La distance n'y fait rien; il n'y a que le premier pas qui coûte.
[The distance is nothing; it is only the first step that costs.]

— Marie Anne de Vichy-Chamrond, marquise du Deffand,
letter to Jean le Rond d'Alembert, July 7, 1763

Tam quaestiones altioris indaginis poscuntur.
[Then these questions require further investigation.]

— Carl Gustav Jacob Jacobi, unpublished notes (c. 1836)

Cecil Graham: *What is a cynic?*
Lord Darlington: *A man who knows the price of everything, and the value of nothing.*
Cecil Graham: *And a sentimentalist, my dear Darlington, is a man who sees an absurd value in everything and doesn't know the market price of any single thing.*

— Oscar Wilde, *Lady Windermere's Fan, A Play About a Good Woman* (1892)

G

Minimum-Cost Flows

[Read Chapters 8, 10, 11, and F first.]
Status: Text beta. Needs figures.

In this final chapter on flows, we consider a significant generalization of the maximum-flow problem that (as far as we know) cannot be solved by modifying the graph and applying a standard flow algorithm. The input to our problem consists of a flow network without special source and target vertices, where each edge e has a ***cost*** $\$(e)$, in addition to the usual edge capacities and vertex balances. Edge costs can be positive, negative, or zero. The total cost of any flow f is then defined as

$$\$(f) = \sum_{e \in E} \$(e) \cdot f(e).$$

The ***minimum-cost flow*** problem asks for a feasible flow with minimum cost, instead of a feasible flow with maximum value.

G.1 Minimum-Cost Circulations

The simplest special case of the minimum-cost flow problem requires the input flow network to satisfy two constraints:

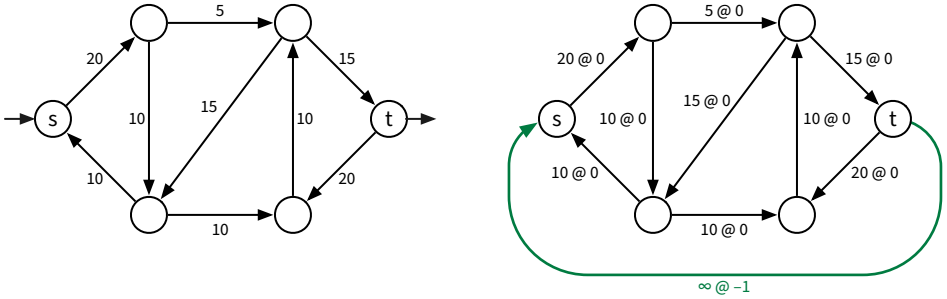
- All vertex balances are zero. Thus, every flow f must satisfy the conservation constraint $\sum_u f(u{\to}v) = \sum_w f(v{\to}w)$ at every vertex v.
- Edges have non-negative capacities and zero lower bounds. Thus, every *feasible* flow f satisfies the capacity constraint $0 \le f(u{\to}v) \le c(u{\to}v)$ at every edge $u{\to}v$.

Because the vertex balances are all zero, any feasible flow in such a network is actually a *circulation*—a sum of directed cycles. This special case is normally called the ***minimum-cost circulation*** problem.

The standard maximum-flow problem can be reduced to the minimum-cost circulation problem as follows. Suppose we are given a flow network $G = (V, E)$ with source and target vertices s and t and non-negative edge capacities $c(e)$. Let $G' = (V, E')$ be the network constructed from G by assigning cost 0 to every edge, and then adding a single edge $t{\to}s$ with infinite capacity and cost -1. Every feasible (s,t)-flow f in G can be extended to a feasible circulation in G' by defining

$$f(t{\to}s) = |f| = \sum_w f(s{\to}w) - \sum_u f(u{\to}s);$$

moreover, the cost of the resulting circulation is precisely $-|f|$. Conversely, for any feasible circulation $f : E' \to \mathbb{R}$, the restriction $f|_E$ is a feasible (s,t)-flow in G with value $f(t{\to}s)$. Thus, any maximum-value (s,t)-flow in G corresponds to a minimum-cost circulation in G' and vice versa.



Maximum (s,t)-flow in G (left) reduces to minimum-cost circulation in G' (right).
Each edge in G' is labeled with capacity @ cost.

Cycle Canceling

We can solve any instance of the minimum-cost circulation problem using a natural generalization of the Ford-Fulkerson augmenting path algorithm called ***cycle canceling***. This algorithm is normally attributed to Morton Klein in 1967, but the key insights date back at least to A. N. Tolstoy's studies of railway transportation networks in the late 1920s.

