CO351 - Network Flow Theory

University of Waterloo

Nicholas Pun

Fall 2019

Contents

List of Algorithms

1 Graph Theory Primer

Let $G = (V, E)$ be a graph, where V is the vertex set and E is the edge set.

Definition 1.1

The **degree** of a vertex $v \in V$ (denoted $\deg(v)$) is the number of edges with one end in v. (i.e. The size of the set $\{va \mid a \in V,\ a \neq v\}$).

A **walk** is a sequence of vertices $v_1 v_2 \dots v_k$ where $v_i v_{i+1}$ is an edge. A **path** is a walk where all vertices are distinct. A **cycle** is a walk where $v_1 = v_k$ and $v_1, \dots v_{k-1}$ are distinct.

Finally, we say a graph is **connected** if there exists a path between any two vertices in G.

Definition 1.2

For $S \subset V$, the **cut** induced by S is the set of all edges with one end in S and one end not in S, denoted $\delta(S) = \{uv \in E \mid u \in S,\ v \notin S\}$. Given two vertices $s, t \in V$ with $s \in S, t \notin S$, we call $\delta(S)$ an **s,t-cut**

An **s,t-path** is a path with starting vertex s and ends on t.

Theorem 1.1

There exists an s,t-path if and only if every s,t-cut is nonempty

Definition 1.3

A **tree** is a connected graph with no cycles. A **spanning tree** is a subgraph that is a tree and has vertex set V

Note the following:

- A tree on n vertices contains $n - 1$ edges.
- If T is a tree, then adding an edge $uv \notin T$ creates exactly one cycle C. Moreover, if xy is an edge in C, then $T + uv - xy$

Let $D = (N, A)$ be a directed graph. N is a set of nodes and A is a set of ordered pairs of nodes (called arcs).

Definition 1.4

For an arc (u, v), we call u the **tail** and v the **head**.

The **out-degree** of node u (denoted $d(u)$ or $d^{\text{out}}(u)$) is the number of arcs with tail u.

The **in-degree** of node u (denoted $d(\overline{u})$ or $d^{\text{in}}(u)$) is the number of arcs with head u.

A **diwalk** is a sequence of nodes $v_1 v_2 \dots v_k$ where (v_i, v_{i+1}) is an arc. **Dipaths** and **Dicycle** are defined analgous to simple graphs but with arcs instead of edges.

For $S \subset N$, the **cut** induced by S is denoted $\delta(S) = \{(u,v) \in A \mid u \in S, v \notin S\}$. (sometimes written as $\delta^{\text{out}}(S)$) This is the set of arcs with tail in S. We denote the complement of S by $\overline{S}$, and define $\delta(\overline{S}) = \{(u,v) \in A \mid u \notin S, v \in S\}$ (sometimes written as $\delta^{\text{in}}(S)$)) to be the set of arcs with head in S. Finally, if $s \in S, t \notin S$, then $\delta(S)$ is an s,t-cut.

Theorem 1.2

There is an s,t-dipath if and only if every s,t cut is non-empty

Proof. ($\Rightarrow$) Suppose there exists an empty s,t-cut $\delta(S)$. This partitions the graph into two sets of nodes S and $N \setminus S$, with $s \in S$ and $t \in N \setminus S$ and no outgoing edges from S to $N \setminus S$. As such, an s,t-dipath cannot exist.

($\Leftarrow$) Suppose every s,t-cut is non-empty and let S be the set of nodes $v \in A$ where a s,v-dipath exists. If $t \in S$, then we're done, so suppose $t \notin S$. Then, $\delta(S)$ is an s,t-cut. By assumption, $\delta(S)$ is non-empty and so there is an arc $(x,y) \in \delta(S)$ with $x \in S$ and $y \in N \setminus S$. Since $x \in S$, an s,x-dipath P exists and since $y \notin S$, there does not exist an s,y-dipath, but $P + (x,y)$ is one. This is a contradiction $\square$

Definition 1.5

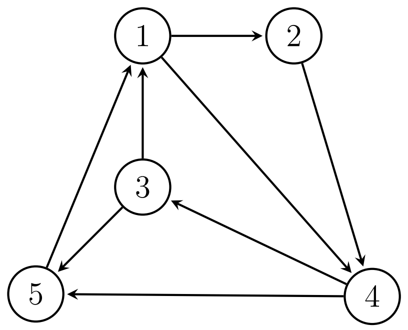
The **node-arc incidence matrix** M of a digraph $D = (N, A)$ is a matrix of $|N|$ rows and $|A|$ columns, such that:

- The rows correspond to the nodes of D,
- The columns correspond to the arcs of D,
- And the entry for node u and arc (i,j), denoted $m_{u,ij}$ is given by:

$$m_{u,ij} = \begin{cases} 0 & \text{if } u \neq i \text{ and } u \neq j, \\ +1 & \text{if } u = j \text{ and,} \\ -1 & \text{if } u = i \end{cases}$$

Example 1.1

An example of a digraph $D = (N, A)$ (on the left) and its corresponding node-arc incidence matrix on the right.



$$\begin{array}{c|cccccccc} & 12 & 14 & 24 & 31 & 35 & 43 & 45 & 51 \\ 1 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 2 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 \end{array}$$

2 Transshipment Problem

2.1 Introduction

Let $D = (N, A)$ be a digraph, $b \in \mathbb{R}^N$ *node demands*, and $w \in \mathbb{R}^A$ *arc costs.* We call $x \in \mathbb{R}^A$ a *flow*, if it satisfies:

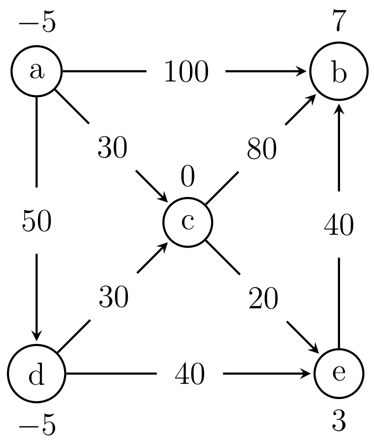
$$\sum_{uv \in A} x_{uv} - \sum_{vk \in A} x_{vk} = b_v \quad \forall v \in N$$

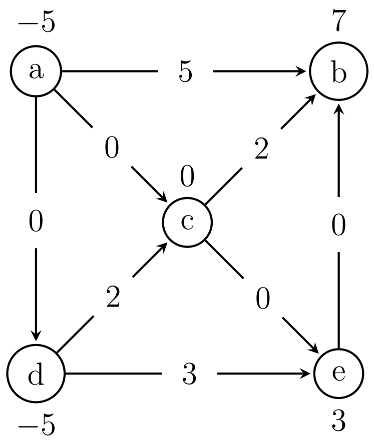
That is, for every node, we want the net flow through the node (the flow into the node minus the flow out of the node) to be equal to the demand of the node. Define $x(S) := \sum_{uv \in S} x_{uv}$. We will usually write the above constraint in the following form:

$$x(\delta(\overline{v})) - x(\delta(v)) = b_v \quad \forall v \in N$$

Example 2.1

The left shows a digraph $D = (N, A)$, along with the node demands next to each node and the arcs costs on each arc. The right shows a feasible flow.





Given the inputs: $D = (N, A), b \in \mathbb{R}^N, w \in \mathbb{R}^A$, the **goal** in the transshipment problem (TP) is to find a flow of minimum cost, where the cost of a flow is defined as $w^\intercal x = \sum_{ij \in A} w_{ij} x_{ij}$.

A common theme throughout this course will be to view our problems through the lens of linear programs (LP). Below is the LP formulation for the TP problem:

$$\begin{aligned} \min \quad & w^\intercal x \\ \text{s.t} \quad & x(\delta(\overline{v})) - x(\delta(v)) = b_v \quad \forall v \in N \\ & x \geq 0 \end{aligned} \tag{2.1}$$

Alternatively, letting M be the node-arc incidence matrix for our digraph, we can also rewrite the LP like so:

$$\begin{aligned} \min \quad & w^\intercal x \\ \text{s.t} \quad & Mx = b \quad \forall v \in N \\ & x \geq 0 \end{aligned}$$

We'll also find the dual formulation of this problem useful as well:

$$\begin{aligned} \max \quad & b^\intercal y \\ \text{s.t} \quad & y_v - y_u \leq w_{uv} \quad \forall uv \in A \\ & y \text{ free} \end{aligned} \tag{2.2}$$

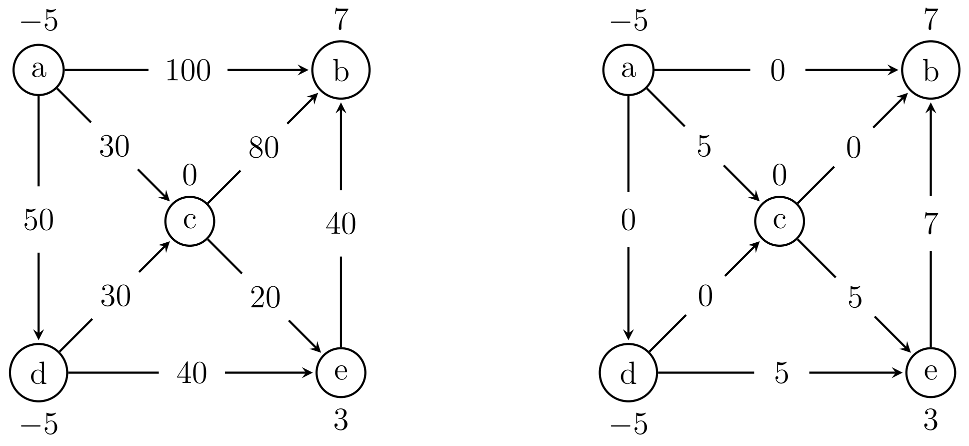
We call the variables $y \in \mathbb{R}^N$ *node potentials.*

Recall from LP theory that two feasible solutions x, y are optimal if and only if the complementary slackness (CS) conditions hold. What are the CS conditions for our primal-dual pair?

$$x_{uv} = 0 \quad \text{OR} \quad y_v - y_v \leq w_{uv} \quad \forall uv \in A \tag{2.3}$$

Example 2.2: Optimal flow

The left shows a digraph $D = (N, A)$, along with the node demands next to each node and the arcs costs on each arc. The right shows (what I claim to be) an optimal flow.



Using the CS conditions, let's check that the flow is indeed optimal.

First, check visually that x is a feasible flow. Then, the CS conditions tell us that $x_{uv} = 0$ or $y_v - y_u = w_{uv}$. There are 4 arcs with non-zero flow, so the following 4 equations must hold:

- $y_c - y_a = 30$
- $y_e - y_c = 20$

- $y_b - y_e = 40$
- $y_e - y_d = 40$

We have 4 equation, but 5 unknowns. But, since y is free, we can set $y_a = 0$. This will give us: $y_b = 90, y_c = 30, y_d = 10, y_e = 50$. And, it remains to check that y with these values is feasible.

- $y_b - y_a \leq 100 \longrightarrow 90 - 0 = 90 \leq 100$
- $y_d - y_a \leq 50 \longrightarrow 10 - 0 = 10 \leq 50$
- $y_d - y_c \leq 30 \longrightarrow 10 - 30 = -20 \leq 30$
- $y_c - y_b \leq 80 \longrightarrow 30 - 90 = -60 \leq 80$

So, both x, y are feasible, and the complementary slackeness conditions hold. As such, x, y are both optimal solutions. The optimal value to this TP is: 730.

Note. Always make sure that both x, y are feasible!

2.2 Network Simplex Method

We've described the basic foundations for the transshipment problem. Our goal now is to translate the simplex method into an algorithm for solving TPs.

Remark 2.1. We will assume $\sum_{v \in N} b_v = 0$ and that $D = (N, A)$ is connected.

2.2.1 Finding a Basis

Let's take a look at the incidence matrix of the digraph from Example 2.2:

$$\begin{array}{c|cccccccc} & ab & ac & ad & cb & eb & dc & ce & de \\ \hline a & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ b & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ c & 0 & 1 & 0 & -1 & 0 & 1 & -1 & 0 \\ d & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 \\ e & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 \end{array}$$

Suppose we were to add up all the rows of this matrix. There is exactly one -1 and one $+1$ in each column (and the rest of the entries are 0). As, such, the sum in each column is 0, and so the sum of all the rows is the zero vector!

In general, if we were to add up all $|N|$ rows of some incidence matrix, we will get the zero vector. So, the rows are linearly dependent, and the rank of our incidence matrix must be $< |N|$! This suggests that we can only choose at most $|N| - 1$ columns to form our basis, and in fact, we will prove that the rank is precisely $|N| - 1$.

Proposition 2.1

The columns of the incidence matrix corresponding to a cycle are linearly dependent.

We'll use the following definition in our proof.

Definition 2.1: Forward/Backward Arc

For an (undirected) cycle $v_0 v_1 \dots v_{k-1} v_0$ in a digraph $D = (N, A)$, an arc of the form $v_i v_{i+1}$ is forward arc and an arc of the form $v_{i+1} v_i$ is a backward arc.

Proof. Let C be a cycle and M_C be the submatrix formed by taking the columns corresponding to arcs of C from the incidence matrix M. Create a new matrix M'_C by taking the columns of M_C corresponding to backwards arcs and multiplying them by -1. M'_C is the incidence matrix of a cycle where every arc is a forward arc (and so a directed cycle).

For every node $v \in C$, $d(v) = d(\overline{v}) = 1$, so for every row representing v in M'_C, there is exactly one entry with 1 and one entry with -1. The sum of the entries of each row in M'_C will be 0, and so the sum of the columns of M'_C is 0, hence the columns of M_C are linearly dependent. $\square$

In fact, the converse of this statement will also hold.

Proposition 2.2

A set of linearly dependent columns of the incidence matrix must include a cycle.

Proof. Let B be a set of linearly dependent columns in M. There exists a nontrivial linear combination of these columns that equal 0. Let $B' = \{e_1, \dots, e_k\}$ be the subset of columns in B where $c_1 M_{e_1} + \dots + c_k M_{e_k} = 0, c_1, \dots, c_k \in \mathbb{R}$ and all $c_i \neq 0$. (Here, M_{e_k} denotes the column corresponding to the arc e_k) For each node v, the sum of the entries of that row equals 0, i.e. $c_1 M_{v,e_1} + \dots + c_k M_{v,e_k} = 0$. ($M_{v,e_i}$ represents the entry for node v in column e_i). Let S be the subset of nodes where $M_{v,e_i} \neq 0$ for at least one i. Since $M_{v,e_i} \neq 0$, then $c_i M_{v,e_i} \neq 0$, and so there must be another entry for node v with $M_{v,e_j} \neq 0, j \neq i$. Hence, for each node $v \in S$, there are at least 2 arcs in B' whose columns in M have non-zero entries for v, and each of these arcs are incident to v. Therefore, the subgraph with S as the nodes and B' as the arcs have the property that every node has degree at least 2, and so must contain a cycle. $\square$

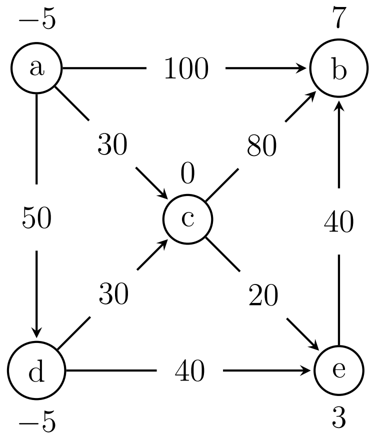
Propositions 2.1 and 2.2 tells is that a basis in our algorithm will be a maximal set of arcs that do not contain a cycle. This is precisely a spanning tree! Let's summarize this in the following theorem:

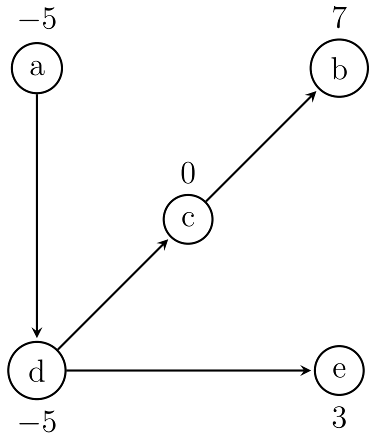
Theorem 2.1

Let M be the incidence matrix of a connected digraph $D = (N, A)$. Then, a set of $|N| - 1$ columns of M is a basis if and only if these $|N| - 1$ arcs corresponding to the columns form a spanning tree of D

Example 2.3: Finding a flow given a tree

The left shows a digraph $D = (N, A)$, along with the node demands next to each node and the arcs costs on each arc. The right shows a tree in D, can we find a flow based on it?



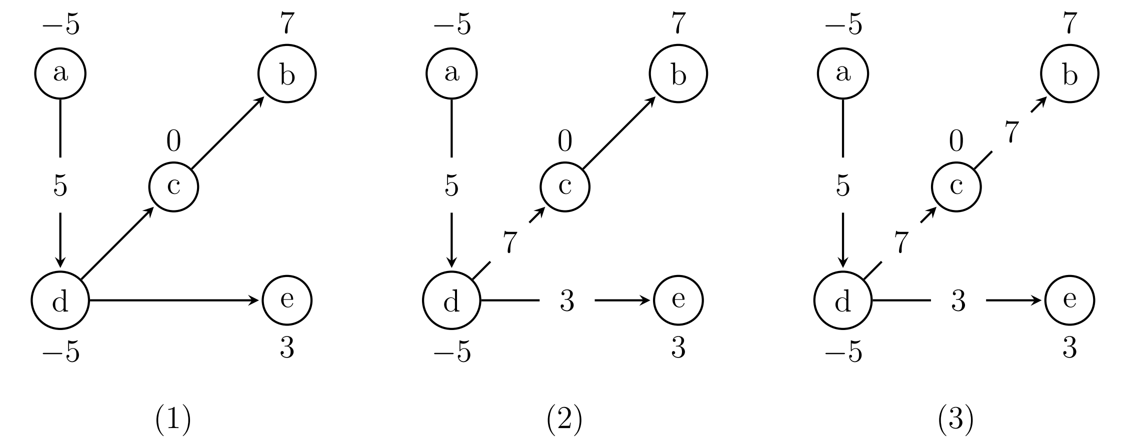


First, note that we can solve for the reduced system $M_T x = b_T$ ... but that's a lot of work. Instead, let's just look at the graph and do this in an iterative fashion:

(1) Start with a leaf. For example, here we can pick node a. Its demand is -5, and the only arc it can send flow through is ad, so ad must have a flow of 5 units.

