feasible u and certain data structure
- **while** a negative cut q found in $G(u)$,
  1. Choose a step size β;
  2. **If** β is unbounded, **return** UNBOUNDED;
  3. **If** $\beta = 0$, **break**;
  4. $u := u + \beta q$;
  5. Update corresponding data structures
- **return** OPTIMAL

Remarks (VII)

- Usually β is chosen such that one constraint is tight.
- The min-cost potential problem is the dual of the min-cost flow problem, so algorithms can solve both problems.
- In the network simplex method, q is chosen from a spanning tree data structure (for linear problems only)