Consider an arbitrary circulation f in G. As usual, the ***residual network*** G_f consists of all edges $u{\to}v$ with non-zero residual capacity $c_f(u{\to}v)$. For each residual edge $u{\to}v$

in G_f that is not an edge in the original graph G, we define its cost as follows:

$$\$(u \to v) := -\$(v \to u).$$

Figure showing flow f and residual graph G_f

★★★

Now let γ be a directed cycle in the residual graph G_f. We define the residual capacity $c_f(\gamma)$ of γ to be the minimum residual capacity of the edges in γ, and the cost $\$(\gamma)$ of γ to be the *sum* of the costs of the edges in γ. More concisely:

$$c_f(\gamma) := \min_{e \in \gamma} c_f(e) \qquad \$(\gamma) := \sum_{e \in \gamma} \$(e)$$

Exactly as in Ford-Fulkerson, we can ***augment*** the circulation f, by pushing R units of flow around γ, to obtain a new circulation f':

$$f'(u \to v) = \begin{cases} f(u \to v) + c_f(\gamma) & \text{if } u \to v \in \gamma \\ f(u \to v) - c_f(\gamma) & \text{if } v \to u \in \gamma \\ f(u \to v) & \text{otherwise} \end{cases}$$

Straightforward calculation now implies that

$$\$(f') = \$(f) + c_f(\gamma) \cdot \$(\gamma).$$

In particular, if $\$(\gamma) < 0$, the new circulation f' has lower cost than the original circulation f. We immediately conclude that ***a feasible circulation f is a minimum-cost circulation if and only if G_f contains no negative cycles.***

Figure showing negative cycle and updated flow f'

★★★

Finally, Klein's cycle canceling algorithm initializes f to the all-zero circulation, and then repeatedly augments f along an arbitrary negative residual cycle, until there are no more negative residual cycles. For the special instances generated by our reduction from maximum flow, every negative-cost residual cycle consists of a path from s to t and the sole negative edge $t \to s$, so cycle canceling is equivalent to Ford and Fulkerson's augmenting path algorithm.

Analysis

In each iteration of the algorithm, we can find a negative-cost cycle in $O(VE)$ time using a straightforward modification of the Bellman-Ford shortest path algorithm (hint, hint). If both the capacity and the cost of each edge is an integer, then each cycle-augmentation decreases the cost of the circulation by a positive integer, and therefore by at least 1.

It follows that the cycle canceling algorithm runs in $\boldsymbol{O(VE \cdot |\$(f^*)|)}$ ***time***, where f^* is the final circulation. The crude bound $\$(f^*) \geq -ECK$, where $C = \max_e c(e)$ and $K = \max_e |\$(e)|$, also implies that the algorithm runs in at most $\boldsymbol{O(VE^2CK)}$ ***time***. As one might expect from Ford-Fulkerson, these time bounds are exponential in the complexity of the input even when capacities are integers, both time bounds are tight in the worst case, and the algorithm may not terminate or even approach a minimum-cost circulation in the limit if any capacities are irrational.[1]

Like Ford-Fulkerson, more careful choices of *which* cycle to cancel lead to more efficient algorithms. Unfortunately, some natural choices are NP-hard to compute, including the cycle with the most negative total cost and the negative cycle with the fewest edges. In the late 1980s, Andrew Goldberg and Bob Tarjan developed a minimum-cost flow algorithm that repeatedly cancels the so-called ***minimum-mean cycle***, which is the cycle whose *average cost per edge* is smallest. By combining an algorithm of Karp to compute minimum-mean cycles in $O(VE)$ time, efficient dynamic tree data structures, and other sophisticated techniques that are (unfortunately) beyond the scope of this class, their algorithm achieves a running time of $\boldsymbol{O(VE^2 \log V)}$.

Reduction from General Minimum-Cost Flow

In the standard general formulation of the minimum-cost flow problem, we are given a directed flow network $G = (V, E)$ with the following additional data:

- a capacity function $c : E \to \mathbb{R}$ such that $c(e) \geq 0$ for all $e \in E$;
- a lower bound function $\ell : E \to \mathbb{R}$ such that $0 \leq \ell(e) \leq c(e)$ for all $e \in E$;
- a balance function $b : V \to \mathbb{R}$ such that $\sum_v b(v) = 0$; and
- a cost function $\$: E \to \mathbb{R}$.

As usual, a feasible flow in G is a function $f : E \to \mathbb{R}$ that satisfies the capacity constraints $\ell(e) \leq f(e) \leq c(e)$ at every edge e and the balance constraints $\sum_u f(u \to v) - \sum_w f(v \to w) = b(v)$ at every vertex v. The minimum-cost flow problem asks for a feasible flow f with minimum total cost $\$(f) = \sum_e \$(e) \cdot f(e)$.

We can solve the general minimum-cost flow problem in two stages. First, find *any* feasible flow f in G, by reducing to the maximum-flow problem as described in the previous chapter. If there is no feasible flow in G, we can immediately report failure. Otherwise, in the second stage, we compute a minimum-cost circulation f' in the residual graph G_f, using (for example) cycle canceling. Tedious definition-chasing implies that the function $f + f'$ is a minimum-cost flow in the original network G. In total, this algorithm runs in $\boldsymbol{O(VE^2 \log V)}$ ***time***.

[1] If the input network has integer capacities, the algorithm must eventually terminate even when costs are irrational, because there are only a finite number of feasible integer circulations. However, the number of iterations could still be exponential, even if we always cancel the cycle with most negative total cost!