(2) Now, d must has -5 demand as well, so it must send a total of 10 (5 from a and 5 by itself) to nodes c and e. e has a demand of 3, so let's send 3 flow through de and the remaining flow (7) goes through dc.

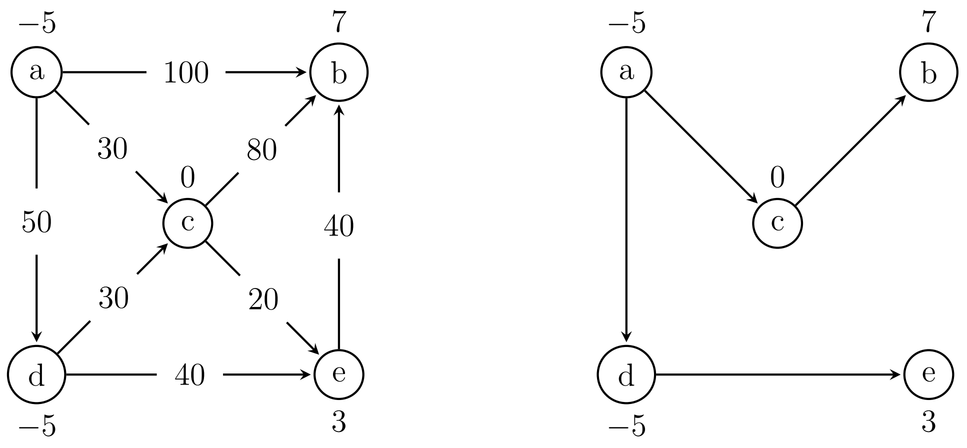
(3) Finally, c has 0 demand, but it just received 7 flow from d, so it must send 7 flow through cb to b. Good thing b has demand 7!



In Example 2.3, we're given a spanning tree and can successfully find a corresponding tree flow. Can we do this for all spanning trees?

Example 2.4: Finding another (?) flow given a tree

The left shows a digraph $D = (N, A)$, along with the node demands next to each node and the arcs costs on each arc. The right shows a tree in D, can we find a flow based on it?

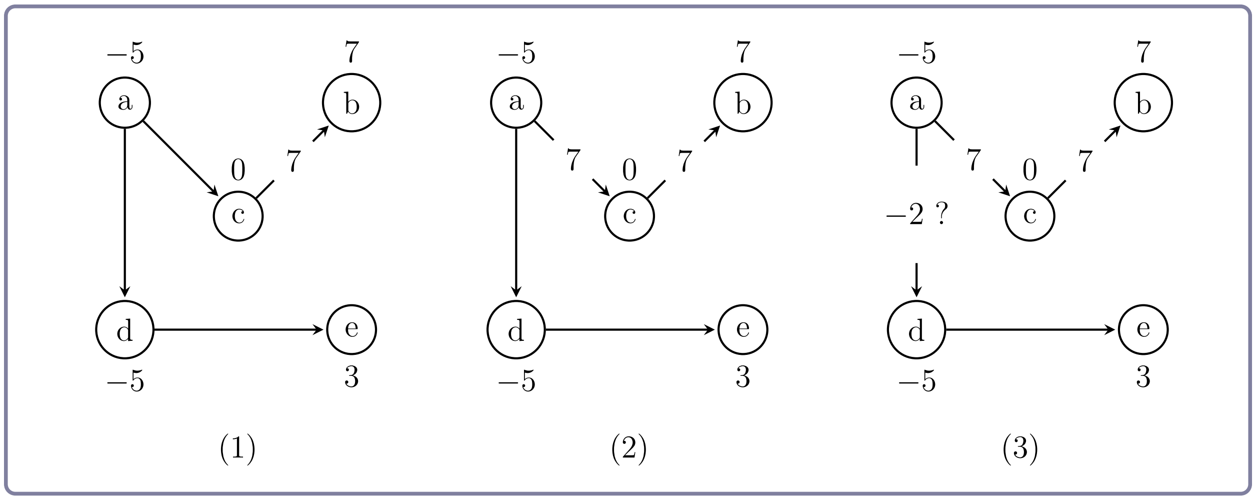


Again, let's work through the tree, starting at the leaf b

(1) b has demand 7, so it'll need 7 flow from c

(2) c has demand 0, so it'll need 7 flow from a

(3) a has demand -5, but it's already pushed 7 flow through ac. That means it must push -2 flow through ad ... that doesn't seem right as flow is nonnegative!



Indeed, as one can see from Example 2.4, we can't simply pick any spanning tree for our basis. This raises the following question.

Question: How do we find an initial basic feasible solution?

We will cover this in Section 2.5 (Hint: We want to construct an auxiliary TP and solve that problem). For now, let's assume that we're given some initial spanning tree with a feasible flow.

2.2.2 Entering and Leaving variables

Question: How to do we choose an entering variable (arc)?

Recall the CS conditions we found back in Equation (2.3). Our flow is optimal if:

$$x_{uv} = 0 \quad \text{OR} \quad y_v - y_v \leq w_{uv} \quad \forall uv \in A$$

Our feasible flow will have $x_{uv} \geq 0$ for every arc uv in the spanning tree. In that case, our CS conditions tell us that we want a corresponding y such that

$$A_B^\intercal y = w_B$$

(where B is the set of arcs in our spanning tree)

Then, for the arcs with 0 flow, we will have an optimal solution if:

$$w_N - A_N^\intercal y \geq 0$$

Further, if there is an arc uv with 0 flow, but $w_{uv} + y_u - y_v < 0$, then our solution is not optimal!

So, to choose an entering arc, we will find an arc $uv \notin B$ such that $w_{uv} + y_u - y_v < 0$, and add that into our basis. We'll give this value $(w_{uv} + y_u - y_v)$ a name:

Definition 2.2: Reduced Cost

Recall that a vector $y \in \mathbb{R}^N$ is called a node potential. Given a node potential y, the reduced cost of an arc uv is:

$$\bar{w}_{uv} = w_{uv} + y_u - y_v$$

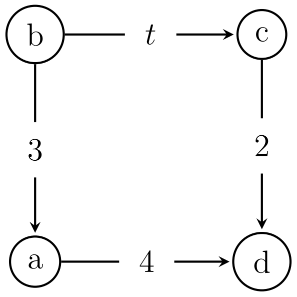
Note. A node potential is feasible if $\bar{w}_{uv} \geq 0$ for all $uv \in A$

This leads us right to our next question:

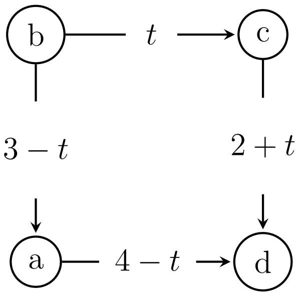
Question: How do we select a leaving variable (arc)?

Adding an arc uv into our spanning tree T creates exactly one cycle, let's call this cycle C. To ensure that the leaving arc gives a spanning tree, the leaving arc needs to be from C.

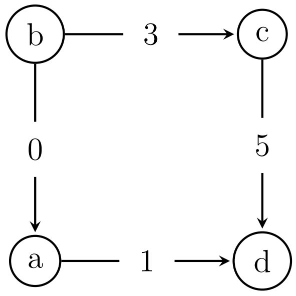
Say we're given the following cycle, where bc is the arc we've added to the spanning tree. To add bc to the spanning tree, we push t units of flow through the arc, around the cycle:



Now, any forward arcs (with respect to bc) will be given additional flow, and any backward arcs will have flow taken away:

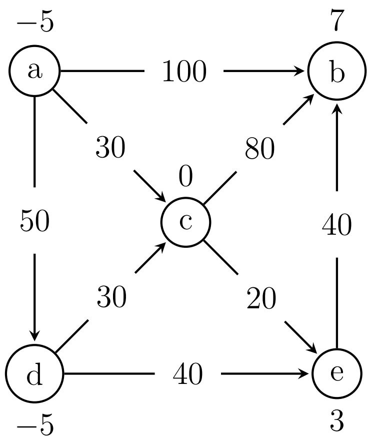


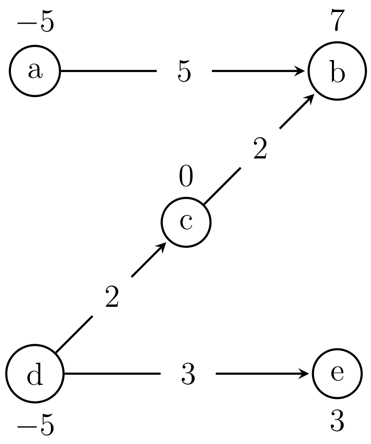
We can't have negative flow, so out of all the backward arcs, we can only choose t to be the minimium flow that is taken away. In this case, we have $t = \min\{4, 3\} = 3$, and the resulting flow looks like:



Example 2.5: Improving a tree flow

On the left, we have the same digraph as with previous examples. On the right, we have a feasible (but not yet optimal!) tree flow.

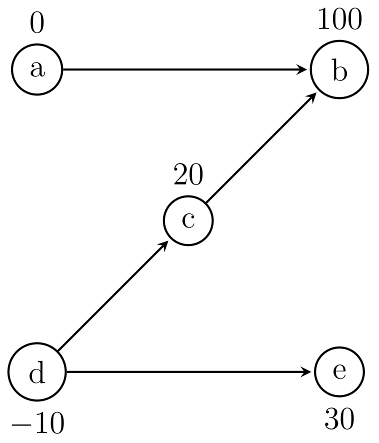




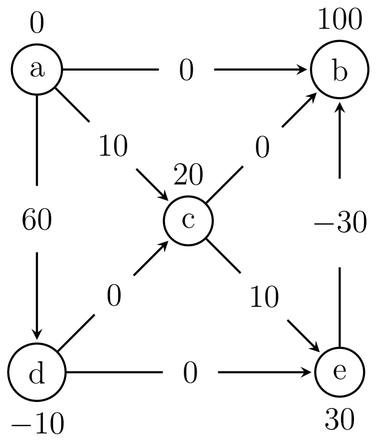
We want to improve this tree flow. First, let's find a feasible node potential y on the nodes. Recall that we want to find y so that $A_B^{\intercal} y = w_B$. So the following constraints must be tight:

- $y_b - y_a = w_{ab} = 100$
- $y_b - y_c = w_{cb} = 80$
- $y_d - y_c = w_{cd} = 30$
- $y_e - y_d = w_{de} = 40$

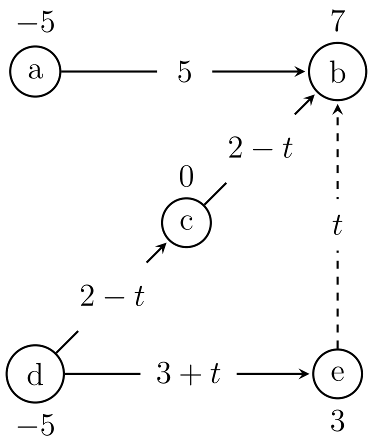
We can always set $y_a = 0$ since y is free, and so this will give us the following node potentials:



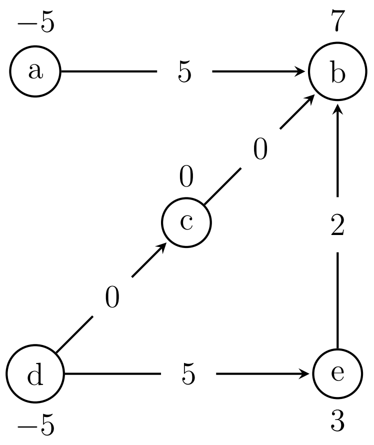
Now, to choose the entering arc, we want to calculate the reduced costs for each arc and choose one with reduced cost < 0. As an example, the reduced cost for arc ac is $\bar{w}_{ac} = w_{ac} + y_a - y_c = 30 + 0 - 20 = 10$. Below is reduced costs for each arc, along with the node potentials.



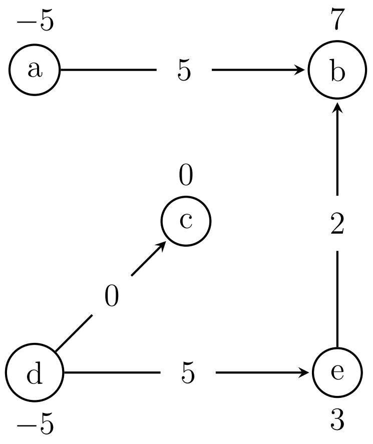
$\bar{w}_{eb} = -30 < 0$, so eb will enter our basis and we push t units of flow through it.

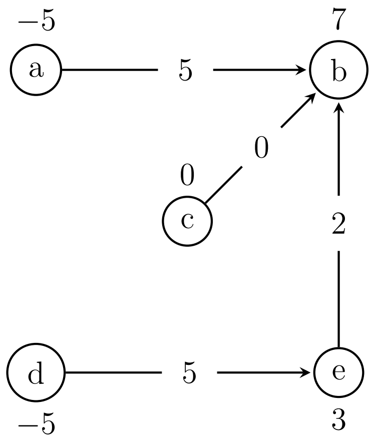


cb and *dc* are our backward arcs, and each currently have a flow of 2, so we can at most push $t = 2$ units of flow through *eb*. As such, the flow on both *cb* and *dc* reach 0, and either can leave the basis. The resulting flow looks like:



And we can continue the algorithm with either of the following spanning trees:





2.2.3 The Algorithm

Algorithm 1: Network Simplex Algorithm

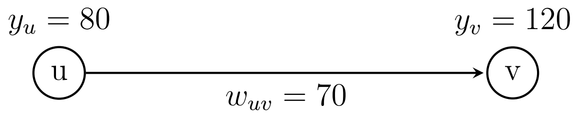
```
Input:
    • $D = (N, A)$ connected digraph
    • $w \in R^A$ arc costs
    • $b \in R^N$ node demands
1  Start with a spanning tree $T$ that has a feasible tree flow $x$;
2  Calculate potentials $y$ such that $\bar{w}_{uv} = 0$ for all $uv \in T$;
3  Calculate $\bar{w}_{uv}$ for any $uv \notin T$;
   while there exists an arc $uv$ with $\bar{w}_{uv} < 0$ do
4      $T + uv$ contains a cycle $C$. Consider the direction of $C$ such that $uv$ is a forward arc;
5      Pick $pq$ such that
           $x_{pq} = \min\{ij \mid ij \text{ is a backward arc of } C\}$
       Let $t = x_{pq}$;
6      For any forward arc in $C$, add a flow of $t$;
7      For any backward arc in $C$, subtract a flow of $t$;
8      $T \leftarrow T + uv - pq$;
9      Go back to step 2;
10 return $T$ with max flow $x$
```

2.3 Economic Interpretation

In this section, we try to provide some intuition for the dual variables by giving an interpretation for node potentials in economic terms.

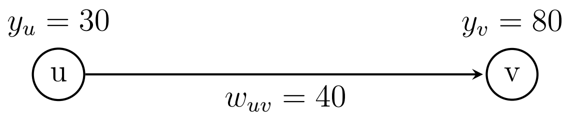
Imagine transporting a set of commodities through our digraph, and the node potentials represents the prices of these commodities at the nodes.

Say we have the following arc uv:



The reduced cost for this arc is: $\bar{w}_{uv} = 70 + 80 - 120 = 30 > 0$. We can interpret this as: "Buying one unit of commodity at u for \$80, transporting through uv for \$70, and selling it at v for \$120". Then, it is clear that we are losing money by using arc uv, so we should never increase flow on arcs with positive reduced cost.

Let's look at an example where $\bar{w}_{uv} < 0$

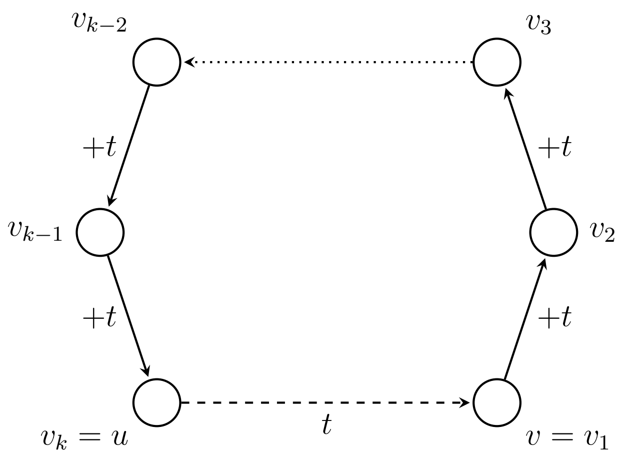


The reduced costs here is: $\bar{w}_{uv} = 40 + 30 - 80 = -10$. In this case, we buy one unit at u for \$30, transport it for \$40, and sell it at v for \$80. We make \$10 by using arc uv! So, when the reduced cost is < 0, we increase flow on that arc.

2.4 Unboundedness

The Network Simplex Algorithm (Algorithm 1) we presented previously is not quite complete. We have yet to characterize when a TP is unbounded or infeasible. In this section, we'll focus on characterizing unboundedness.

Unboundedness occurs when no leaving arc can be found, so that you can push flow on the entering arc indefinitely. When does this happen? Say uv is the entering arc and the cycle it creates is $T + uv = C = v_1 v_2 \dots v_k v_1$:



All arcs are forward arcs, and so we can increase the flow indefinitely. Let y_{v_1} be the dual potential at v_1, then:

$$\begin{aligned}
y_{v_2} &= y_{v_1} + w_{v_1 v_2} \\
y_{v_3} &= y_{v_2} + w_{v_2 v_3} = y_{v_1} + w_{v_1 v_2} + w_{v_2 v_3} \\
&\vdots \\
y_{v_2} &= y_{v_{k-1}} + w_{v_{k-1} v_k} \\
&= y_{v_1} + w_{v_1 v_2} + \dots + w_{v_{k-1} v_k}
\end{aligned}$$

The reduced cost of uv is then:

$$\begin{aligned}\bar{w}_{uv} &= y_{v_k} + w_{uv} - y_{v_1} \\ &= (y_{v_1} + w_{v_1v_2} + \ldots + w_{v_{k-1}v_k})w_{uv} - y_{v_1} \\ &= \sum_{e\in C} w_e < 0 \quad (\text{since } \bar{w}_{uv} < 0)\end{aligned}$$

Definition 2.3: Negative Dicycle

A negative dicycle is a directed cycle C, where the sum of the weights along the cycle is negative, i.e. $\sum_{e\in C} w_e < 0$

And the presence of a negative dicycle is precisely the characterization of an unbounded TP, since we would be able to send flow through the cycle to reduce cost indefinitely!

Theorem 2.2

A feasible TP is unbounded if and only if there exists a negative dicycle

Proof. ($\Leftarrow$) Let C be a negative dicycle with $w(C) < 0$ and x^* be a feasible flow. Define a flow x^C, where $x^C_e = \begin{cases} t & \text{if } e \in C \\ 0 & \text{if } e \notin C \end{cases}$. So, $x^C(\delta(\bar{v})) - x^C(\delta(v)) = 0$ for each $v \in N$ since C is a dicycle.

Then, $x^* + x^C$ is a feasible flow, since:

$$\begin{aligned}&(x^* + x^C)(\delta(\bar{v})) - (x^* + x^C)(\delta(v)) \\ &= x * (\delta(\bar{v})) - x * (\delta(v)) + x^C(\delta(\bar{v})) - x^C(\delta(v)) \\ &= b_v + 0 = b_v\end{aligned}$$

And the objective value using $x^* + x^C$ is:

$$\begin{aligned}w^\intercal(x^* + x^C) &= w^\intercal x^* + w^\intercal x^C \\ &= w^\intercal x^* + t \cdot w(C)\end{aligned}$$

While $w^\intercal x^*$ is fixed, we can make $t \cdot w(C)$ arbitarily large, and as $t \longrightarrow \infty$, $t \cdot w(C) \longrightarrow -\infty$ (since $w(C) < 0$), so the total objective value $w^\intercal(x^* + x^C) \longrightarrow \infty$

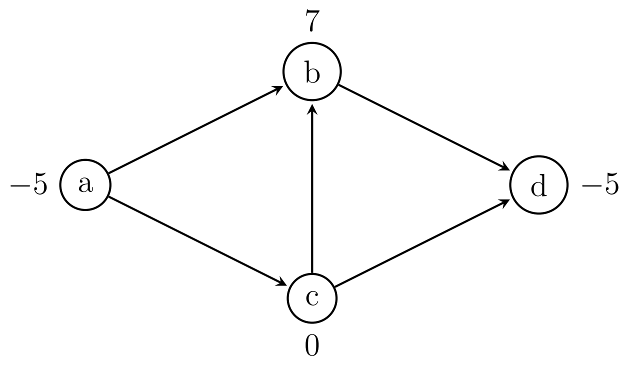
($\Rightarrow$) TODO □

2.5 Infeasibility

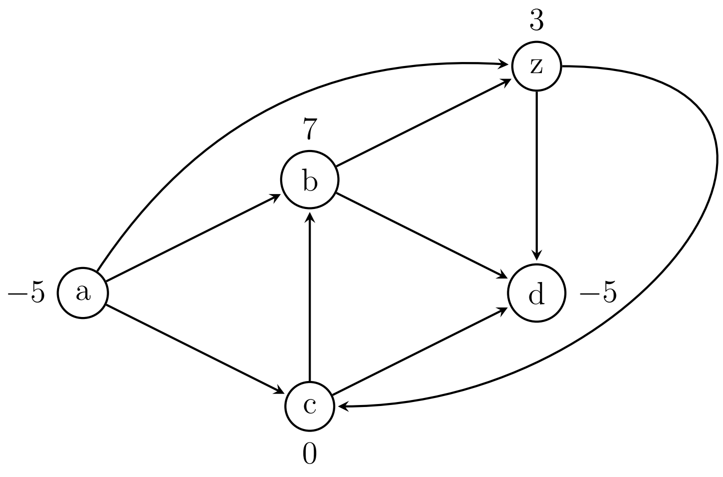
Question: How do we find an initial tree flow?

As promised, we will discuss how to find an initial tree flow in this section. Turns out, this will help us find an infeasibility characterization.

Recall that in the simplex method, we find our initial solution using an auxiliary problem. We will create an auxiliary TP here. We know that every node with negative demands (the supply nodes) supply b_v units of flow. Let's create a new node z that accepts all the flow from the supply nodes. Now, since we assume $\sum b_v = 0$, z has just enough excess flow to push towards the demand nodes. We demonstrate this below:

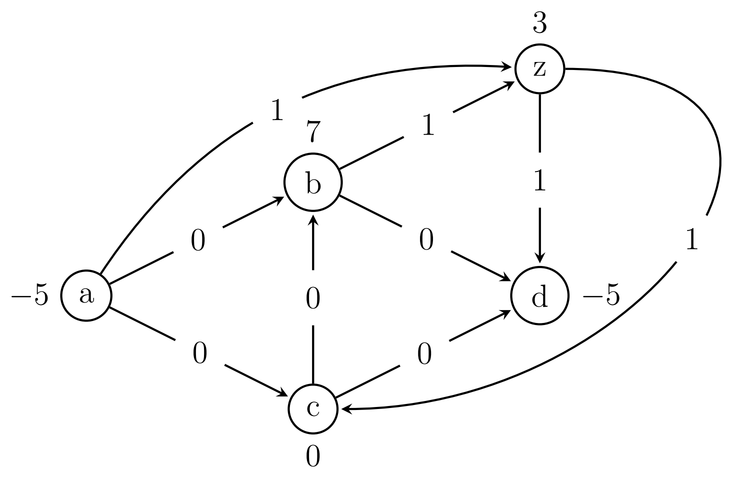


Original Digraph



Digraph with our new node

We want any solution to this modified digraph not use any of the new arcs, so let's set the cost of the original arcs to be 0 and the new arcs to be 1



Auxiliary TP

Definition 2.4: Auxiliary TP

Given a digraph $D = (N, A)$ and node demands $b \in R^M$, the auxiliary TP $D' = (N', A')$ with node demands $b' \in R^{N'}$ and arc costs $w' \in R^{A'}$ is:

- $N' = N \cup \{z\}$, where z is a new node
- $A' = A \cup \{uz \mid u \in N, b(u) < 0\} \cup \{zu \mid u \in N, b(u) > 0\}$
- $b'(z) = 0,\ b'(v) = b(v) \quad \forall v \in N$
- $w'(e) = \begin{cases} 0 & \text{if } e \in A \\ 1 & \text{if } e \in A' \setminus A \end{cases}$

Remark 2.2.

- The auxiliary TP has a natural feasible solution (send all flow through z, as described above)
- The lower bound is 0 (Every new arc receives 0 flow).
- Combining these 2 observations $\Rightarrow$ An optimal solution exists, and if the optimal solution is 0, then that means we are sending 0 flow through the new arcs. So, any flow must be going through the original arcs. This is an initial solution!

Theorem 2.3

A TP is feasible if and only if its auxiliary TP has optimal value 0

And, this behaviour matches that of an auxiliary problem for the two-phase simplex method!

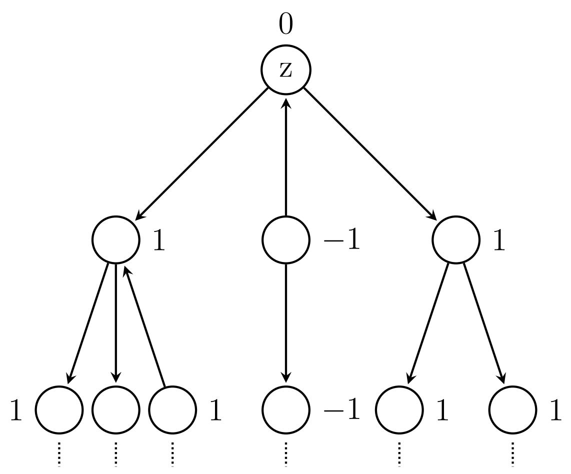
This is great, but still requires us to solve the auxiliary TP in order to determine feasibility of the original. Instead, we aim for a characterization that can determine feasibility using only the original TP. To get there, let's assume that our TP is not feasible, and bootstrap off of what we can determine about our instance through the auxiliary TP.

Assuming the TP is not feasible, the auxiliary TP has an optimal solution with strictly positive optimal value. Let x^* be such a solution that is a also a tree. Let's look at the dual potentials for this solution, setting $y_z = 0$

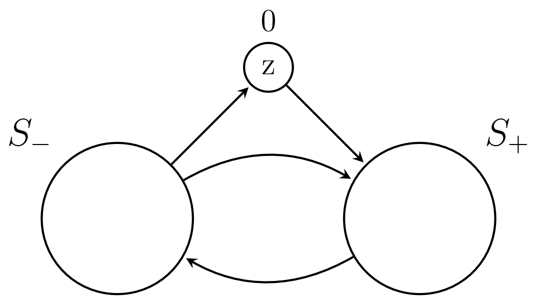
For any node with an incoming arc uz with non-zero flow in x^*, we get that $\bar{w}_{uz} = w_{uz} + y_u - y_z = 0 \Rightarrow 1 + y_u - 0 = 0 \Rightarrow y_u = -1$.

And, for any node with an outgoing arc zv with non-zero flow in x^*, we get that $\bar{w}_{zv} = w_{zv} + y_z - y_v = 0 \Rightarrow 1 + 0 - y_v = 0 \Rightarrow y_v = 1$.

It is easy to check that every other node will have dual potential equal to its parent.

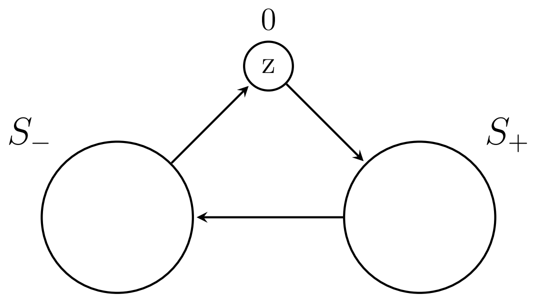


This labelling partitions the nodes in N into two sets according to their dual potentials. Let's give these two sets a name: let S_+ be the nodes with potential $+1$ and S_- be the nodes with potentials -1. We can reduce our focus to these two sets and z, along with the possible arcs that could appear between each entity.



In fact, doing a bit more analysis, we find that no arcs go from S_- to S_+. Why? Suppose there is such an arc e, then $\overline{w}_e \geq 0$, since otherwise, we would have added such an arc to our tree flow. However, doing a bit of calculation, $\overline{w}_e = 0 + (-1) - 1 = -2 < 0$. So, there are no arcs from S_- to S_+!

Performing the same analysis on arcs from S_+ to S_-, we find that the reduced cost: $\overline{w}_e = 0 + 1 - (-1) = 2 > 0$, and so the flow through this arc must be 0. Phew, this is good, since these arcs weren't considered in our tree flow. So, in fact, our diagram should have one less type of arc.



Good, we've done a lot of work now, what can we conclude about the original TP from this? First, the set S_- pushes a non-zero amount of flow through z towards S_+, but receives no flow from S_+ (Since all such arcs have 0-flow). That must mean that $\sum_{v \in S_-} b_v < 0$. But, remember that z is a node we added for the auxiliary TP—it doesn't exist in the original TP. That is, if we remove z, then S_- is a set with no outgoing arcs.

In summary, S_- has negative demand, but no outgoing arcs! So, there's nowhere to push the excess flow out from S_-. This will be our infeasbility characterization.

Theorem 2.4

A TP with digraph $D = (N, A)$ and node demands $b \in \mathbb{R}^N$ is infeasible if and only if there exists $S \subseteq N$ such that $b(S) < 0$ and $\delta(S) = \emptyset$

Proof. ($\Rightarrow$) Formalize what we've done above.

($\Leftarrow$) Suppose there exist $S \subseteq N$ with $b(S) < 0$ and $\delta(S) = \emptyset$. We want to show that the TP is infeasible.

Suppose for a contradiction that there exists a feasible flow x. Then

$$\begin{aligned} 0 > b(S) &= \sum_{v \in S} b_v \\ &= \sum_{v \in S} (x(\delta(\overline{v})) - x(\delta(v))) \\ &= x(\delta(\overline{S})) - x(\delta(S)) \\ &= x(\delta(\overline{S})) \quad (x(\delta(S)) = 0, \text{ since } \delta(S) = \emptyset) \end{aligned}$$

This gives us that $x(\delta(\overline{S})) < 0$, but $x \geq 0$, so $x(\delta(\overline{S})) \geq 0$, giving us our contradiction. □

3 Minimum Cost Flow Problem

3.1 Introduction

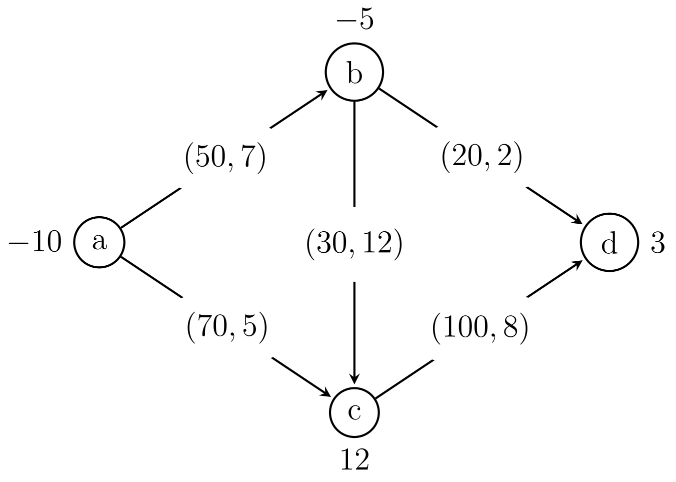
The *Minimum Cost Flow Problem* (MCFP) is quite similar to the transhippment problem. We are given:

- A digraph $D = (N, A)$,
- Node demands $b \in \mathbb{R}^N$,
- Arc costs $w \in R^A$, and
- (the new addition) Arc capacities $c \in R^A$, representing the maximum amount of flow that can be pushed through an arc. (We'll assume that $c_e > 0$, since otherwise, we can just remove the arc)

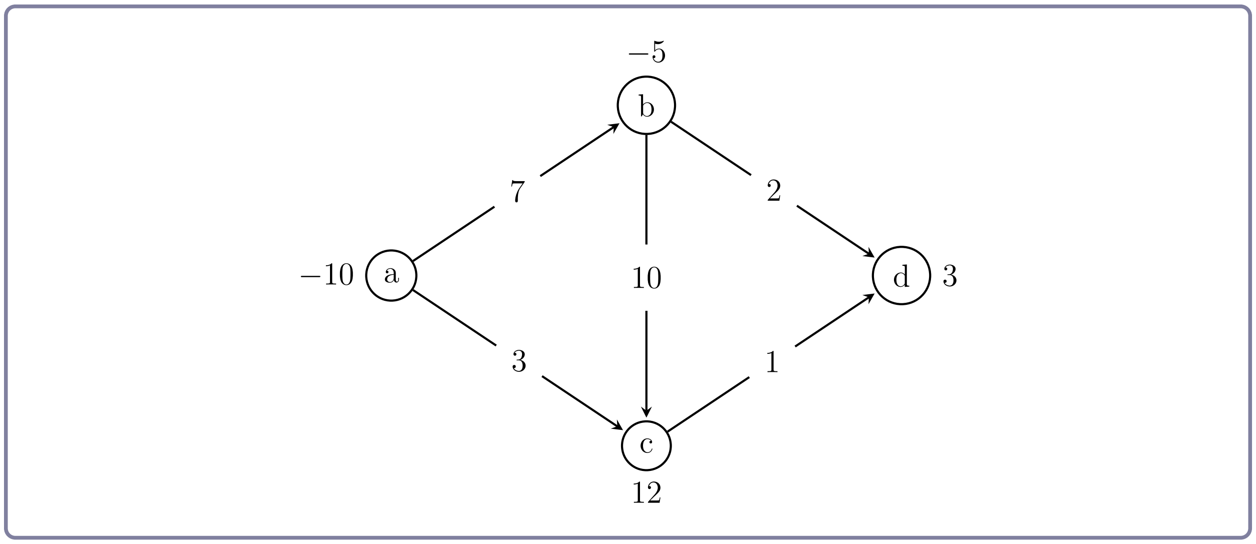
Our goal is for this problem is to find a feasible flow (if it exists) satisfying the node demands *and arc capacities*, while minimizing total cost.

Example 3.1

Below is an example of an MCFP. Arcs are labelled with a pair (w_e, c_e)—the first value is the arc cost, and the second is the arc capacity.



And, this example has a feasible solution:



We can extend our LP formulation from TP for MCFP by including the arc capacity constraints:

$$\begin{aligned} \min \quad & w^{\intercal} x \\ \text{s.t} \quad & x(\delta(\overline{v})) - x(\delta(v)) = b_v \quad \forall v \in N \\ & x_e \leq c_e \quad \forall e \in A \\ & x \geq 0 \end{aligned} \tag{3.1}$$

The dual of this LP is:

$$\begin{aligned} \max \quad & b^{\intercal} y - c^{\intercal} z \\ \text{s.t} \quad & -y_u + y_v - z_{uv} \leq w_{uv} \quad \forall uv \in A \\ & z_{uv} \geq 0 \quad \forall uv \in A \\ & y \text{ free} \end{aligned} \tag{3.2}$$

In this section, we'll tweak the NSM algorithm to solve MCFP instances and find an infeasibility characterization for the problem. Note that this problem cannot be unbounded since $x_e \leq c_e$ places an upper bound on the maximum flow we can push.

3.2 Network Simplex Method for MCFP

Much like with the transshipment problem, we'll develop a similar Network Simplex Method for MCFP.

3.2.1 Finding a Basis

Again, we start by asking the question: *What does a basis look like for MCFP?*

Let's put our constraints from our LP in Equation (3.1) into matrix form. We'll introduce $|A|$ slack variables s_e to get rid of the inequality in the 2nd constraint ($x_e \leq c_e$)

$$\begin{array}{c|c}
 & x_e \quad\quad s_e \\
\begin{array}{c} V \\ A \end{array} & \left[\begin{array}{c|c} M_{|V|\times|A|} & 0_{|V|\times|A|} \\ \hline I_{|A|\times|A|} & I_{|A|\times|A|} \end{array}\right] = \begin{bmatrix} b_v \\ c_e \end{bmatrix}
\end{array}$$

(M is the node-arc incidence matrix, I is the identity matrix, and 0 is the matrix of all zeroes)

We know that M has a rank of $|N| - 1$. We'll extend this by putting all of the other constraints into our basis. There are $|A|$ remaining constraints, so the total rank of our matrix is $|N| - 1 + |A|$. That means that a basis for the MCFP will have $|N| - 1 + |A|$ variables, and all non-basic variables will be set to 0.

Observe that for each arc $e \in A$, at least one of x_e or s_e is in the basis. Why? If both $x_e = s_e = 0$, then $x_e + s_e = 0 < c_e$, since we assume that c_e are all positive. So, this violates feasibility.