G.2 Successive Shortest Paths

The cycle canceling algorithm implies a natural characterization of minimum-cost flows. Consider a general flow network $G = (V, E)$ with capacities, lower bounds, costs, and balances. A pseudoflow $\psi : E \to \mathbb{R}$ is a minimum-cost flow if and only if it satisfies three conditions:

- **Feasible:** $\ell(e) \le \psi(e) \le c(e)$ for every edge e.
- **Balanced:** $\sum_u \psi(u{\to}v) - \sum_w \psi(v{\to}w) = b(v)$ for every vertex v.
- **Locally optimal:** The residual graph G_ψ contains no negative-cost cycles.

The cycle-canceling algorithm incrementally improves a feasible and balanced pseudoflow (that is, a circulation) until it is also locally optimal. A complementary strategy called ***successive shortest paths*** incrementally improves a feasible and *locally optimal* pseudoflow until it is also *balanced*. The successive shortest paths algorithm was initially proposed by Ford and Fulkerson in the mid-1950s, and then later rediscovered by William Jewell in 1958, Masao Iri in 1960, and Robert Busacker and Paul Gowen in 1960.

The successive shortest paths algorithm requires two simplifying assumptions about the initial flow network G: (1) All lower bounds are zero, and (2) all costs are non-negative. Otherwise, we can replace G with the residual graph of the following pseudoflow:

$$f(u{\to}v) = \begin{cases} c(u{\to}v) & \text{if } \$(u{\to}v) < 0 \\ \ell(u{\to}v) & \text{if } \$(u{\to}v) \ge 0 \text{ and } \ell(u{\to}v) > 0 \\ 0 & \text{otherwise} \end{cases}$$

This residual graph may have non-zero balances at its vertices, even if all balances in the original flow network are zero. If these simplifying assumptions hold, the all-zero pseudoflow is both feasible and locally optimal, but not necessarily balanced.

The successive shortest path algorithm begins by initializing ψ to the all-zero pseudoflow, and then repeatedly augments ψ by pushing flow along a *shortest* path in the residual graph G_ψ from any vertex with negative residual balance to any vertex with positive residual balance, where the length of a path in G_ψ is the sum of the residual costs of its edges. The algorithm is described in more detail in Figure G.1.[2]

SuccessiveShortestPaths is *obviously* an instantiation of the FeasibleFlow algorithm from the previous chapter; the only difference is that we require each augmenting path σ to be a shortest path, instead of an arbitrary path. So *obviously* the algorithm either returns a feasible flow (that is, a feasible and balanced pseudoflow) or reports correctly that no such flow exists. *Obviously.*

Unfortunately, like almost(?) every other sentence that uses the words "obviously", the previous argument misses a crucial subtlety: Shortest paths are only *well-defined*

[2]To simplify the pseudocode, I'm assuming implicitly that there is a feasible flow in G. If no demand node t is reachable in G_ψ from a supply node s, then G has no feasible flow.

```
SUCCESSIVESHORTESTPATHS(V, E, c, b, $):
  for every edge e ∈ E
      ψ(e) ← 0
  B ← Σ_v |b(v)|/2
  while B > 0
      construct G_ψ
      s ← any vertex with b_ψ(s) < 0
      t ← any vertex reachable from s with b_ψ(t) > 0
      σ ← shortest path in G_ψ from s to t
      AUGMENT(s, t, σ)
  return ψ

AUGMENT(s, t, σ):
  R ← min{−b_ψ(s), b_ψ(t), min_{e∈σ} c_ψ(e)}
  B ← B − R
  for every directed edge e ∈ σ
      if e ∈ E
          ψ(e) ← ψ(e) + R
      else ⟨⟨rev(e) ∈ E⟩⟩
          ψ(e) ← ψ(e) − R
```

Figure G.1. The successive shortest paths algorithm for minimum-cost flows.

in graphs without negative cycles. To argue that SUCCESSIVESHORTESTPATHS is even a well-defined algorithm, much less a *correct* algorithm, we need to prove the following lemma:

Lemma G.1. *After every iteration of the main loop of SUCCESSIVESHORTESTPATHS, the feasible pseudoflow ψ is locally optimal; that is, the residual graph G_ψ contains no negative cycles.*

Proof: We prove the lemma by induction. The base case is immediate; the initial all-zero pseudoflow is locally optimal because the graph G has no negative edges.

Suppose that ψ is locally optimal at the beginning of an iteration of the main loop. By definition, there are no negative cycles in the residual graph G_ψ, so shortest paths are well-defined. Let s be an arbitrary supply vertex and let t be an arbitrary demand vertex in G_ψ. Let σ be a shortest path in G_ψ, and let ψ' be the resulting pseudoflow after augmenting along σ. We need to prove that ψ' is locally optimal.