Therefore, for each arc $e \in A$, we have the following 3 possibilities:

1. Only x_e is in the basis (and s_e is not)
2. Only s_e is in the basis
3. Both x_e and s_e are in the basis.

How many arcs can there be in case 3? Let's try to count them; denote the number of arcs in case 3 with k. Then, there are $|A| - k$ arcs in case 1 or 2. This gives a total of $2k + |A| - k = k + |A|$ variables. Our basis has size $|N| + |A| - 1$, so $k = |N| - 1$.

Remark 3.1. These $|N| - 1$ arcs don't contain a cycle.

Proof. Exercise! (This number should seem awfully similar to something we've shown before) □

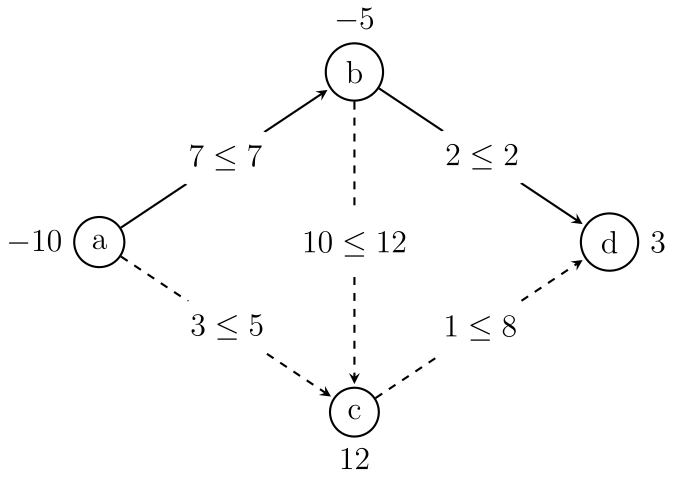
So, these $|N| - 1$ arcs from case 3 must represent a spanning tree!

In Summary:

A basic feasible solution in MCFP consists of a spanning tree T and a feasible flow where all arcs not in T satisfies $x_e \in \{0, c_e\}$. (Since if the arc falls under case 1, then only x_e is in basis and so $s_e = 0 \Rightarrow x_e = c_e$, and similarly for arcs under case 2)

Example 3.2

Below is a feasible flow (labelled with $x_e \le c_e$). The dashed lines are in the spanning tree. So, the variables in the basis are: $x_{ab}, x_{bd}, x_{ac}, s_{ac}, x_{bc}, s_{bc}, x_{cd}, s_{cd}$.



3.2.2 Optimality Conditions

Good, now we know what a basic feasible solution looks like for MCFP, we want to be able to improve it until it's optimal. Again, we'll use the CS conditions to derive optimal conditions.

Using Equations (3.1) and (3.2), we obtain the following 2 CS conditions:

1. $z_{uv} > 0 \Rightarrow x_{uv} = c_{uv}$
2. $-y_u + y_v - z_{uv} < w_{uv} \Rightarrow x_{uv} = 0$

There's a lot of variables here! Let's try to rewrite these conditions in terms reduced costs. (Recall that the reduced cost for an arc uv is $\overline{w}_{uv} = y_u + w_{uv} - y_v$)

Then, the dual constraint: $-y_u + y_v - z_{uv} \le w_{uv}$ can be rewritten as $z_{uv} \ge -\overline{w}_{uv}$, and we'll keep the second constraint: $z_{uv} \ge 0$ as is. Note that in the dual LP, our objective function maximizes $b^\intercal y - c^\intercal z$. In particular, we want to minimize $c^\intercal z$. So, $z_{uv} = \max\{0, -\overline{w}_{uv}\}$

Using this, we can rewrite our two CS conditions:

1. We had $z_{uv} > 0 \Rightarrow x_{uv} = c_{uv}$, but $z_{uv} > 0$ if and only if $\overline{w}_{uv} < 0$. So, this is equivalent to saying:
$$\overline{w}_{uv} < 0 \Rightarrow x_{uv} = c_{uv}$$
2. We had $-y_u + y_v - z_{uv} < w_{uv} \Rightarrow x_{uv} = 0$. But, we can rewrite the premise as $z_{uv} > -\overline{w}_{uv}$ and this is true if and only if $\overline{w}_{uv} > 0$. So, this condition is equivalent to:
$$\overline{w}_{uv} > 0 \Rightarrow x_{uv} = 0$$

In Summary:

1. $\overline{w}_{uv} < 0 \Rightarrow x_{uv} = c_{uv}$
2. $\overline{w}_{uv} > 0 \Rightarrow x_{uv} = 0$

And, if $0 < x_{uv} \leq c_{uv}$, then $\overline{w}_u v = 0$.

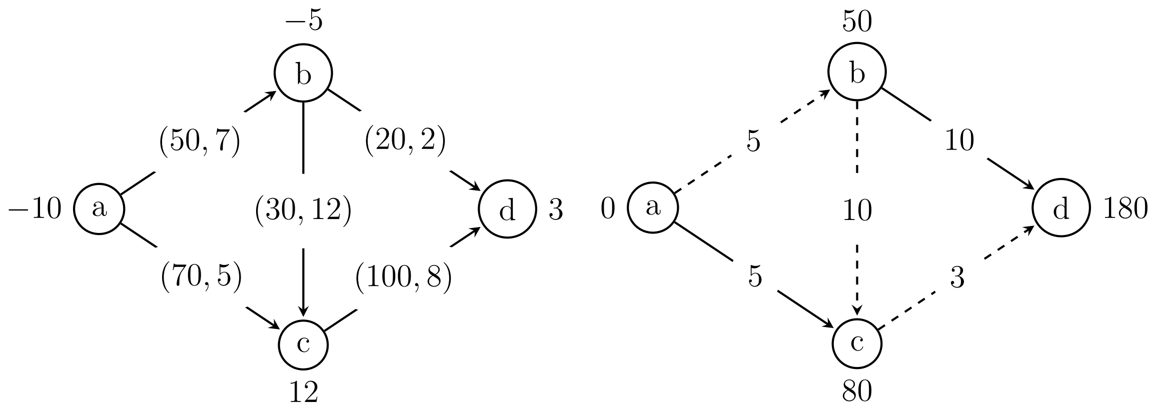
To gain some intuition for the optimality conditions, recall that we introduced an economic interpretation for NSM for TPs in Section 2.3. We can extend the interpretation by capping the amount we can transport through an arc uv by c_{uv}. Then, using the same interpretation of the reduced cost, when $\overline{w}_{uv} < 0$, we can make profit by using arc uv, so we should send as much as possible through uv. The maximum we can send is c_{uv}, so set $x_{uv} = c_{uv}$.

And, if $\overline{w}_{uv} > 0$, we lose money through uv, so we don't send anything through this arc, i.e. $x_{uv} = 0$

Example 3.3: Checking for optimality

On the left is a digraph with arc labelled by (x_e, c_e) and on the right is a feasible flow along with feasible node potentials. The dashed lines are in the spanning tree.

We can check that $\overline{w}_{bc} = w_{bc} + y_b - y_c = -110 < 0$, but $x_{bc} = 10 \neq 12$, so this flow is not optimal



3.2.3 Entering and Leaving Arcs

Now, for the final step in creating NSM for MCFP.

For entering arcs, we should choose arcs that violate either optimality condition (which we'll quickly recall):

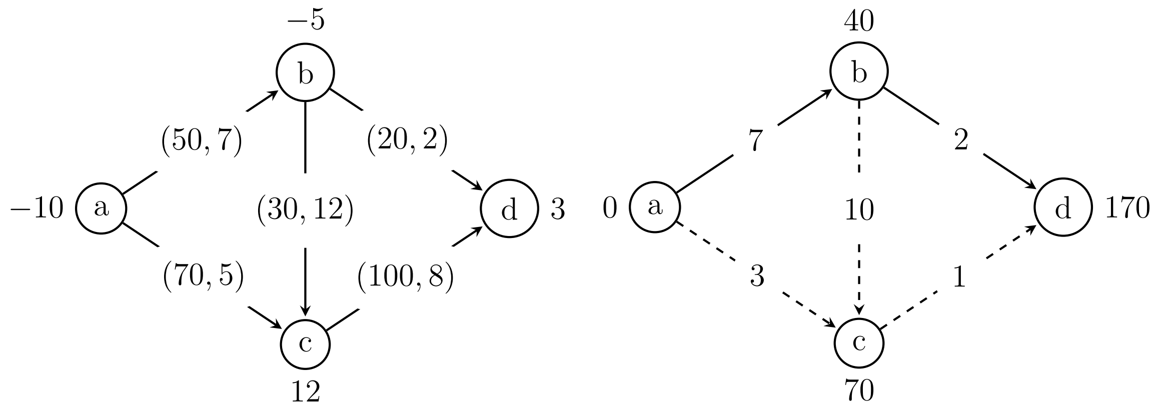
1. $\overline{w}_{uv} < 0 \Rightarrow x_{uv} = c_{uv}$
2. $\overline{w}_{uv} > 0 \Rightarrow x_{uv} = 0$

If the reduced cost $\overline{w}_{uv} < 0$, but $x_{uv} < c_{uv}$, then there's nothing to modify here. We proceed as we did with NSM for TP: Create a cycle using the arc uv, and push as much flow through uv as we can. The arc that leaves the basis is the one that reaches capacity or is reduced to 0 flow.

However, if the reduced cost $\overline{w}_{uv} > 0$, we want to make $x_{uv} = 0$. In this case, we want to *decrease* flow through uv. So, instead of orienting the cycle in the direction of uv, let's orient the cycle in the reverse direction of uv. The leaving arc is, again, the arc that reaches capacity or is reduced to 0 flow.

Example 3.4: Removing flow from the entering arc

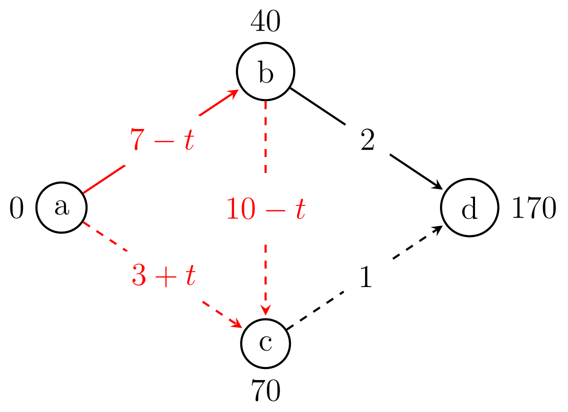
On the left is a digraph with arc labelled by (x_e, c_e) and on the right is a feasible flow along with feasible node potentials. The dashed lines are in the spanning tree.



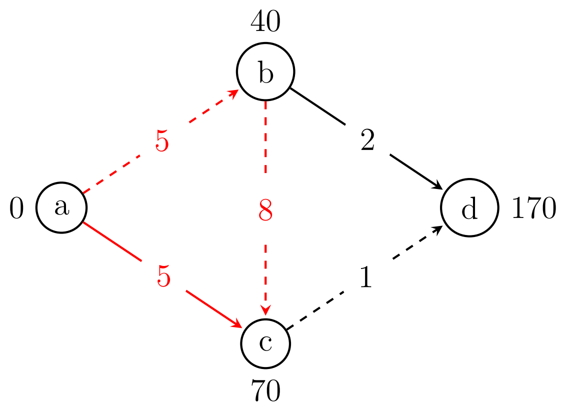
We want to calculate the reduced costs on bd and ab.

- $\overline{w}_{bd} = 20 - 170 + 40 = -110 < 0$ and $x_{bd} = c_{bd}$, so this arc is fine.
- $\overline{w}_{ab} = 50 - 40 = 10 > 0$, but $x_{ab} \neq 0$, so this arc violates the optimality conditions.

So, ab is our entering arc, and we want to *decrease* the flow through this arc. Orient ab as a reverse arc in the cycle acb, and let t be the flow we push through the cycle



We can take away at most 7 and 10 flow from ab and bc respectively. We can add at most 2 flow to ac, since its capacity is 5. So, $t = \min\{7, 10, 2\} = 2$. ac reaches capacity so it leaves the basis and ab enters.



And, one can check that this flow is optimal!

3.2.4 The Algorithm

Algorithm 2: Network Simplex Algorithm For MCFP

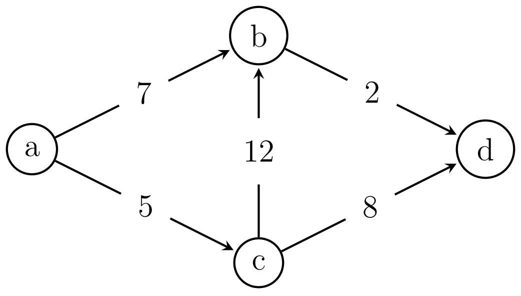
```
  Input:
    • $D = (N, A)$ connected digraph
    • $w \in R^A$ arc costs
    • $b \in R^N$ node demands
    • $c \in R^A$ arc capacities
1 Look for an arc such that
    1. $\overline{w}_e < 0$ and $x_e = 0$, or
    2. $\overline{w}_e > 0$ and $x_e = c_e$
2 $T + e$ has a cycle $C$. Depending on which case $e$ falls under:
    1. Orient $C$ such that $e$ is a forward arc
    2. Orient $C$ such that $e$ is a backward arc
3 Find $t$ that is the minimum of: $\{c_f - x_f \ : \ f$ is a forward arc$\}$ and
  $\{x_f \ : \ f$ is a backward arc$\}$
4 Pick $f$ to be an arc in $C$ that is used to pick $t$
5 Update the flow:
    • Add $t$ to forward arcs in the cycle,
    • Subtract $t$ from backward arcs
6 Replace $T$ with $T + e - f$
7 Repeat until optimal
```

3.3 Infeasibility

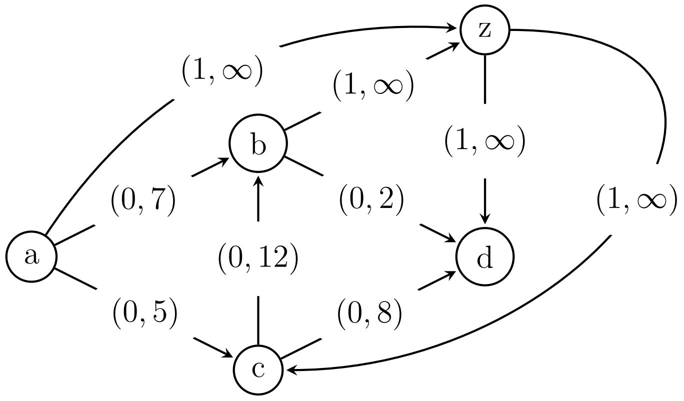
To solve the feasibility problem, we use the same auxiliary digraph as before. The capacities on the original arcs remain the same, and the capacities on the new arcs can be ∞ or some large number (e.g. Sum of all demands)

Example 3.5: Auxiliary Digraph for MCFP

The first digraph is the MCFP labelled with arc capacities, and the second digraph is the auxiliary digraph for this instance.



Original Digraph



Auxiliary Digraph

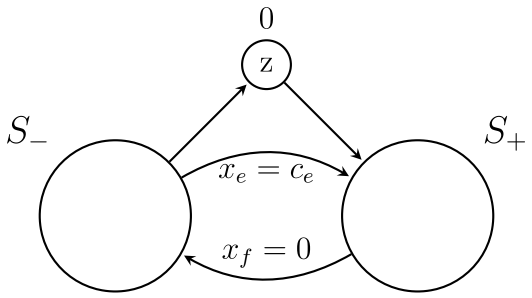
Theorem 3.1

A MCFP is feasible if and only if its corresponding auxiliary MCFP has optimal value 0.

We will construct a infeasbility characterization similar to the one we have for TPs.

Suppose our MCFP is infeasbible, then the auxiliary MCFP has optimal value greater than 0. Take x to be a tree flow. Just like we did back in Section 2.5, set the potentials y so that $y_z = 0$. We'll find that we can, yet again, partition the nodes into the sets S_+ and S_-.

The reduced cost on e is $\overline{w}_e = -2 < 0$, so $x_e = c_e$, and the reduced cost on f is $\overline{w}_f = 2 > 0$, so $x_f = 0$



Now, consider S_+. The net flow on S_+ is just the flow coming in, since all outgoing arcs have 0 flow. All flow coming in from S_- are at capacity *and* there is some (strictly positive) incoming flow from z. However, z is an extra node we added for the auxiliary digraph, so when we remove it, that must mean that S_+ requires more flow than S_- can give. So, $b(S_+) > c(\delta(\overline{S}_+)$.

Theorem 3.2

An MCFP is infeasible if and only if there exists $S \subseteq N$ such that $b(S) > c(\delta(\overline{S}))$

Proof. ($\Rightarrow$) Formalize what we have above!

($\Leftarrow$) Suppose there exists $S \subseteq N$ with $b(S) > c(\delta(\overline{S}))$ and a feasible flow x. Then:

$$\begin{aligned} b(S) = \sum_{v \in S} b_v &= \sum_{v \in S} x(\delta(\overline{v})) - x(\delta(v)) \\ &= x(\delta(\overline{S})) - x(\delta(S)) \\ &\leq c(\delta(\overline{S})) - 0 = c(\delta(\overline{S})) \end{aligned}$$

But that's a contradiction. □

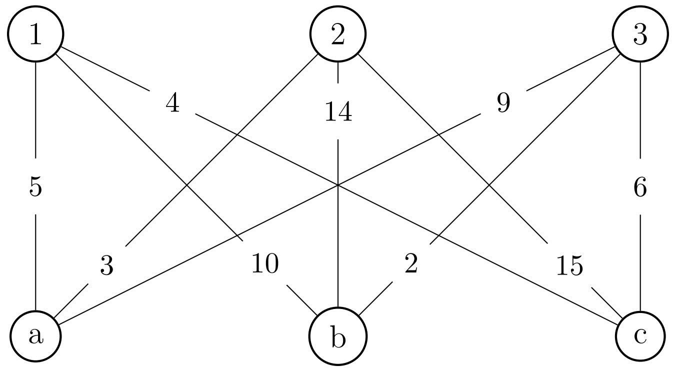
4 Applications of TP and MCFP (TODO)

4.1 Minimum Bipartite Perfect Matching (MBPM)

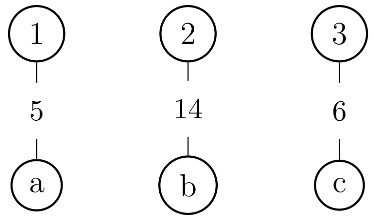
Goal: Given a bipartite graph $G = (V = A \cup B, E)$, with $|A| = |B|$, and edge costs $w \in \mathbb{R}^E$, find a perfect matching in G of minimum total cost.

Example 4.1: Perfect Matchings

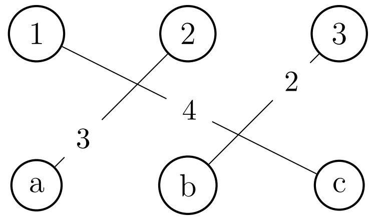
Below is a complete bipartite graph G with bipartiion $\{1, 2, 3\}$ and $\{a, b, c\}$ and 2 perfect matchings of G



Complete Bipartite Graph G



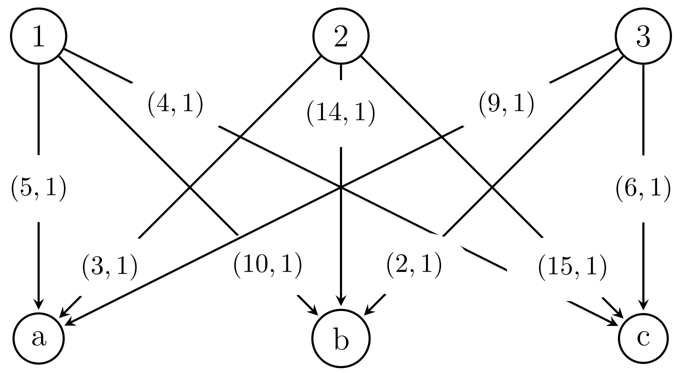
Perfect matching with cost 25



Perfect matching with cost 9

We will formulate this as an MCFP.

Direct each edge from A to B. For each arc, set the capacity to be 1 And, for each node, set $b_v = \begin{cases} -1 \text{ if } v \in A \\ 1 \text{ if } v \in B \end{cases}$. The arc costs stay the same. For example, using the graph from Example 4.1, our resulting digraph for the MCFP instance becomes:



MCFP Formulation for Example 4.1
The arcs are labelled (x_e, c_e)

We want to show a correspondence between optimal solutions to this MCFP and MBPM.

Theorem 4.1: Integrality Condition

If an MCFP has an optimal solution, and all capacities and node demands are integers, then there exists an integral optimal solution

Proof. The proof is based on the NSM algorithm. Since node demands are integral, we can always start off with an integral basic feasible solution. Then, in every iteration, we modify our feasible solution by pushing/removing flow until an arc reaches capacity or 0 flow. But capacities are also integral, so in each iteration where we modify the feasible solution, we remain integral. □

By this theorem, we can say that for our MCFP formulation, there exists an integer-valued optimal solution x. So, $x_e = 0$ or $x_e = 1$ for each arc e and $M = \{e \mid x_e = 1\}$ is exactly a perfect matching because each node in A can push out at most 1 unit of flow, and so is incident with exactly one edge in M.

In the reverse direction, we also note that any PM corresponds to an integral flow. So, our MCFP formulation solves the MBPM problem.

4.2 Airline Scheduling

5 Shortest Dipaths

5.1 Introduction

We'll now switch gears to the problem of finding shortest dipaths between two nodes s and t in our digraph. Solving this problem will require some of the tools we've built so far, and we will work our way towards 3 different algorithms that solve this problem.

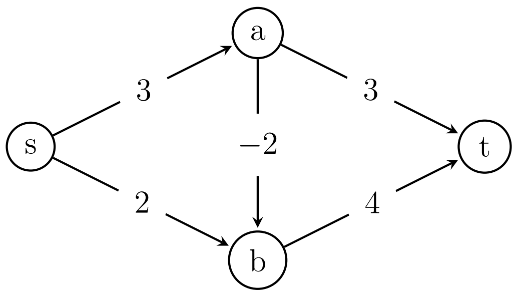
Problem: Given a digraph $D = (N, A)$, arc costs $w \in R^A$ and 2 distinct nodes $s, t \in N$, find the minimum cost s, t-dipath.

Note. Negative arc costs are allowed!

Remark 5.1. Assume that every node can be reached from s via a dipath.

Example 5.1

Below is a digraph with its arcs labelled with its arc cost.



s, a, t is a valid dipath of cost 6.

s, b, t is also a valid dipath of cost 6.

s, a, b, t is a minimum cost dipath of cost 5.

LP Formulation:

Definition 5.1: Characteristic Vector of a Path

Let P be an s, t-dipath. Then, the characteristic vector of P is $x^P \in \mathbb{R}^A$ such that

$$x_e^P = \begin{cases} 1 \text{ if } e \in A(P) \\ 0 \text{ if } e \notin A(P) \end{cases}$$

(where $A(P)$ is the arc set of P)

We can treat x^P as a flow. We set node s to have demand -1 (or supply of 1), node t to have demand 1, and all other nodes to have demand 0. This suggests the following LP formulation for our problem.

$$\begin{aligned} \min \quad & w^\intercal x \\ \text{s.t} \quad & x(\delta(\overline{v})) - x(\delta(v)) = \begin{cases} -1 \text{ if } v = s \\ 1 \text{ if } v = t \\ 0 \text{ otherwise} \end{cases} \qquad \forall v \in N \\ & x \geq 0 \end{aligned} \tag{5.1}$$

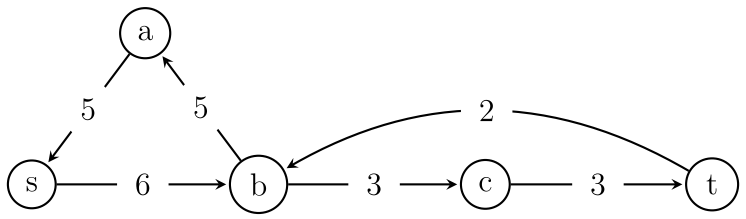
And, this LP gives the dual:

$$\begin{aligned} \max \quad & y_t - y_s \\ \text{s.t} \quad & y_v - y_u \leq w_{uv} \quad \forall uv \in A \\ & y \text{ free} \end{aligned} \tag{5.2}$$

Note. This LP is just a special case of a TP

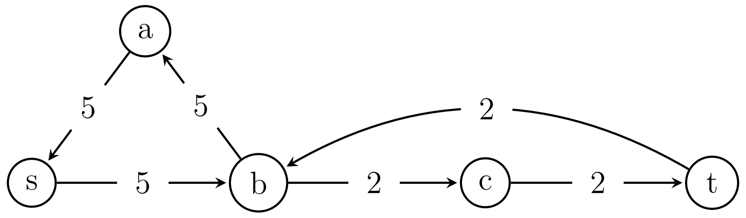
We also note that the characteristic vector of any s,t-dipath is an integral feasible solution. (Any s,t-dipath will send exactly 1 unit of flow from s through the path until it reaches t)

However, the converse of this may not be true! That is, integral feasible solutions to the LP might not be an s,t-dipath For example, we could consider the following digraph and flow:



This flow is a feasible solution to our LP, but we clearly don't have an s,t-dipath.

However, this solution *contains* an s,t-dipath: $s \to b \to c \to t$. Let's call this P. If we subtract x^P from this flow, we obtain the following:



This is a set of dicycles!

Theorem 5.1

If $\overline{x}$ is an integral feasible solution to our LP (*Equation* (5.1)), then $\overline{x}$ is a sum of the characteristic vector of an s,t-dipath and a collection of dicycles.

Proof. Let $F = \{e \in A : \overline{x}_e > 0\}$ (since our flow is feasible, this set will be non-empty). Let $\delta(S)$ be an s,t-cut. The net flow of S is -1 (since the supply is -1), so there must be at least one arc in $\delta(S)$ with non-zero flow, i.e. $|F \cap \delta(S)| \geq 1$.

So, there is an s,t-dipath P using the arcs of F. Let $x' = \overline{x} - x^P$. Since $\overline{x}, x^P$ satisfy the flow constraints, then:

- $x'(\delta(\overline{s})) - x'(\delta(s)) = (\overline{x} - x^P)(\delta(\overline{s})) - (\overline{x} - x^P)(\delta(s)) = -1 - (-1) = 0$, and
- $x'(\delta(\overline{t})) - x'(\delta(t)) = (\overline{x} - x^P)(\delta(\overline{t})) - (\overline{x} - x^P)(\delta(t)) = 1 - 1 = 0$, and
- $x'(\delta(\overline{v})) - x'(\delta(v)) = (\overline{x} - x^P)(\delta(\overline{v})) - (\overline{x} - x^P)(\delta(v)) = 0$, for all other $v \in N$

So, $x'(\delta(\overline{v})) - x'(\delta(v)) = 0$ for all $v \in N$.

So, we've found our s,t-dipath, it remains to show what remains in the flow are all dicycles.

Let $F' = \{e \in A : x'_e > 0\}$. If $F' = \emptyset$, then we're done. So, suppose $F' \neq \emptyset$, and take the longest dipath $v_1 \dots v_k$ in F'.

Since $x'(\delta(\overline{v})) - x'(\delta(v)) = 0$, there is an arc $v_k v_i$ for some $i < k$ (and such v_i cannot be from outside the path since we have taken the longest dipath). This forms a dicycle $C : v_i v_{i+1} \dots v_k v_i$.

Then, $x'' = x' - x^C$ satisfies $x''(\delta(\overline{v})) - x''(\delta(v)) = 0$ for all $v \in N$, and the sum of the flow has decreased by at least 1, so by induction, we are done. □

So, suppose $\overline{x}$ is an optimal integral solution to our LP. By this theorem, we can decompose $\overline{x}$ into an s,t-dipath and a collection of dicycles, that is: $\overline{x} = x^P + x^{C_1} + \dots + x^{C_k}$, where P is an s,t-dipath and $C_1, \dots, C_k$ are dicycles.

If $\overline{x} = x^P$, then we're done, so suppose there is at least one dicycle.

If there are <u>no</u> negative dicycles, then $w^\intercal \overline{x} \geq w^\intercal x^P$, and so P is an optimal solution in the form of an s,t-dipath. We can find P by using the method described in the proof above.

However, if there are negative dicycles, then our problem is unbounded! (Since we can just send arbitrary amounts of flow through the cycle to decrease our cost).

So, our previous techniques can only help us solve this problem if there are <u>no</u> negative dicycles. Let's summarize this.

Theorem 5.2

If there are no negative dicycles, then our LP formulation has an optimal solution that is the characteristic vector of an s,t-dipath

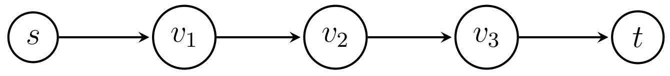
One of our goals in the rest of this section will be to tackle the issue of negative dicycles. Our algorithm should be able to detect that there is a negative dicycle and report it to us.

5.2 Ford's Algorithm

In this section, we will create an algorithm that will find all shortest s, v-dipaths (provided there are no negative dicycles). Note that it will not be able to detect negative dicycles. Although this may not be ideal, we will find that this algorithm serves as a good starting point for the discussion of the algorithms that come after.

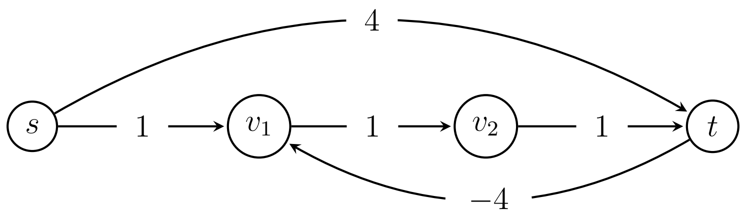
Some motivation for the algorithm:

Consider the following path:



If there are no negative dicycles, then this shortest s, t-dipath also contains the shortest s, v_3-dipath, s, v_2-dipath, and s, v_1-dipath.

It is worth seeing an example where there are negative dicycles:



Here, rather than taking the arc su for a cost of 1, the cheaper path is stu for a cost of 0.

So, in the absence of negative dicycles, let's prove our motivating idea:

Theorem 5.3

Let $D = (N, A)$ be a digraph with arc costs $w \in \mathbb{R}^A$ with no negative dicycles. If $v_1 v_2 \dots v_k$ is the shortest v_1, v_k-dipath, then $v_1 \dots v_i$ is the shortest v_1, v_i-dipath.

Proof. Since there are no negative dicycles, our LP has an optimal solution which is the characteristic vector of an v_1, v_k-dipath P. Then, there is a corresponding optimal dual solution y where all arcs of P are equality arcs (due to CS conditions). Let $P' : v_1 \dots v_i$ for some $i < k$. y is still feasible for the dual LP of the shortest v_1, v_i-dipath problem and any arc in P' is in P, so all arcs of P' are equality arcs. So, the CS conditions are satisfied for the v_1, v_i-dipath problem $\square$

Definition 5.2: Rooted Tree (at s)

A tree T is rooted at s if for all $t \in N(T)$, the unique s, t-path in T is an s, t-dipath

We can build a rooted tree at s representing the shortest s, v-dipaths for all $v \in N \setminus \{s\}$ by solving the single s, v-dipath problem multiple times. However, we can, in fact, do much better than this, and we'll show that we can find them all at once.

First, we'll need two results on rooted trees:

Proposition 5.1

There is an s, t-dipath in D for all $t \in N$ if and only if there exists a spanning tree in D rooted at s

Proposition 5.2

Let T be a spanning tree in D. Then, T is rooted at s if and only if the in-degree of s is 0, and the in-degree of all other nodes is 1 in T

Proof. TODO! □

Definition 5.3: Predecessor

In a spanning tree T, rooted at s, for each node $v \in N \setminus \{s\}$, its predecessor is the unique node u so that uv is an arc in T. We write this as $p_v = u$

5.2.1 The Algorithm

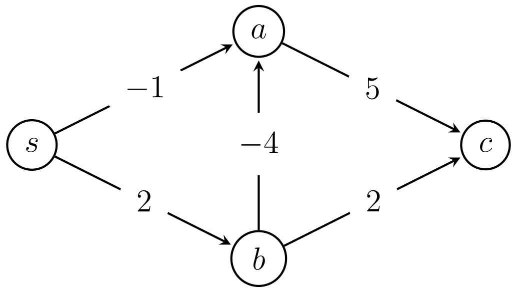
At each step of the algorithm, we maintain a potential and the predecessor of each node.

Algorithm 3: Ford's Algorithm

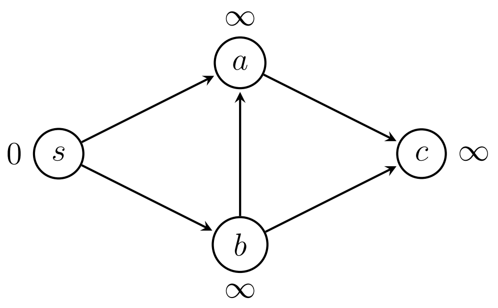
```
1 Initialization:
      • Set $y_s = 0$ and $y_v = \infty \ \forall v \in N \setminus \{s\}$
      • Set $p_v =$ undefined $\forall v \in N$
2 while $y$ is not feasible do
      (a) Find an arc $uv \in A$ where $\overline{w}_{uv} < 0$ (So $y_u + w_{uv} < y_v$)
      (b) Set $y_v = y_u + w_{uv}$ (i.e. Make it an equality arc)
      (c) Set $p_v = u$
```

Example 5.2: Ford's Algorithm

We'll run Ford's algorithm to find the spanning tree rooted at s in the digraph below.

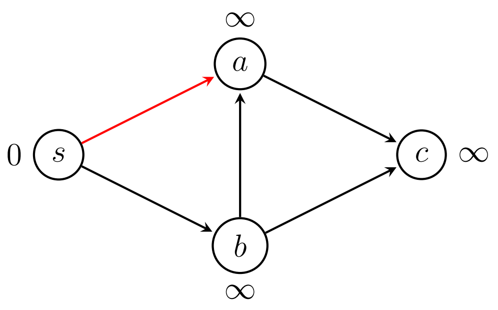


Initialization: We've labelled the nodes with their dual potentials. (Note: UD stands for undefined)



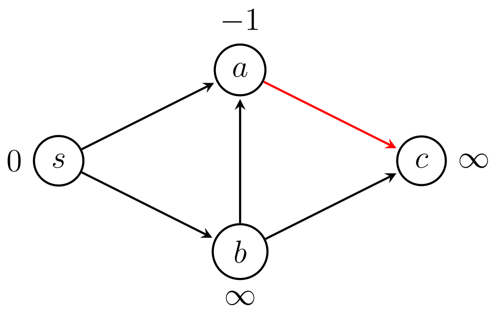
Nodes	a	b	c
Preds.	UD	UD	UD

Iteration 1: $\overline{w}_{sa} = 1 - y_a = -\infty < 0$, so we correct sa by setting $y_a = -1$. The predecessor for a is set to be s



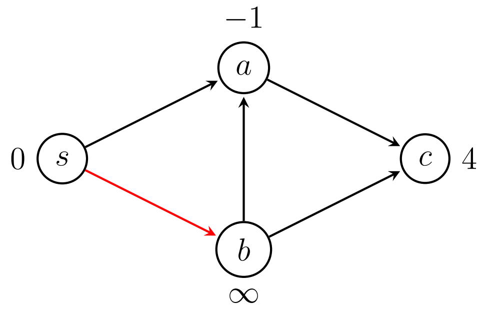
Nodes	a	b	c
Preds.	UD	UD	UD

Iteration 2: $\overline{w}_{ac} = -1 + 5 - y_c = -\infty < 0$, so we correct ac by setting $y_c = 4$. The predecessor for c is set to be a



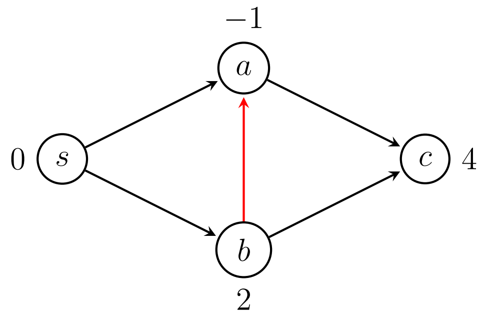
Nodes	a	b	c
Preds.	s	UD	UD

Iteration 3: $\overline{w}_{sb} = 2 - \infty = -\infty < 0$, so we correct sb by setting $y_b = 2$. The predecessor for b is set to be s



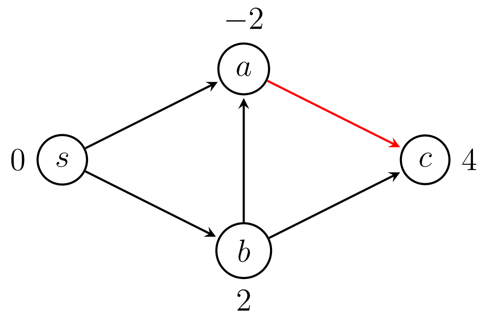
Nodes	a	b	c
Preds.	s	UD	a

Iteration 4: $\overline{w}_{ba} = 2 + (-4) - (-1) < 0$, so we correct ba by setting $y_a = -2$. The predecessor for a is set to be b



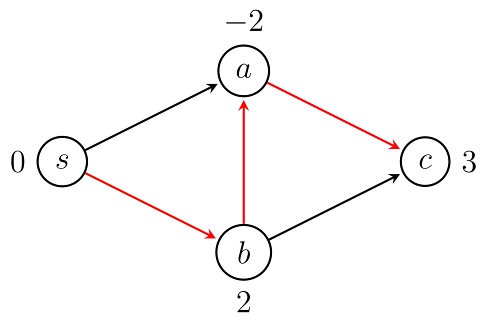
Nodes	a	b	c
Preds.	s	s	a

Iteration 5: $\overline{w}_{ac} = -2 + 5 - 4 < 0$, so we correct ac by setting $y_c = 3$. The predecessor for c stays the same.