For the sake of argument, suppose the residual graph $G_{\psi'}$ contains a negative cycle γ. Let $f = \sigma + \gamma$ denote the pseudoflow in G_ψ obtained by sending one unit of flow along σ and then one unit of flow along γ. Although γ may contain edges that are not in G_ψ, all such edges must be reversals of edges in σ; thus, f is positive only on edges in G_ψ. Routine calculations imply that f is an integral (s,t)-flow in G_ψ with value 1.[3] Thus,

[3]Here I'm ignoring the residual balances in G_ψ. Note that f may not be a *feasible* flow in G_ψ, but that doesn't matter.

by the Flow Decomposition Lemma, f can be decomposed into a single path σ' from s to t and possibly one or more cycles $\gamma_1, \ldots, \gamma_k$. Our assumption that ψ is locally optimal implies that each cycle γ_i has non-negative cost, and therefore $\$(\sigma') \le \$(f) < \$(\sigma)$. But this is impossible, because σ is a *shortest* path from s to t. □

Figure with a complicated instance of σ and γ.

★★★

For the special instances of minimum-cost flow generated by our reduction from maximum flow, the only supply node is the source vertex s, the only deficit node is the target vertex t, and every edge in the network has cost 0 except the edge $s \to t$, which has cost 1. Thus, for these instances, the successive shortest paths is equivalent to Ford and Fulkerson's augmenting path algorithm (followed by a single augmentation of the edge $s \to t$).

Analysis

In each iteration of the algorithm, we can find a shortest in $O(VE)$ time using a the Bellman-Ford shortest path algorithm. (Actually, we can use Dijkstra's algorithm to compute the *first* shortest path, but augmenting along a path of positive-cost edges introduces residual edges with negative costs, so we can't use Dijkstra's algorithm in later iterations.)

Assuming the capacity of each edge is an integer, then each augmentation decreases the total absolute balance B by a positive integer, and therefore by at least 1. It follows that the cycle canceling algorithm halts after at most B iterations and thus runs in ***$O(VEB)$ time***. As one might expect from Ford-Fulkerson, this time bound is exponential in the complexity of the input when capacities are integers, the time bound are tight in the worst case, and the algorithm may not terminate or even approach a minimum-cost circulation in the limit if any capacities are irrational.

G.3 Node Potentials and Reduced Costs

The main bottleneck in Ford and Fulkerson's successive shortest paths algorithm is computing a shortest path in every iteration. Because some edges in the residual graph may have negative costs, we are forced to use Bellman-Ford instead of Dijkstra's algorithm. Or so it seems.

The following clever trick to speed up successive shortest paths was proposed by Jack Edmonds and Richard Karp in 1969 (published in 1972) and independently by Nobuaki Tomizawa in 1970 (published in 1971). The same trick was later applied by Donald Johnson in the late 1970s to speed up all-pairs shortest-path algorithms; Johnson attributes the trick to Edmonds and Karp.

Suppose each vertex v in the flow network has been assigned an arbitrary real ***potential*** $\pi(v)$. We define the ***reduced cost*** of each edge with respect to potential

function π as follows:

$$\$^\pi(u{\to}v) \;:=\; \pi(u) + \$(u{\to}v) - \pi(v).$$

Then for any path α from u to v, we immediately have

$$\$^\pi(\alpha) \;=\; \sum_{e\in\alpha} \$^\pi(e) \;=\; \pi(u) + \sum_{e\in\alpha} \$(e) - \pi(v) \;=\; \pi(u) + \$(\alpha) - \pi(v).$$

The potentials of all the intermediate vertices on α cancel out; intuitively, for each intermediate vertex x, we get an "entrance gift" of $\pi(x)$ when the path enters x, which we immediately pay back as an "exit tax" when the path leaves x. This simple observation has two important consequences:

- Shortest paths with respect to the reduced cost function $\$^\pi$ are identical to shortest paths with respect to the original cost function $\$$.
- For any cycle γ, we immediately have $\$^\pi(\gamma) = \$(\gamma)$; thus, a flow network G has no cycles with negative cost if and only if G has no cycles with negative *reduced* cost.

Edmonds, Karp, and Tomizawa observed that if we choose the right potential function, we can ensure that all reduced edge costs are non-negative, allowing us to compute shortest paths using Dijkstra's algorithm instead of Bellman-Ford.

Lemma G.2. *A flow network G has no negative-cost cycles if and only if, for some potential function $\pi : V \to \mathbb{R}$, every edge in G has non-negative reduced cost.*

Proof: One direction is easy. Suppose G contains a cycle γ with negative cost. Then for every potential function π, we have $\$^\pi(\gamma) = \$(\gamma) < 0$, which implies that at least one edge in γ has negative reduced cost.

Conversely, suppose G has no negative-cost cycles. Then shortest path distances with respect to the cost function $\$$ are well-defined. Fix an arbitrary vertex s, and for every vertex v, let $\boldsymbol{dist(v)}$ denote the shortest-path distance from s to v. (If necessary, add zero-capacity, high-cost edges from s to every other vertex, so that these distances are all finite.) Then for every edge $u{\to}v$, we immediately have

$$dist(v) \le dist(u) + \$(u{\to}v),$$

since otherwise there would be a path from s through u to v with cost less than $dist(v)$; this inequality can be rewritten as

$$dist(u) + \$(u{\to}v) - dist(v) \ge 0.$$

Thus, if use these shortest-path distances as potentials, every edge in G has non-negative reduced cost. □

Speeding up Successive Shortest Paths

Now we seem to have a chicken-and-egg problem: We need a good potential function to compute shortest paths quickly, but we need shortest paths to compute a good potential function!