Nodes	a	b	c
Preds.	b	s	a

And ... we're done! We can check that the red arcs are all equality arcs and that the node potentials are feasible.



Nodes	a	b	c
Preds.	b	s	a

5.2.2 Correctness

We want to show that the algorithm actually produces a spanning tree rooted at s.

Let's first make the following definition.

Definition 5.4: Predecessor Digraph

Let $D = (N, A)$ be the input digraph for our algorithm. At any point in the algorithm, the predecessor digraph, denoted D_p, is a digraph with node set $N(D_p) = N$ and arc set $A(D_p) = \{p_v v \ : \ v \in N \setminus \{s\}\}$.

Remark 5.2. y_v never increases in the algorithm

And this is since we only modify y_v if we find an arc uv with $y_v > y_u + w_{uv}$. This operation only ever decreases y_v

We'll need the following fact in later proofs:

Proposition 5.3

Throughout the algorithm, $\overline{w}_e \leq 0$ for all arcs in D_p.

Proof. Focus on an arbitrary node v and its unique predecessor $p_v = u$ (if it exists). When v was corrected, $\overline{w}_{uv} = 0$. Then, until u changes, the reduced cost on $\overline{w}_{uv}$ stays non-positive, *and* only a change in y_u can change the reduced cost on $\overline{w}_{uv}$. By the remark, y_u decreases, so the reduced cost $\overline{w}_{uv} = w_{uv} + y_u - y_v$ also decreases. □

Proposition 5.4

If D_p contains a dicycle (at any point in the algorithm), then D contains a negative dicycle and the algorithm does not terminate.

Proof. Suppose we produce a dicycle C in D_p by correcting the arc uv, and let C be $v = v_1, v_2, \ldots, v_k = u, v_1$.

Since we corrected uv, it must have been the case that $y_u + w_{vu} - y_v < 0$. So, $y_{v_k} + w_{v_1 v_k} - y_{v_1} < 0$.

And, we also just showed that $\overline{w}_e \leq 0$ for all $e \in A(D_p)$, so for each $i = 1, 2, \ldots, k-1$: $y_{v_i} + w_{v_i v_{i+1}} - y_{v_{i+1}} < 0$. Taking the sum of these k inequalities, we get: $w_{v_1 v_2} + \ldots + w_{v_k v_1} < 0$, so C is a negative dicycle.

And, if there is a negative dicycle, then there is no feasible potential (Try summing up the dual constraints that correspond to the dicycle). As such, the algorithm will never terminate. □

And, there is one more case where the algorithm will not terminate:

Proposition 5.5

If s has a predecessor, then D contains a negative dicycle and the algorithm does not terminate.

Proof. TODO!/Exercise? □

But, in the absence of negative dicycles, the algorithm will terminate. Does it properly give us a spanning tree rooted at s?

Proposition 5.6

If the algorithm terminates, then D_p is a spanning tree of shortest dipaths rooted at s, with y_v as the cost of the shortest s, v-dipath.

Proof. Since the algorithm terminates, D_p does not contain a cycle and s does not have a predecessor. So, D_p is indeed a spanning tree. Further, since all nodes other than s has a predecessor, the in-degree is 1, so D_p is rooted at s.

Now, all arcs in D_p are equality arcs, since we know $\overline{w}_e \leq 0$ and $\overline{w}_e < 0$ is impossible, since the algorithm has terminated and so y is feasible.

For $v \in N$, let P be the s, v-dipath in D_p. Consider the LP formulation of the shortest s, v-dipath problem. x^P is feasible for the primal, and y is feasible for the dual. The CS conditions hold since all arcs in P are equality arcs, so x^P is optimal. The objective of the dual is $y_v - y_s = y_v - 0 = y_v$, so y_v is the cost of the shortest s, v-dipath. □

5.3 Bellman-Ford

Bellman-Ford is a modification of Ford's algorithm that can handle the case of negative dicycles. As well, we can provide an upper bound on the runtime of this algorithm

5.3.1 The Algorithm

The idea for Bellman-Ford is to go through the arcs in (a finite amount of) "passes". In each pass, we perform the same action as with Ford's algorithm.

Algorithm 4: Bellman-Ford Algorithm

```
1 Initialization:
      • Set $y_s = 0$ and $y_v = \infty \; \forall v \in N \setminus \{s\}$
      • Set $p_v =$ undefined $\forall v \in N$
      • Set $i \leftarrow 0$
2 while $i < |N| - 1$ do
      (a) For each arc $uv \in A$, if $\overline{w}_{uv} < 0$, then set $y_v = y_u + w_{uv}$ and $p_v = u$
      (b) Set $i \leftarrow i + 1$
```

The while loops runs a maximum of $|N|$ times, and in each iteration, we check every arc in A. So in total the runtime of this algorithm is: $\mathcal{O}(|N||A|)$.

If a feasible potential is found once the algorithm terminates, then everything we've proved about Ford's algorithm applies and so we have our optimal rooted tree.

However, if we still have an infeasible potential by termination, then there is a negative dicycle in our digraph.

5.3.2 Correctness

Notation: We'll use d_v to denote the cost of a shortest s, v-dipath.

First, we'll show a lemma:

Lemma 5.1

Suppose there are no negative dicycles. Then, at any point in the algorithm $y_v \geq d_v$

Proof. This clearly holds at initialization since $\infty \geq d_v$ (and d_v is finite).

If $y_v \neq \infty$, then we can trace v back to s using D_p. For each of these arcs $\overline{w}_e \leq 0$. Adding all of these inequalities for each arc in this s, v-dipath, we get:

$$\sum_{ab \in P} \overline{w}_{ab} = \sum_{ab \in P} w_{ab} + y_a - y_b = w(P) - y_v \leq 0 \Rightarrow w(P) \leq y_v$$

But, $w(P) \geq d_v$, since d_v is the cost of the a shortest s, v-dipath, so $d_v \leq y_v$, as required. □

Theorem 5.4

Suppose there are no negative dicycles. After the i-th iteration, if there is a shortest s, v-dipath using at most i arcs, then $y_v = d_v$

Proof. We'll prove this by induction on i. The statement holds for $i = 0$. Assume the statement holds after the i-th iteration.

We want to show that the statement holds for $(i+1)$-th iteration. Pick v which has a shortest s, v-dipath that uses at most $i + 1$ arcs.

If the dipath uses at most i arcs, then by induction, $y_v = d_v$ and by lemma, y_v will not change.

So, suppose the shortest s, v-dipath uses exactly $i + 1$ arcs and let $P : s = v_1 \dots v_{i+1} = v$. Since there are no negative dicycles, $v_1 \dots v_i$ is a shortest s, v_i-dipath that uses i arcs and by induction, $y_{v_i} = d_{v_i}$ after the i-th iteration. This does not change in the $(i + 1)$-th iteration.

Consider $\overline{w}_{v_i v_{i+1}}$, there are three cases:

1. $\overline{w}_{v_i v_{i+1}} = 0$: Then, $y_{v_{i+1}} = y_{v_i} + w_{v_i v_{i+1}} = d_{v_i} + w_{v_i v_{i+1}} = w(P) = d_{v_{i+1}}$. The arc was already corrected before the $(i + 1)$-th iteration, and will not change in this iteration (or future ones).
2. $\overline{w}_{v_i v_{i+1}} > 0$: Then, $y_{v_{i+1}} < d_{v_{i+1}}$. But, this contradicts the lemma, so this case does not happen
3. $\overline{w}_{v_i v_{i+1}} < 0$: Then, in the $(i + 1)$-th iteration, the algorithm corrects this arc, and so $y_{v_{i+1}} = d_{v_{i+1}}$.

By induction, we're done! □

Corollary 5.1

At the end of Bellman-Ford, if y is feasible, then $y_v = d_v$ for all $v \in N$. Otherwise, you can conclude there is a negative dicycle.

Proof. Any shortest dipath uses at most $|N| - 1$ arcs. So, if y is feasible by the end of the $|N| - 1$ passes in Bellman-Ford, there must be no negative dicycles and by theorem $y_v = d_v \; \forall v \in N$. If y is not feasible, then it must be the case that there is a negative dicycle. □

5.4 Dijkstra's Algorithm

Dijkstra's algorithm can only be used on the special case where there are no negative costs in our digraph. It is more efficient than Bellman-Ford.

Recall that in the analysis of Bellman-Ford, we showed that once $y_v = d_v$ for some node v, then y_v will never change. Dijkstra's algorithm takes advantage of this by keeping track of such nodes and wasting computations.

Let's formalize this.

Dijkstra's algorithm keeps a set T of nodes on which the spanning tree is built. In each iteration of the algorithm, we raise the dual potentials of only the non-tree nodes by t. Now, what happens to the reduced costs after raising these potentials?

1. If $u, v \notin N(T)$, then both y_u, y_v increase by t, so $\overline{w}_{uv}$ does not change.
2. If $u, v \in N(T)$, then both y_u, y_v stay the same, so $\overline{w}_{uv}$ does not change.
3. If $u \notin N(T), v \in N(T)$, then y_u increases, while y_v stays the same, so overall $\overline{w}_{uv}$ increases. This is fine.
4. If $u \in N(T), v \notin N(T)$, then y_u stays the same, while y_v increases by t, so overall $\overline{w}_{uv}$ decreases. In this case, to keep feasibility, we want to choose t by picking the smallest $\overline{w}_{uv}$ amongst $\delta(N(T))$. Such an arc becomes an equality arc and is added to T.

5.4.1 The Algorithm

Algorithm 5: Dijkstra's Algorithm

```
1 Initialization:
      • Set $y_v = 0$ for all $v \in N$
      • Set $T = \{s\}$
2 while $T$ is not a spanning tree do
      (a) Pick $uv \in \delta(N(T))$ such that: $\overline{w}_{uv} = \min\{\overline{w}_e : e \in \delta(N(T))\}$
      (b) Set $y_z \leftarrow y_z + \overline{w}_{uv}$ for all $z \in N \setminus N(T)$
      (c) Add $uv$ and $u$ to $T$
```

5.4.2 Correctness

We'll briefly justify this algorithm. Most of the ideas are the same as the analysis on Bellman-Ford.

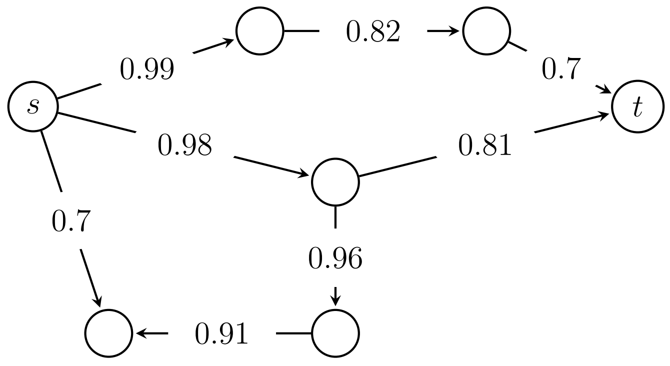
- y is always feasible (y is feasible at initialization, since we don't have negative costs, and stays feasible in each iteration)
- All arcs in T are equality arcs
- The algorithm produces a spanning tree rooted at s

So, the same LP argument gives that the resulting spanning tree must be a tree of shortest s, v-dipaths for all $v \in N$.

5.5 Applications

5.5.1 Network Reliability

Problem: Given a network $D = (N, A)$, each arc e is assigned an associated reliability r_e, where $0 < r_e \leq 1$, representing the probability that e is operational.



For a given dipath P, the reliability of P is $\prod_{e\in P} r_e$, denoted $r(P)$.

Goal: Maximize the reliability amongst all s,t-dipaths

We can apply Dijkstra's algorithm on this problem. Note that $\log(r(P)) = \sum_{e\in P} log(r_e)$. And, we can turn this maximization problem into a minimization problem by multiplying -1 on each arc.

So, for each arc ij, the cost is $-\log(r_{ij})$. Since $0 < r_{ij} \leq 1$, $-\log(r_{ij}) > 0$, so Dijkstra's algorithm applies!

5.5.2 Currency Exchange

Problem: We have a set of currencies and exchange rates r_{uv} between two currencies u and v, representing how much one unit of currency u converts into currency v.

Goal: Exchange a series of currencies back to the original one so that we make a profit. That is, we want a dicycle $v_1v_2\dots v_kv_1$ so that $r_{v_1v_2}r_{v_2v_3}\dots r_{v_kv_1} > 1$

Setup: We use the same trick as with the Network Reliability example:

$$\begin{aligned}\prod_{e\in C} r_e > 1 &\Leftrightarrow \log\prod_{e\in C} r_e > 1\\ &\Leftrightarrow \sum_{e\in C} -(\log r_e) < 1\end{aligned}$$

So, we'll label each arc with cost $-\log r_e$, and since we're interested in finding a negative dicycle, we can just run Bellman-Ford.

6 Maximum Flow

6.1 Introduction

For the maximum flow problem, we are given a digraph $D = (N, A)$, arc capacities $c \in \mathbb{R}^A$, and a source and sink node $s, t \in N$. The goal is to push as much flow as we can from s to t.

<u>LP Formulation:</u>

As always, we'll formulate the LP for this problem:

$$\begin{aligned} \max \quad & x(\delta(s)) - x(\delta(\overline{s})) & \\ \text{s.t} \quad & x(\delta(\overline{v})) - x(\delta(v)) = 0 & \forall v \in N \setminus \{s,t\} \\ & x_e \le c_e & \forall e \in A \\ & x \ge 0 & \end{aligned} \tag{6.1}$$

Our objective function reflects our goal by maximizing flow out of s.

<u>Exercise:</u> Formulate this as an MCFP. (TODO)

There are 2 main classes of algorithms for this problem: the augmenting-path based algorithms and the "push-relabel" type algorithms. We will go over both types. Further, in this section, we will derive and prove the *Max Flow, Min Cut Theorem* and apply it towards some interesting combinatorial results.

6.2 Ford-Fulkerson Algorithm

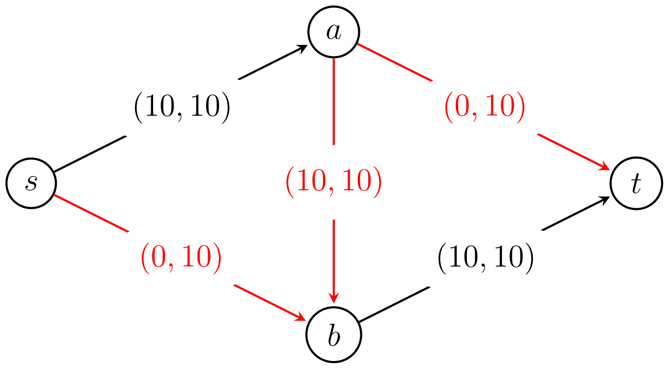
The Ford-Fulkerson algorithm belongs to the class of algorithms that uses augmenting paths to continually improve upon the flow. The idea is based on the following two examples:

1. Suppose we are given the flow (where the arcs are labelled (x_e, c_e)):

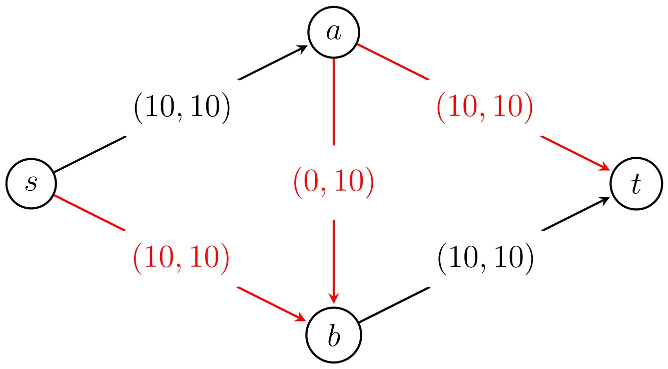
$$\begin{tikzcd} s \arrow[r, "{(1,4)}"] & a \arrow[r, "{(1,2)}"] & t \end{tikzcd}$$

   It is clear that we can push more flow through the arcs not yet at capacity. In this case, we can improve our flow by pushing 1 extra flow through both arcs as we are constrained by the capacity of at

2. Now, suppose we are given the (slightly more complicated) flow:



Here, we can imagine pushing a flow of 10 through the path $s \to b \to a \to t$. Note that ba is a backwards arc with respect to our digraph, so instead of adding flow, we subtract flow. Once we're done, this gives us the flow:



which is better than what we started out with!

To formalize this idea, we define the following digraph:

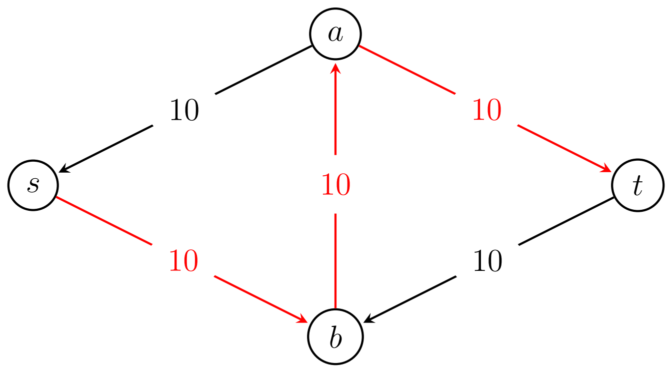
Definition 6.1: Residual Digraph

Given a digraph $D = (N, A)$, arc capacities $c \in R^A$, and a flow x, the corresponding <u>residual digraph</u> (with respect to x) $D' = (N', A')$ is defined:

1. $N(D') = N(D)$
2. For each arc $uv \in A$
   - If $c_{uv} > x_{uv}$, then add uv with capacity (known as the residual) $c_{uv} - x_{uv}$. We call this a forward arc
   - If $x_{uv} > 0$, then add vu with residual x_{uv}. We call this a backward arc.