We can break this apparent cycle as follows. Because the original flow network G has only non-negative edge costs, we can start with $\pi(v) = 0$ at every vertex v. Then after each iteration of the main loop, we update the vertex potentials to ensure that reduced costs in the new residual graph are still non-negative. Specifically, ***we use the shortest-path distances from each iteration as the vertex potentials in the next iteration***, as described in the pseudocode below.

```
FastSuccessiveShortestPaths(V, E, c, b, $):
  for each edge e ∈ E
      ψ(e) ← 0
  for each vertex v ∈ V
      π(e) ← 0
  B ← Σ_v |b(v)|/2
  while B > 0
      construct G_ψ
      s ← any vertex with b_ψ(s) < 0
      t ← any vertex reachablse from s with b_ψ(t) > 0
      dist ← Disjkstra(G_ψ, s, $^π)
      σ ← shortest path in G_ψ from s to t     ⟨⟨computed by Disjktra⟩⟩
      Augment(s, t, σ)
      for each vertex v ∈ V
          π(v) ← dist(v)
  return ψ
```

Lemma G.3. *After every iteration of the main loop of* FastSuccessiveShortestPaths*, every edge $u \to v$ in the residual graph G_ψ has non-negative reduced cost: $\$^\pi(u \to v) \geq 0$.*

Proof: We prove the lemma by induction. The base case is immediate, because the original graph G has no negative-cost edges.

Suppose at the beginning of some iteration that for every edge $u \to v$ in G_ψ, we have $\$^\pi(u \to v) \geq 0$. Let s be an arbitrary supply vertex, and for any vertex v, let $dist(v)$ denote the shortest-path distance from s to v with respect to the reduced cost function $\$^\pi$. Let σ be a shortest path in G_ψ from s to some demand vertex t, and let ψ' be the resulting pseudoflow after augmenting along σ.

Now let $u \to v$ be an arbitrary edge in the new residual graph $G_{\psi'}$. We need to prove that $\$^{dist}(u \to v) \geq 0$. There are two cases to consider:

- Suppose $u \to v$ is an edge in the old residual graph G_ψ. Then following the proof of Lemma 2, we immediately have $\$^{dist}(u \to v) = dist(u) + \$^\pi(u \to v) - dist(v) \geq 0$.

- Suppose $u \to v$ is not an edge in the old residual graph G_ψ. Then $u \to v$ must be the reversal of an edge in σ. It follows that $dist(u) = dist(v) + \$^\pi(v \to u)$, and therefore

$$\begin{aligned}\$^{dist}(u \to v) &:= dist(u) + \$^\pi(u \to v) - dist(v) \\ &= dist(u) + \pi(u) + \$(u \to v) - \pi(v) - dist(v) \\ &= dist(u) + \pi(u) - \$(v \to u) - \pi(v) - dist(v) \\ &= dist(u) - \$^\pi(v \to u) - dist(v) \\ &= 0.\end{aligned}$$

In both cases, we conclude that $\$^{dist}(u \to v) \geq 0$, as required. □

This lemma implies that we can compute shortest paths at every iteration of FASTSUCCESSIVESHORTESTPATHS using Disjktra's algorithm in $O(E \log V)$ time. It follows that the overall algorithm runs in $\boldsymbol{O(BE \log V)}$ ***time***.

The fastest minimum-cost flow algorithm currently known (at least among algorithms whose running times depend only on V and E), due to James Orlin in the early 1990s, reduces the problem to $O(E \log V)$ iterations of Dijkstra's shortest-path algorithm and therefore runs in $\boldsymbol{O(E^2 \log^2 V)}$ ***time***.

G.4 Transshipment and Transportation

The ***transshipment*** and ***transportation*** problems are two interesting special cases of minimum-cost flows, where the flow values are not directly limited by capacity constraints on the edges, but rather indirectly limited by the balance conditions at the vertices.

A transshipment network is a directed graph $G = (V, E)$ with vertex balances $b : V \to \mathbb{R}$ and edge costs $\$: E \to \mathbb{R}$, but *no* capacity function; each edge is allowed to carry *any* non-negative amount of flow. Thus, a feasible flow in such a network is a function $f : E \to \mathbb{R}$ such that $f(e) \geq 0$ for every edge e and $\sum_u f(u \to v) - \sum_w f(v \to w) = b(v)$ for every vertex v. As usual, the transshipment problem asks for a feasible flow with minimum total cost $\$(f) = \sum_e \$(e) \cdot f(e)$.

The transshipment problem was first cast in this discrete form by Alex Orden in 1956. However, transshipment has been a common practice in long-distance commerce for as long as there has been long-distance commerce.

Traditionally, the vertices in a transshipment network are divided into three categories: *supply* or *source* vertices have negative balance; *demand* or *target* vertices have positive balance, and ***transshipment*** vertices have zero balance. The *transportation* problem is a special case of the transshipment problem in which the input satisfies two additional conditions:

- There are no transshipment nodes; every vertex has a non-zero balance constraint.
- Every edge $u \to v$ leaves a supply node u and enters a demand node v. Thus, the flow network is a directed bipartite graph, where the vertex partition separates supply vertices from demand vertices.

Some formulations of the transportation problem require G to be a *complete* bipartite graph, but if necessary we can introduce any missing edges with infinite cost.

The transportation problem was first cast in this modern form by Frank Hitchcock in 1942 and independently by Tjalling Koopmans in 1947; however, earlier versions of the problem were studied by Leonid Kantorivich in the late 1930s, A. N. Tolstoy in the 1920s and 1930s, and even earlier (as a continuous mass-transport problem) by Gaspard Monge in the late 1700s.

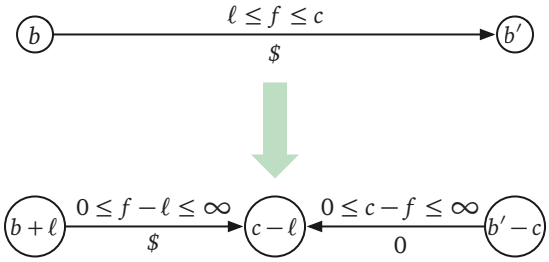
Reducing Minimum-Cost Flow to Transportation

Like the minimum-cost circulation problem, is is possible to reduce arbitrary instances of minimum-cost flow to the transportation problem; the following reduction was described by Delbert Fulkerson in 1960, generalizing Orden's 1956 reduction from the transshipment problem. Let $G = (V, E)$ be an arbitrary flow network, where every edge e has an associated lower bound $\ell(e)$, capacity $c(e)$, and cost $\$(e)$, and every vertex v has an associated balance $b(v)$.

We can remove the capacity constraints from any single edge $u \to v$ as follows. First, replace the edge with a pair of edges $u \to z_{uv}$ and $v \to z_{uv}$, where z_{uv} is a new vertex. Next, set the various constraints for the new vertex and edges as follows:

- $\ell(u \to z_{uv}) = \ell(v \to z_{uv}) = 0$,
- $c(u \to z_{uv}) = c(v \to z_{uv}) = \infty$,
- $\$(u \to z_{uv}) = \$(u \to v)$ and $\$(v \to z_{uv}) = 0$, and
- $b(z_{uv}) = c(u \to v) - \ell(u \to v)$;

Finally, increase the balance $b(u)$ by $\ell(u \to v)$ and decrease the balance $b(v)$ by $c(u \to v)$. Call the resulting flow network G'.



Removing capacity constraints from a single edge.

Given any feasible flow f in the original network G, we can construct a corresponding flow f' in G' by setting

$$f'(e) = \begin{cases} f(u \to v) - \ell(u \to v) & \text{if } e = u \to z_{uv}, \\ c(u \to v) - f(u \to v) & \text{if } e = v \to z_{uv}, \\ f(e) & \text{otherwise.} \end{cases}$$

Routine calculations imply that f' is a feasible flow in G' with cost $\$(f) - \ell(u \to v) \cdot \$(u \to v)$. Conversely, given a feasible flow f' in G', we can construct a corresponding flow f in G by setting

$$f(e) = \begin{cases} f'(u \to z_{uv}) + \ell(u \to v) & \text{if } e = u \to v, \\ f'(e) & \text{otherwise.} \end{cases}$$

Again, routine calculations imply that f is a feasible flow in G with cost $\$(f) + \ell(u \to v) \cdot \$(u \to v)$. We conclude, for both flow transformations, that f is a minimum-cost flow in G if and only if f' is a minimum cost flow in G'.

Applying this transformation to every edge in G, we obtain a bipartite transportation network H with $V + E$ vertices and $2E$ uncapacitated edges in $O(E)$ time. Given a minimum-cost flow in H, we can easily recover a minimum-cost flow in the original network G in $O(E)$ time.

Special case: Assignment

We conclude by describing two further interesting special cases of the transshipment/transportation problem.

The ***assignment*** problem is a special case of the Hitchcock-Koopman's transportation problem in which every supply vertex has balance -1 and every demand vertex has balance $+1$. In other words, the assignment problem asks for a ***minimum-weight perfect matching*** in a bipartite graph with weighted edges (where "weight" is what we previously called "cost"). The assignment problem can also be reformulated as a ***maximum***-weight perfect matching problem by negating all edge weights.

The assignment problem was first posed *and solved* by the German mathematician Carl Jacobi in the early 1800s, motivated by the problem of solving a system of ordinary differential equations. Jacobi's algorithm was not published until after his death.[4] Jacobi's algorithm was rediscovered by Harold Kuhn in the mid 1950s. Kuhn called his algorithm the "Hungarian method", because its relied on combinatorial results on perfect matchings published by Hungarian mathematicians Dénes Kőnig and Jenő Egerváry in 1931. (Alas, Jacobi was not Hungarian.) In 1957, topologist James Munkres observed that the Jacobi-Egerváry-Kuhn algorithm runs in $O(V^4)$ time; with more care, the running time can be improved to $O(V^3)$.