We may sometimes write $D(x)$ to denote this digraph.

Looking back at the example above, the residual digraph before we modify the flow looks like:



Observe that the dipath $s \to b \to a \to t$ in this residual digraph directly encodes the series of additions and subtractions of flow we performed.

Definition 6.2: Augmenting Path

An augmenting path is an s, t-dipath in D'

6.2.1 The Algorithm

Algorithm 6: Ford-Fulkerson Algorithm

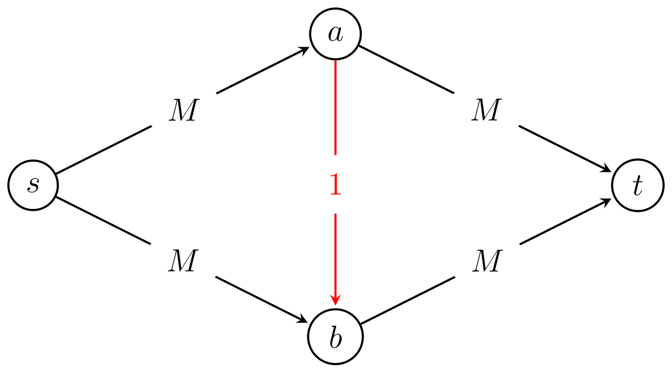
```
1 Initialization:
      • Set $x_e = 0 \,\forall e \in A$
2 while $D'$ has an augmenting path do
      (a) Find an augmenting path $P$ in $D'$
      (b) Let $\gamma$ be the minimum residual in $P$. Push $\gamma$ flow along $P$ in $D$—increasing flow in $D$
          for forward arcs and decreasing flow in $D$ for reverse arcs
      (c) Update $D'$ given this new flow
```

6.2.2 Correctness

First, note that we have yet to give a runtime for this algorithm. Each step in the while loop can be done in polynomial time. However, will the while loop ever terminate?

- If c integral, then at each step, we increase the flow by at least 1, so eventually, the algorithm terminates
- If c rational, then we can multiple c by its GCD to obtain integer capacities
- If c is irrational, it's possible that our algorithm never terminates!

However, even if c is integral, our algorithm may still not terminate in polynomial time with respect to the size of our digraph. Consider the following digraph (labelled with arc capacities and M is some very large number):



Suppose ab is always included in our augmenting path. Then, in each iteration, we will push at most 1 unit of flow, so the algorithm will perform at least M iterations. Since M can be as large as we want, this algorithm runs in *pseudopolynomial* time.

We can fix this by picking *shortest* augmenting paths:

Proposition 6.1: Edmonds-Karp Algorithm

If we pick an augmenting path with the fewest number of arcs, then we only need $|N||A|$ iterations

Proof. TODO!/Exercise □

Now, suppose we use the shortest path rule and the algorithm terminates. We want to show that the resulting flow is a maximum flow.

Theorem 6.1

x is a maximum s,t-flow if and only if there are no augmenting paths

To prove this theorem, we'll need a lemma:

Lemma 6.1

Given digraph $D = (N, A)$, arc capacities $c \in \mathbb{R}^A$, and nodes $s, t \in N$. The value of any s,t-flow is at most the capacity of any s,t-cut.

Proof. Let x be any s,t-flow and $\delta(S)$ be any s,t-cut. The net flow of S is equal to the net flow of s (since the net flow on the other nodes is 0).

Then, the value of flow x is:

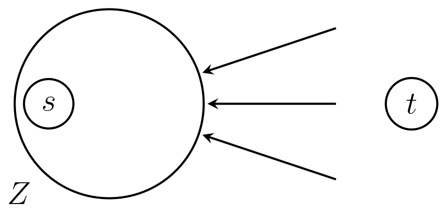
$$\begin{aligned} x(\delta(s)) - x(\delta(\overline{s})) &= x(\delta(S)) - x(\delta(\overline{S})) \\ &\leq x(\delta(S)) \\ &\leq c(\delta(S)) \end{aligned}$$

□

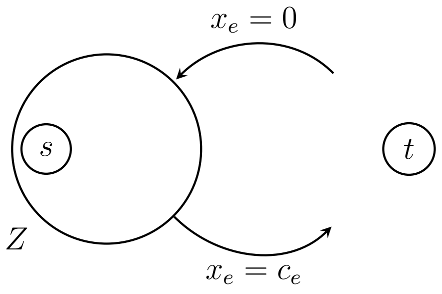
Proof. (of Theorem 6.1)
(⇒) If there exists an augmenting path P, then we can push flow from s to t through P, hence increasing the flow from s to t. So, x is not a maximum s,t-flow.

(⇐) Suppose there are no augmenting paths. Let Z be the set of all nodes reachable from s in D'. Then, $s \in Z, t \notin Z$ and $\delta_{D'}(Z) = \emptyset$.

- Consider an arc uv in $\delta_D(Z)$, since $uv \notin \delta_{D'}(Z)$, $x_{uv} = c_{uv}$.
- Consider an arc ab in $\delta_D(\overline{Z})$, since $ba \notin \delta_{D'}(Z)$, $x_{ab} = 0$.



Residual Digraph D'



Digraph D

So, the net flow out of s is: $x(\delta(s)) - x(\delta(\overline{s})) = x(\delta(Z)) - x(\delta(\overline{Z})) = c(\delta(Z))$, and by the lemma, x is a maximum flow. (And, $\delta(Z)$ is a minimum capacitys,t-cut) □

6.3 Max-flow, Min-cut Theorem (and Combinatorial Applications)

We now introduce a very important theorem in this course: the Max-flow, Min-cut theorem. It's an awesome result that shows how awesome duality is *and* relates two classes of problems in network flow theory.

Theorem 6.2: Max-flow, Min-cut Theorem

The maximum value of an s,t-flow is equal to the minimum capacity of an s,t-cut.

Proof. The maximum flow always exists since the LP (Equation (6.1)) is feasible and not unbounded (the c_e's are finite). Then, by the proof of Theorem 6.1, if x is a max s,t-flow, then there exists a corresponding s,t-cut $\delta(Z)$ with $x(\delta(s)) - x(\delta(\overline{s})) = c(\delta(Z))$ □

We'll now show how we can use this theorem to derive other interesting results in combinatorics.

6.3.1 Menger's Theorem

There are multiple versions of Menger's theorem. Here we'll prove the *arc-disjoint* and *node-disjoint* versions.

Menger's theorem is concerned with the following question: Given a digraph $D = (N, A)$ and nodes $s, t \in N$, how many arcs (or nodes) do we need to remove to disconnect s from t?

Definition 6.3: Disconnect (for arcs)

$A' \subseteq A$ disconnects s from t if there is no s,t-dipath in $D' = (N, A - A')$

Definition 6.4: Arc-disjoint

A set of s,t-dipaths is arc-disjoint if no two of the dipaths share an arc.

Theorem 6.3: Menger's Theorem - Arc-disjoint Version

Given $D = (N, A)$ and nodes $s, t \in N$, the maximum number of arc-disjoint s,t-dipaths is equal to the minimum number of arcs that disconnect s from t.

We'll prove this using the Max-flow, Min Cut theorem and the help of the following proposition

Proposition 6.2: Flow Decomposition

Given $D = (N, A)$, nodes $s, t \in N$ and integer arc capacities, if x is an s,t-flow of value k and x is integral, then x is the characteristic vector of k s,t-dipaths and a collection of dicycles.

Proof. TODO (Incomplete)

When $k = 0$ (i.e. x is a circulation), then x is the sum of characteristic vectors of dicycles (TODO: Expand)

If $k > 0$, then there is an s,t-dipath P (TODO: Expand, similar to previous decomposition theorem) and $x - x^P$ is an s,t-flow of value $k - 1$, so we are done by induction on k. $\square$

Proof. (of Theorem 6.3)
If there are k arc-disjoint s,t-dipaths, then we must remove at least one arc from each of these dipaths. So the maximum number of arc-disjoint s,t-dipaths is at least the minimum number of arcs that disconnect s from t. So, we just have to show equality.

Let our digraph, along with arc capacities of 1 on all the arcs be an instance of the maximum flow problem. Then, by max-flow, min-cut theorem on our digraph, there exists an s,t-flow with the same value k as the capacity of an s,t-cut $\delta(Z)$. We may assume that x is integral, since the capacities are all integral. Then, by Flow Decomposition, x is the sum of k s,t-dipaths and a collection of dicycles.

Since $c = 1$, each arc is used in at most 1 s,t-dipath (splitting a flow of 1 amongst more than one arc would imply that x is a rational flow). So, these s,t-dipaths are all arc-disjoint. If we remove all k arcs in $\delta(Z)$, then s is disconnected from t. □

The node-disjoint version of this theorem is similar! Instead of removing arcs, we remove nodes (and incident arcs) until we disconnect the digraph.

Theorem 6.4: Menger's Theorem - Node-disjoint Version

If st is not an arc, then the maximum number of node-disjoint s,t-dipaths is equal to the minimum number of nodes whose removal disconnects s from t

Note. The set of nodes that we remove is sometimes called an s,t-separating set.

Proof. TODO □

6.3.2 König's Theorem

Definition 6.5: Vertex Cover

A vertex cover is a set of vertices such that every edge has at least one endpoint in the set.

Observation: The size of a matching is at most the size of a vertex cover (Since a vertex cover will use at least one end of each edge in the matching). So, the size of a maximum matching must be at most the size of a minimum vertex cover.

Theorem 6.5: König's Theorem

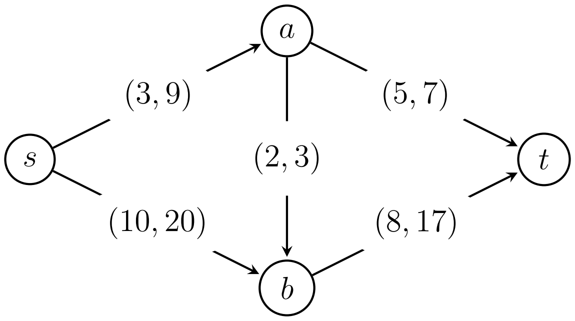
In a bipartite graph, the size of a max matching is equal to the size of a minimum vertex cover

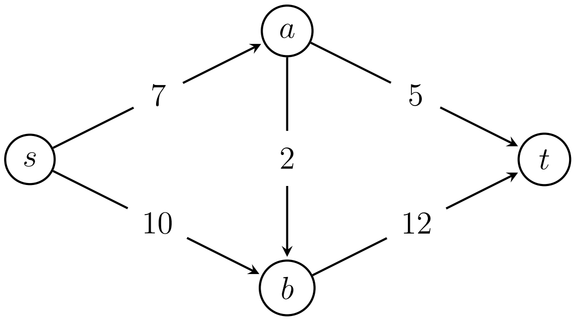
Proof. TODO □

6.4 Flows with Lower Bounds

In this section, we generalize the maximum flow problem by introducing non-negative lower bounds $\ell \in \mathbb{R}^A$ for every arc. This generalization will give us a bit more expressive power in the following section of applications of max flows.

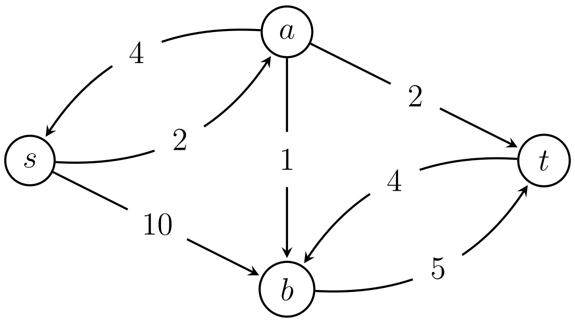
We now enforce that a flow must satisfy $\ell_e \leq x_e \leq c_e$ for all $e \in A$. Below, on the left, we have a digraph where the arcs are labelled with its lower and upper bound (ℓ_e, c_e). On the right, we have a feasible flow for the digraph.





With the addition of the lower bound, it is no longer obvious that a feasible solution exists, since we can no longer just send the all-0 flow. But, let's assume that a feasible flow exists.

In order to use Ford-Fulkerson, we'll have to make a modification to our residual digraph. The node set and forward arcs remain the same. For backward arcs, if $x_{uv} > l_{uv}$, then add vu an arc with residual $x_{uv} - l_{uv}$. This will prevent us from decreasing flow past the lower bound. An example of this is given below using the flow above.



The rest of the algorithm remains the same, and when it terminates, we'll leave it to the reader to verify that x is indeed a maximum s,t-flow. The following lemma may be useful:

Lemma 6.2

For any s,t-flow x and any s,t-cut $\delta(S)$, the value of x is at most $c(\delta(S)) - \ell(\delta(\overline{S}))$

Proof. Same idea as Lemma 6.1 $\square$

Further, the lemma above suggests the following generalized version of the max-flow, min-cut theorem:

Theorem 6.6: Generalized Max-Flow, Min-Cut

Given a digraph with lower bounds, the maximum value of an s, t-flow is equal to the minimum value of $c(\delta(S)) - \ell(\delta(\overline{S}))$ (provided a feasible solution exists)

As an aside, we can also generalize the feasibilty characterization for such networks:

Theorem 6.7: Feasibility Characterization for Flows with Lower Bounds

The network is infeasible if and only if there exists an s, t-cut such that $c(\delta(S)) < \ell(\delta(\overline{S}))$

6.5 Applications

6.5.1 Matrix Rounding

6.5.2 Maximum Closure Problem

6.6 Push-Relabel Algorithm

While proving the correctness of Ford-Fulkerson, we established the optimality condition: we have a maximum flow once we can no longer find an s, t-dipath in the residual digraph. (i.e. There are no more augmenting paths) To that end, Ford-Fulkerson maintains a feasible flow and incrementally increases the flow until this optimality condition is reached.

The Push-Relabel (also called Preflow-Push) algorithm uses a complementary approach. We start with an infeasible flow, but maintain the optimality condition until a feasible flow is reached. Once this occurs, because we have maintained the optimality condition, our feasible flow is, in fact, optimal. We call this intermediate, infeasible flow a preflow:

Definition 6.6: Preflow, Excess

An s, t-preflow x is a flow that satisfies:

- $0 \leq x_e \leq c_e$ (the flow does not exceed capacity), and
- $x(\delta(\overline{v})) \geq x(\delta(v))$ for all $v \in N \setminus \{s, t\}$. (There can be more inflow than outflow.)

The excess at v is $e_x(v) = x(\delta(\overline{v})) - x(\delta(v))$

Given a preflow, our algorithm does one of two actions:

1. **Push:** Push flow out from nodes that have greater than 0 excess...subject to some requirements, or
2. **Relabel:** If said requirements aren't met, we relabel the node.

This label we'll have on the nodes will be called its "height". And the requirement for pushing out excess is always push flow downwards (i.e. From a node of greater height to a node of lesser height). For every node $v \in N$, we associate it with its height $h(v)$.

Definition 6.7: Compatible

A set of heights h is compatible with a preflow x if:

(a) $h(s) = |N|$

(b) $h(t) = 0$

(c) $h(v) \geq h(u) - 1 \quad \forall uv \in A(D')$

Turns out, compatibility is all we need to ensure the optimality condition holds.

Lemma 6.3

If a preflow x and height h are compatible, then the residual digraph D' has no s,t-dipath.

We'll prove this in a bit. Let's state the algorithm first.

6.6.1 The Algorithm

Note. We'll use r_{uv} to denote the residual on uv in D'

Algorithm 7: Push-Relabel Algorithm

```
1 Initialization:
      • Set $h(s) = |N|$ and $h(v) = 0$ for all other nodes
      • Set preflow $x$ to be $x_e = c_e$ if $e \in \delta_D(s)$ and $x_e = 0$ otherwise.
        (That is, push out as much as possible out of $s$)
2 while $\exists u \in N \setminus \{s,t\}$ with $e_x(u) > 0$ do
      (a) If $\exists uv \in A(D')$ where $h(v) = h(u) - 1$, then push $\min\{r_{uv}, e_x(u)\}$ on $uv$.
          ("Push" means to add flow on forward arcs and subtract flow on backward arcs)
      (b) Else increment $h(u)$ by 1 (Relabel)
```

6.6.2 Correctness

First we'll prove Lemma 6.3.

Proof. Suppose there *does* exist an s,t-dipath in D', let's call it $P : v_0 v_1 \ldots v_{k-1} v_k$ (where

$s = v_0$ and $t = v_k$). Since x, h are compatible $h(v_i) \geq h(v_{i-1}) - 1$ for $i = 1, \ldots k$. So in fact:

$$\begin{aligned} h(t) = h(v_k) &\geq h(v_{k-1}) - 1 \\ &\geq h(v_{k-2}) - 1 - 1 \\ &\ldots \\ &\geq h(s) - k \end{aligned}$$

So, $h(s) - k \leq h(t)$, but again, because of compatibility, $h(s) = |N|$ and $h(t) = 0$, so we can rewrite this inequality as $|N| - k \leq 0 \Rightarrow |N| \leq k$, which is a contradiction! ☐

The following corollary is an easy statement to verify:

Corollary 6.1

If x is a feasible flow with compatible heights h, then x is a maximum flow

We stated earlier that the algorithm maintains the optimality condition. Let's show that this is actually true:

Theorem 6.8

The algorithm maintains a preflow and a height function that are compatible with each other.

Proof. This is true at initialization.

Suppose we perform a push operation on arc uv. The resulting flow is a preflow since we only push $\min\{e_x(u), r_{uv}\}$. We also only push when $h(v) = h(u) - 1$, so the heights were compatible before the push. After the push, the only new arc we need to consider in D' is vu. But the heights have not changed after the push operation, so $h(u) = h(v) + 1 \geq h(v) - 1$, as required.

Suppose we perform a relabel operation on node u. It must have been the case that $h(u) \leq h(v)$ for all $uv \in A(D')$. Then, incrementing $h(u)$ by 1 gives $h(u) - 1 \leq h(v)$ for all $uv \in A(D')$. ☐

So, *if* the algorithm terminates, then there is no excess flow nodes other than s, t, so this is a feasible flow. Also, we've just showed that we always maintain a compatible height function. Hence, this flow must also be optimal.

It remains to show that the algorithm *does* terminate. To do so, we'll bound the number of relabel and push operations.

Relabel: The total number of relabel operations is at most $2|N|^2$

We start with a lemma:

Lemma 6.4

If $e_x(u) > 0$, then there is a u, s-dipath in D'

Remark 6.1. $e_x(u) \geq 0 \quad \forall u \in N \setminus \{s, t\}$

Proof. We'll prove the contrapositive. Suppose there are no u, s-dipaths in D'.

Let Z be the set of all nodes $u \in N$ with no u, s-dipath in D'. We note that the cut: $\delta_{D'}(Z) = \emptyset$, as nodes outside of our set can reach s, so if the cut were not empty we would be able to find a dipath to s via nodes in the cut.

So, there are only incoming arcs into Z. For every arc $vu \in \delta_{D'}(\overline{Z})$:

- If $vu \in A(D)$, then $x_{vu} = 0$
- If $uv \in A(D)$, then $x_{uv} = c_{uv}$

Then:

$$\begin{aligned}\sum_{u \in Z} e_x(u) &= \sum_{u \in Z} (x(\delta(\overline{u})) - x(\delta(u))) \quad \text{(By definition of excess)} \\ &= x(\delta(\overline{T})) - x(\delta(T)) \\ &= 0 - c(\delta(T)) \leq 0\end{aligned}$$

But, by remark, $e_x(u) \geq 0$, so it must be that $e_x(u) = 0$ ☐

This helps us to give a bound on the height function.

Proposition 6.3

$h(u) \leq 2|N| - 1$

Proof. Suppose we increase the height $h(u)$ to $2|N|$. This means that $e_x(u) > 0$. By the lemma, there exists a u, s-dipath in D', which has length at most $|N| - 1$. Using the same argument as in the proof of Lemma 6.3, $h(s) \geq h(u) - |N| + 1 \Rightarrow |N| \geq |N| + 1$, which is a contradiction. ☐

So, each node can be relabelled at most $2|N| - 1$ times. There are also at most $|N|$ nodes to relabel. This gives our bound on the total number of relabel operations:

Corollary 6.2

The total number of relabel operations is at most $2|N|^2$

We now bound the maximum number of pushes. Let's start with a definition:

Definition 6.8: Saturating and Non-Saturating Pushes

A push on an arc $uv \in A(D')$ is called:

- saturating if we push r_{uv}, and
- non-saturating if we push $e_x(u)$

We'll bound the number of each type of push individually.

Saturating Push:

Theorem 6.9

The total number of saturating pushes is at most $2|N||A|$.

Proof. Suppose we perform a saturating push on an arc uv. Then,

- It must be that $h(u) = h(v) + 1$, and
- Since the push is saturating, after the push, the arc uv has residual 0 and is removed from the residual digraph.

So, in order to push along uv again, we must first push along vu. This requires at least 2 relabels: the first relabelling v and the second relabelling u. So, between any 2 saturating pushes, there are at least 2 relabel operations. But, each node can be relabelled at most $2|N|$ times, so there are at most $|N|$ saturating pushes on uv

And, since there are at most $2|A|$ possible arcs in D', there can be at most $2|N||A|$ saturating pushes in total. □

Non-Saturating Push:

Theorem 6.10

The total number of non-saturating pushes is at most $4|N|^2|A|$.

Proof. Define the following function:

$$\Phi(x, h) = \sum_{\substack{v \in N \\ e_x(v) > 0}} h(v)$$

Note that at initialization $\Phi = 0$ and otherwise always non-negative.

We will determine the effect on $\Phi(x, h)$ by each type of operation:

- **Relabel**: Relabels are only done on nodes with excess, and so $\Phi(x,h)$ increases by 1 for every relabel. There at most $2|N|^2$ relabels so the maximum increase of Φ due to relabel operations is $2|N|^2$
- **Saturating Push**: Consider a saturating push on uv. This decreases $e_x(u)$ and increases $e_x(v)$. If $e_x(v) = 0$ before the push, then after the push, we add $h(v)$ to $\Phi(x,h)$. Since $h(v) \leq 2|N| - 1$, we add at most $2|N| - 1$. There are at most $2|N||A|$ saturating pushes, so the maximum increase of Φ due to saturating pushes is $2|N||A|(2|N| - 1) \approx 4|N|^2|A|$.
- **Non-Saturating Push**: Consider a non-saturating push on uv. $e_x(u)$ becomes 0, so we subtract $h(u)$ from $\Phi(x,h)$. We may need to add $h(v)$ to $\Phi(x,h)$, but $h(u) = h(v)+1$ so we'll see a decrease in $\Phi(x,h)$ by at least 1.

So, the maximum increase in $\Phi(x,h)$ due to relabels or saturating pushes is $4|N|^2|A|+2|N|^2$. Since $\Phi(x,h)$ stays non-negative and each non-saturating push decreases $\Phi(x,h)$ by at least 1, there are at most $4|N|^2|A|+2|N|^2$ non-staturating pushes. (And, $4|N|^2|A|$ is the dominating term) □

Corollary 6.3

The algorithm takes at most $2|N|^2 + 2|N||A| + 4|N|^2|A|$ operations

Example 6.1

TODO!

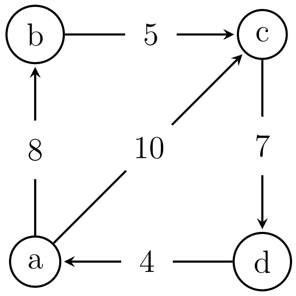
7 Global Minimum Cut

7.1 Hao-Orlin

Our goal in this section is to find a minimum capacity non-trivial cut in $D = (N, A)$.

Example 7.1

Suppose we are given the following directed graph:



The cut $\delta(S)$ where $S = \{a, b\}$ has capacity $c(\delta(S)) = 10 + 5$ (We ignore incoming arcs)

The cut $\delta(T)$ where $T = \{c, d\}$ has capacity $c(\delta(T)) = 4$

Naive Algorithm: We can pick all possible pairs of nodes s, t and run a maximum flow algorithm on each pair. There are $\mathcal{O}(|N|^2)$ such pairs, so we'll have to do this $\mathcal{O}(|N|^2)$ times.

But: we can be more efficient than that. Suppose we pick any $s \in N$. s is either in the set that makes the global minimum cut, or it isn't. So, we just need to find all s, t-cuts and t, s-cuts and output the minimum. This only takes $2|N| - 1$ iterations of a maximum flow algorithm.

We can do better still: Suppose we had an algorithm that gave the minimum cut containing s (efficiently). Then, we only need to run this algorithm for every $s \in N$ to find the global minimum cut.

We'll call this the minimum s-cut problem:

Definition 7.1: s-cut

Given $s \in N$, an s-cut has the form $\delta(S)$ where $s \in S$ and $S \neq N$

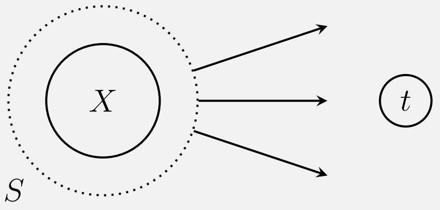
The goal of Hao-Orlin is to solve the minimum s-cut problem quickly. It is a modification of the push-relabel algorithm

7.1.1 Preparatory Work

We'll need to tinker with the push-relabel algorithm a bit in order to make the algorithm do what we want. Here, we'll make some definitions and lay the groundwork for Hao-Orlin.

Definition 7.2: X, t-cut

Given $X \subseteq N$ and $t \notin X$, an $\underline{X, t\text{-cut}}$ has the form $\delta(S)$, where $X \subseteq S, t \notin S$



An X, t-cut

Suppose we are given the following algorithm:

Algorithm 8: Generic Algorithm for minimum s-cut

```
1 Set $X = \{s\}$
2 while $X \neq N$ do
      (a) Pick $t \notin X$, find a minimum $X, t$-cut
      (b) Add $t$ to $X$
3 Output minimum cut over all cuts found.
```

We claim that this helps us find the minimum s-cut.

Let $\delta(S)$ be a minimum s-cut. Consider the first time we pick some $t \notin S$ in Algorithm 8. The algorithm gives us a minimum X, t-cut $\delta(S^*)$ and since we started with $X = \{s\}$, this is also an s-cut. But, $\delta(S)$ is also an X, t-cut. So, both cuts must have the same capacity.

In fact, Hao-Orlin will build on this algorithm. But first, we need to modify a couple more aspects of the push-relabel algorithm.

Definition 7.3: X-preflow

For $X \subseteq N$, an $\underline{X\text{-preflow}}$ is a flow x where $e_x(v) \geq 0$ whenever $v \notin X$

We will keep our definition of $\underline{\text{height}}$ the same. But, we will need to redefine compatibility.

Definition 7.4: Compatible

Heights h are <u>compatible</u> with an X-preflow if:

1. $h(v) = |N|$ for all $v \in X$
2. $h(t) \leq |X| - 1$
3. $h(v) \geq h(u) - 1$ for all $uv \in D'$

And we'll introduce two new concepts:

Definition 7.5: Level, Cut-Level

A <u>level</u> k consists of all nodes with height k, denoted $H(k)$

A <u>cut-level</u> is a level k where no arc goes from $H(k)$ to $H(k-1)$ in D'

These two definitions will allow us to rewrite the following lemma using terminology that will show up in our final algorithm:

Lemma 7.1

If $\delta(S)$ is an X,t-cut with $\delta_{D'}(S) = \emptyset$ and $e_x(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$, then $\delta(S)$ is a minimum X,t-cut.

Using the definitions of level and cut-level, we can rewrite this lemma to the following:

Corollary 7.1

If ℓ is a cut level and $e_x(v) = 0$ for all v with $h(v) < \ell$, except t, then $\{v \mid h(v) \geq \ell\}$ is a minimum X,t-cut.

Let's prove the lemma:

Proof. We want to show that x is maximum X,t-flow.

First, let's establish some bounds. By Lemma 6.1, we can bound the net outflow of S:

$$x(\delta(S)) - x(\delta(\overline{S})) \leq c(\delta(S))$$

This flow goes to $N \setminus S$ so:

$$\sum_{v \in N \setminus S} e_x(v) = x(\delta(S)) - x(\delta(\overline{S})) \leq c(\delta(S))$$

Finally, the excess at t is at most the inflow from all nodes in $N \setminus S$, so:

$$e(t) \leq \sum_{v \in N \setminus S} e_x(v) = x(\delta(S)) - x(\delta(\overline{S})) \leq c(\delta(S)) \tag{7.1}$$

We're done if we can show that the bounds in Equation (7.1) are tight. This shows that x attains the upper bound and t obtains all the excess.

Using the assumption $\delta_{D'}(S) = \emptyset$, it must be that:

$$x(\delta(S)) - x(\delta(\overline{S})) = c(\delta(S))$$

Using the assumption that $e_x(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$:

$$e(t) = \sum_{v \in N \setminus S} e_x(v)$$

So, $\delta(S)$ is a minimum X, t-cut. $\square$

7.1.2 The Algorithm

Idea: We want to use Corollary 7.1. To that end, we'll keep track of a cut-level ℓ and aim to get rid of excess on nodes below ℓ while running push-relabel.

Note. We'll keep an invariant: Non-empty levels will be kept consecutive (This will be useful later)

Algorithm 9: Hao-Orlin Algorithm

```
1 Initialization:
      • Set $X = \{s\}$ ($s$ arbitrary)
      • Choose $t \in N \setminus X$
      • Set $h(s) = |N|$ and $h(v) = 0$ otherwise
      • Set the cut level $\ell$ to be $|N| - 1$
      • Send as much flow out of $X$ as possible
2 while $X \neq N$ do
          1. Run push-relabel with the following exceptions:
              (a) Only select nodes $v$ such that $e(v) > 0$ and $h(v) < \ell$
              (b) If we want to relabel $v$ and $v$ is the only node with height $h(v)$.
                  Do not relabel. Instead, set $\ell = h(v)$
              (c) If we want to relabel $v$ to $\ell$, then relabel and reset $\ell = |N| - 1$
          2. When no node satisfies $e(v) > 0$ and $h(v) < \ell$:
              (a) Store the cut $\{v : h(v) \geq \ell\}$ (This is a min $X, t$-cut)
              (b) Add $t$ to $X$
              (c) Set $h(t) = |N|$, send as much flow out of $X$ as possible, and pick the new $t$ to be
                  the node with the lowest height (This step maintains comp)
              (d) Reset $\ell = |N| - 1$
3 Output minimum cut over all cuts found.
```

Some notes on the exceptions:

1. Step 1a ensures that we can apply Corollary 7.1
2. Step 1b maintains the invariant
3. Step 1c is used to maintain the cut level. For example, suppose there is an arc uv where $h(u) = h(v) = \ell - 1$, and we want to relabel u to ℓ. This would violate the condition as uv is an arc going across $H(\ell)$ and $H(\ell - 1)$.

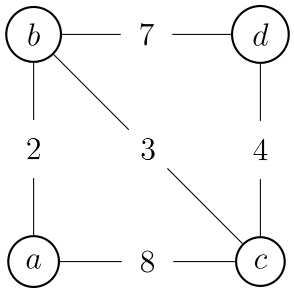
7.1.3 Correctness

7.2 Karger's Randomized Algorithm

We'll now consider minimum cuts in undirected graphs. The setup here is: Given an undirected graph $G = (V, E)$ and edge capacities $c \in \mathbb{R}^E_+$, we want to find the minimum capacity non-trivial cut in G.

Example 7.2

Suppose we are given the following graph:



The cut $\delta(\{a\})$ has capacity $c(\delta(\{a\})) = 2 + 8 = 10$.

The cut $\delta(\{a, c\})$ has capacity $c(\delta(\{a, c\})) = 2 + 3 + 4 = 10$.

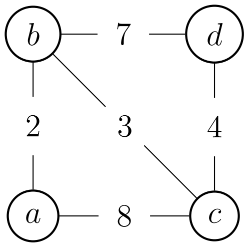
We first define the operation of edge contraction:

Definition 7.6: Edge Contraction

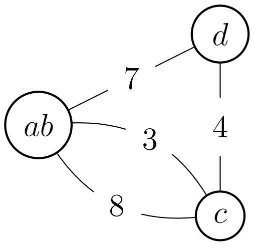
A contraction on edge $e = uv$ is removing the edge e, then merging u and v into one vertex w and the edges incident to u or v are now incident to w

Example 7.3: Edge Contraction

An edge contraction of the edge ab on the left graph gives the graph on the right:



Original Graph



Resulting Graph

The idea for Karger's algorithm is to repeatedly contract edges until only 2 vertices remain. The edges between the remaining two super-vertices represents a cut (and for some probability, it may also be a minimum cut).

Algorithm 10: Karger's Algorithm

```
1 while $|V| > 2$ do
      (a) Pick an edge $uv$ with probability $\frac{c_{uv}}{\sum_{e \in E} c_e}$
      (b) Contract $uv$
2 Let $(S, V \setminus S)$ be the vertex sets corresponding to the 2 vertices remaining. return
  $\delta(S)$
```

This is a randomized algorithm, so we want to ask how often the resulting cut will be a minimum cut. We'll need the following remark:

Remark 7.1. Let $G = (V, E)$ be an undirected graph.

$$\sum_{e \in E} c_e = \frac{1}{2} \sum_{v \in V} c(\delta(\{v\}))$$

Proof. Every edge uv appears in 2 such cuts: $\delta(\{u\})$ and $\delta(\{v\})$. The result follows. □

And now to the main result:

Theorem 7.1

Let $\delta(S)$ be a specific minimum cut. The probability that the algorithm produces $\delta(S)$ is at least $\frac{1}{\binom{|V|}{2}}$

Proof. Consider the $(k+1)$-th iteration of our algorithm when we have $|V| - k$ vertices remaining. Assume that we have not yet contracted any edge in $\delta(S)$ in iteration $1, \ldots, k$. Call our current graph $G' = (V', E')$. The probability of selecting an edge in $\delta(S)$ is:

$$\begin{aligned}
\frac{c(\delta(S))}{\sum_{e \in E'} c_e} &= \frac{c(\delta(S))}{\sum_{v \in V'} c(\delta(\{v\}))} && \text{(By remark)} \\
&\leq \frac{c(\delta(S))}{\sum_{v \in V'} c(\delta(S))} && \text{(Since } \delta(S) \text{ is a minimum cut)} \\
&= \frac{c(\delta(S))}{\frac{|V|-k}{2} c(\delta(S))} \\
&= \frac{2}{|V| - k}
\end{aligned}$$

So the probabilty that $\delta(S)$ survives this iteration is at least $1 - \frac{2}{|V|-k}$

The largest possible k is $|V| - 3$ (since the algorithm terminates when there are only 2 vertices remaining). Overall, the probability that $\delta(S)$ survives every contraction is at least:

$$\begin{aligned}
&\prod_{k=0}^{|V|-3}(1-\frac{2}{|V|-k}) \\
&= \prod_{k=0}^{|V|-3}(1-\frac{|V|-k-2}{|V|-k}) \\
&= \frac{|V|-2}{|V|}\frac{|V|-3}{|V|-1}\frac{|V|-4}{|V|-2}\cdots\frac{2}{4}\frac{1}{3} \\
&= \frac{2}{|V|(|V|-1)} \\
&= \frac{1}{\binom{|V|}{2}}
\end{aligned}$$

□

This probability is not too low. But, if we were to only run the algorithm once, we would most likely not obtain any minimum cut!

To boost this probability, we want to re-run the algorithm at least $\alpha|V|^2$, and output the smallest capacity cut that we find.

Corollary 7.2

The probability that the algorithm produces a specific minimum cut $\delta(S)$ after $\alpha|V|^2$ runs is at least $1-e^{-2\alpha}$ $(\alpha \geq 1)$

Proof. We'll use the inequality $1-x \leq e^{-x}$.

The probability of not producing $\delta(S)$ is at most:

$$\begin{aligned}
\left(1-\frac{1}{\binom{|V|}{2}}\right)^{\alpha|V|^2} &\leq \left(1-\frac{2}{|V|^2}\right)^{\alpha|V|^2} \\
&\leq \left(e^{-\frac{2}{|V|^2}}\right)^{\alpha|V|^2} \\
&= e^{-2k}
\end{aligned}$$

And so our desired result follows. □