Any instance of the transportation problem *with integer balances* can be transformed into an equivalent instance of the assignment problem, by replacing each vertex v with $|b(v)|$ equivalent copies.

[4]Carl Gustav Jacob Jacobi. De investigando ordine systematis aequationum differentialum vulgarium cujuscunque. *J. Reine Angew. Math.* 64(4):297–320, 1865. Posthumously published by Carl Borchardt. English translation by François Ollivier: Looking for the order of a system of arbitrary ordinary differential equations. *Applicable Algebra in Engineering, Communication and Computing*, 20(1):7–32, 2009.

Special Case: Single-Supply Transshipment

Perhaps the most important special case of the *transhipment* problem requires that the network has exactly one supply node s. This special case is better known as the ***single-source shortest path*** problem. Specifically, let T be a shortest-path tree of G rooted at the supply vertex s, and let $dist(v)$ denote the shortest-path distance from s to v. We define a canonical flow $f_T : E \to \mathbb{R}$ for this spanning tree as follows:

$$f_T(u{\to}v) = \begin{cases} \sum_{w \downarrow v} b(w) & \text{if } u{\to}v \in T \\ 0 & \text{otherwise} \end{cases}$$

(Here $w \downarrow v$ indicates that w is a descendant of v in T.) Equivalently, f_t) is the sum of $V-1$ path flows, each sending $b(v)$ units of flow along the unique path from s to some other vertex v; thus,

$$\$(f_T) = \sum_v dist(v) \cdot b(v).$$

We can prove this canonical flow is optimal in (at least) two different ways. First, f_T is precisely the minimum-cost flow constructed by the successive shortest-path algorithm (at least if the shortest path tree T is unique). Second, following the proof of Lemma 1, if the residual graph G_{f_T} contained any negative cycles, we could find a shorter path from s to some other vertex v.

Ford's generic relaxation algorithm to compute shortest path trees is morally equivalent to cycle canceling. Ford's algorithm maintains a tentative shortest path tree T. Any tense edge $u{\to}v$ indicates a negative cycle in the residual graph $G_T = G_{f_T}$, consisting of $u{\to}v$ and the unique path from v to u in T. Relaxing $u{\to}v$ replaces the unique edge $x{\to}v$ in T with $u{\to}v$.

In fact, for *every* transhipment network G without negative cycles, there is an minimum-cost flow is non-zero only on the edges of some spanning tree of G.

Exercises

Need more!

★★★

1. Describe and analyze an algorithm for the following problem, first posed and solved by the German mathematician Carl Jacobi in the early 1800s.

> *Disponantur nn quantitates $h_k^{(i)}$ quaecunque in schema Quadrati, ita ut k habeantur n series horizontales et n series verticales, quarum quaeque est n terminorum. Ex illis quantitatibus eligantur n transversales, i.e. in seriebus horizontalibus simul atque verticalibus diversis positae, quod fieri potest $1.2 \ldots n$ modis; ex omnibus illis modis quaerendum est is, qui summam n numerorum electorum suppeditet maximam.*

For the tiny minority of students who are not fluent in mid-19th century academic Latin, here is a modern English formulation of Jacobi's problem. Suppose we are given an $n \times n$ matrix M. Describe and analyze an algorithm that computes a permutation σ that maximizes the sum $\sum_{i=1}^{n} M_{i,\sigma(i)}$, or equivalently, permutes the columns of M so that the sum of the elements along the diagonal is as large as possible.

Please write a complete, self-contained solution in English, not in mid-19th century academic Latin. Or Hungarian.

2. Suppose you are given a directed flow network G, in which every edge has an *integer* capacity and an *integer* cost, and each vertex has an *integer* balance, along with an *integer* minimum-cost flow f^* in G. Describe algorithm for the following problems:

   (a) IncCost($u \to v$): Increase the cost of $u \to v$ by 1 and update the minimum-cost flow.

   (b) DecCost($u \to v$): Decrease the cost of $u \to v$ by 1 and update the minimum-cost flow.

   Both algorithms should modify f^* so that it is still a minimum-cost flow, more quickly than recomputing a minimum-cost flow from scratch.

3. Every year, Professor Dumbledore assigns the instructors at Hogwarts to various faculty committees. There are n faculty members and c committees. Each committee member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, Professor Snape might declare that he would serve on the Student Recruiting Committee for 1000 Galleons, that he would *pay* 10000 Galleons to serve on the Defense Against the Dark Arts Course Revision Committee, and that he would not serve on the Muggle Relations committee for any price.

   Conversely, Dumbledore knows how many instructors are needed for each committee, as well as a list of instructors who would be suitable members for each committee. (For example: "Dark Arts Revision: 5 members, anyone but Snape.") If Dumbledore assigns an instructor to a committee, he must pay that instructor's price from the Hogwarts treasury.

   Dumbledore needs to assign instructors to committees so that (1) each committee is full, (3) no instructor is assigned to more than three committees, (2) only suitable and willing instructors are assigned to each committee, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves Dumbledore's problem, or correctly reports that there is no valid assignment whose total cost is finite.

4. Vince wants to borrow a certain amount of money from his friends as cheaply as possible, possibly after first arranging a sequence of intermediate loans. Each of

Vince's friends have a different amount of money that they can lend (possibly zero). For any two people x and y, there is a maximum amount of money (possibly zero or infinite) that x is willing to lend to y and a certain profit (possibly zero or even negative) that x expects from any loan to y.

For example, suppose Vince wants to borrow $\$100$ from his friends Ben and Naomi, who have the following constraints:

- Ben has $\$500$ available to lend.
- Ben is willing to lend up to $\$150$ to Vince at a profit of 20¢ per dollar.
- Ben is willing to lend up to $\$50$ to Naomi, at a loss of 10¢ per dollar.
- Naomi has $\$50$ available to lend.
- Naomi is willing to lend any amount of money to Vince, at a profit of 10¢ per dollar.
- Naomi is not willing to lend money to Ben.

If Vince borrows $\$100$ directly from Ben, he needs $\$120$ to pay off the loan. If Vince borrows $\$50$ from Ben and $\$50$ from Naomi, he needs $\$115$ to pay off the loan: $\$60$ for Ben and $\$55$ for Naomi. But if Vince asks Naomi to borrow $\$50$ from Ben and then borrows the entire $\$100$ from Naomi, then he needs only $\$110$ to pay off Naomi, who can then pay off Ben with just $\$45$. With the same constraints, the maximum amount of money that Vince can borrow is $\$250$.

Describe and analyze an algorithm that finds a sequence of loans that minimizes the amount Vince needs to pay everyone off, or correctly reports that Vince cannot borrow his desired amount. The input has the following components:

- An array $Money[1..n]$, where $Money[i]$ is the amount of money that friend i has.
- An array $MaxLoan[1..n, 0..n]$, where $MaxLoan[i, j]$ is the amount of money that friend i is willing to lend to friend j. "Friend 0" is Vince.
- An array $Profit[1..n, 0..n]$, where $Profit[i, j]$ is the profit per dollar that friend i expects from any load to friend j. Again, "friend 0" is Vince.
- The total amount T that Vince wants to borrow.

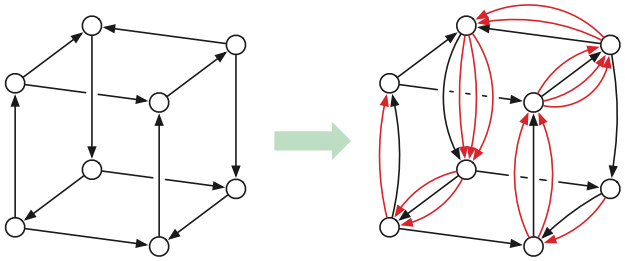
5. An *Euler tour* in a directed graph G is a closed walk (starting and ending at the same vertex) that traverses every edge in G exactly once; a directed graph is *Eulerian* if it has an Euler tour. Euler tours are named after Leonhard Euler, who was the first person to systematically study them, starting with the Bridges of Königsberg puzzle.

   (a) Prove that a directed graph G with no isolated vertices is Eulerian if and only if (1) G is strongly connected and (2) the in-degree of each vertex of G is equal to its out-degree. *[Hint: Flow decomposition!]*

   (b) Suppose that we are given a strongly connected directed graph G with no isolated vertices that is *not* Eulerian, and we want to make G Eulerian by duplicating

existing edges. Each edge e has a duplication cost $\$(e) \geq 0$. We are allowed to add as many copies of an existing edge e as we like, but we must pay $\$(e)$ for each new copy. On the other hand, if G does not already have an edge from vertex u to vertex v, we cannot add a new edge from u to v.

Describe an algorithm that computes the minimum-cost set of edge-duplications that makes G Eulerian.



Making a directed cube graph Eulerian.

6. An ***(s, t)-series-parallel*** graph is an directed acyclic graph with two designated vertices s (the *source*) and t (the *target* or *sink*) and with one of the following structures:

   - **Base case:** A single directed edge from s to t.
   - **Series:** The union of an (s,u)-series-parallel graph and a (u,t)-series-parallel graph that share a common vertex u but no other vertices or edges.
   - **Parallel:** The union of two smaller (s,t)-series-parallel graphs with the same source s and target t, but with no other vertices or edges in common.

   Recall that any series-parallel graph can be represented by a binary *decomposition tree*, whose interior nodes correspond to series compositions and parallel compositions, and whose leaves correspond to individual edges. The decomposition tree can be constructed in $O(V+E)$ time.

   (a) Describe an efficient algorithm to compute a *minimum-cost* maximum flow from s to t in an (s,t)-series-parallel graph whose edges have *unit* capacity and arbitrary costs.

   ♥(b) Describe an efficient algorithm to compute a *minimum-cost* maximum flow from s to t in an (s,t)-series-parallel graph whose edges have *arbitrary* capacities and costs.