week06-01–graph-models

February 20, 2025

1 George McNinch Math 87 - Spring 2025

2 Week 6

3 Graph models (max flow/min cut)

4 Graphs more generally

Recall that when we previously found *network flows* – a form of *graph model* – to be useful in setting up linear programs.

We are now investigate *Graph Models* more generally general and discuss methods referred to as *max flow and min cut* that use duality to understand possible solutions.

5 Example (aka trains and the cold war?)

In the mid 1950s, Air Force Researchers Harris and Ross closely studied the rail network of the Soviet Union and the Eastern Bloc countries. They identified 44 *districts* between which rail cargo moved and 105 links between these districts in the rail network. From this, they were able to calculate the maximum shipping capacity from Russia to Europe and, most importantly, where the *bottleneck* in the network was that limited the capacity. This report was classified until 1999…

This problem, and many others, can be modelled naturally using graphs.

6 Graphs (definitions)

Definition A graph $G = (V, E)$ is a pair (V, E) consisting of a set V of *nodes* – or vertices –, and a set E of edges. Each edge $e \in E$ connects two vertices $v, w \in V$.

We say that G is a *directed graph* if E consists of *directed edges*; in this case E is a subset of the cartesian product $V \times V$ – in other words, an edge $e \in E$ can be identified with an ordered pair $(v, w) \in V \times V$; this should be interpreted as indicating that the edge E goes *from* the vertex v and *to* the vertex w. When such graphs are *drawn*, typically the edge $e = (u, v)$ is indicated by an *arrow* from node u to node v. We might also write the symbol $u \to v$ for this edge.

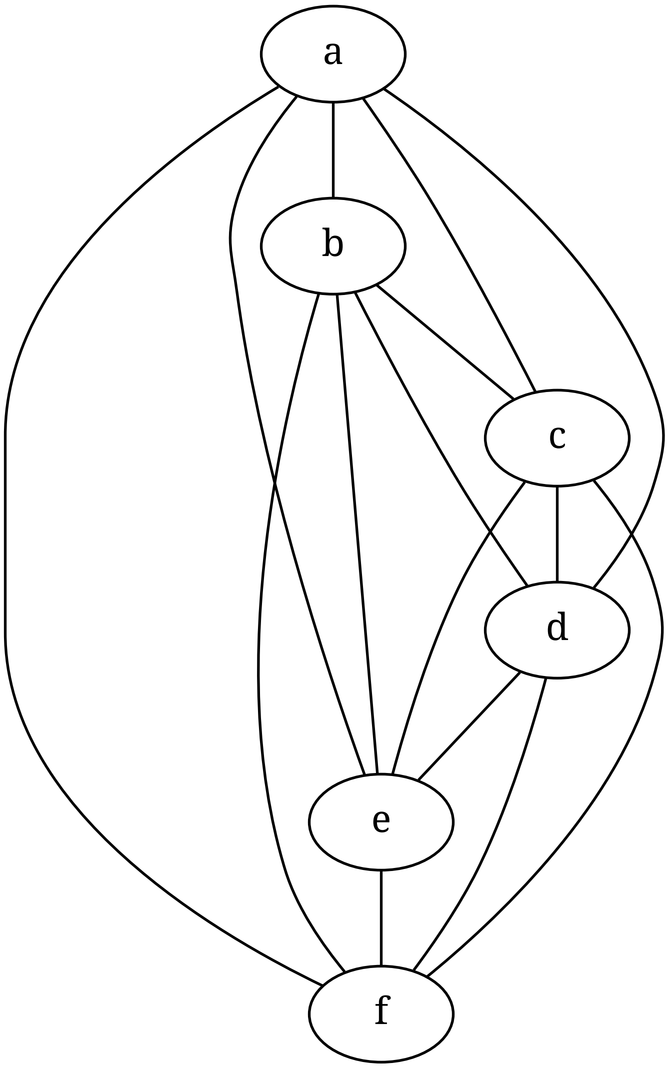
We say that G is an *undirected graph* if E consists of *undirected edges*. For $u, v \in V$, the pair determines an undirected edge $[u, v]$, and $[u, v] = [v, u]$. When we draw an undirected graph, an edge $[u, v]$ is indicated just by an undecorated line between u and v.

Finally, G is a *weighted (directed or undirected) graph* if we are given a function $c : E \to \mathbb{R}$ which assigns a *weight* or *capacity* $c(e)$ to each edge $e \in E$. When drawing a weighted graph, we typically *label* the edges with their weights.

7 Here are a few examples:

8 Undirected graph

[1]:
```
from graphviz import Graph,Digraph
import itertools as it

## https://www.graphviz.org/
## https://graphviz.readthedocs.io/en/stable/index.html

# make the regular graph on the vertices I
def gg(I):
    gg = Graph('regular graph')
    for i in I:
        gg.node(f"{i}")

    #  in the regular graph, there is an edge between every pair of distinct
    #  vertices
    E = [(i,j) for (i,j) in it.product(I,I) if I.index(i) < I.index(j)]

    for (i,j) in E:
        gg.edge(f"{i}",f"{j}")

    return gg

gg(['a','b','c','d','e','f'])
```
[1]:

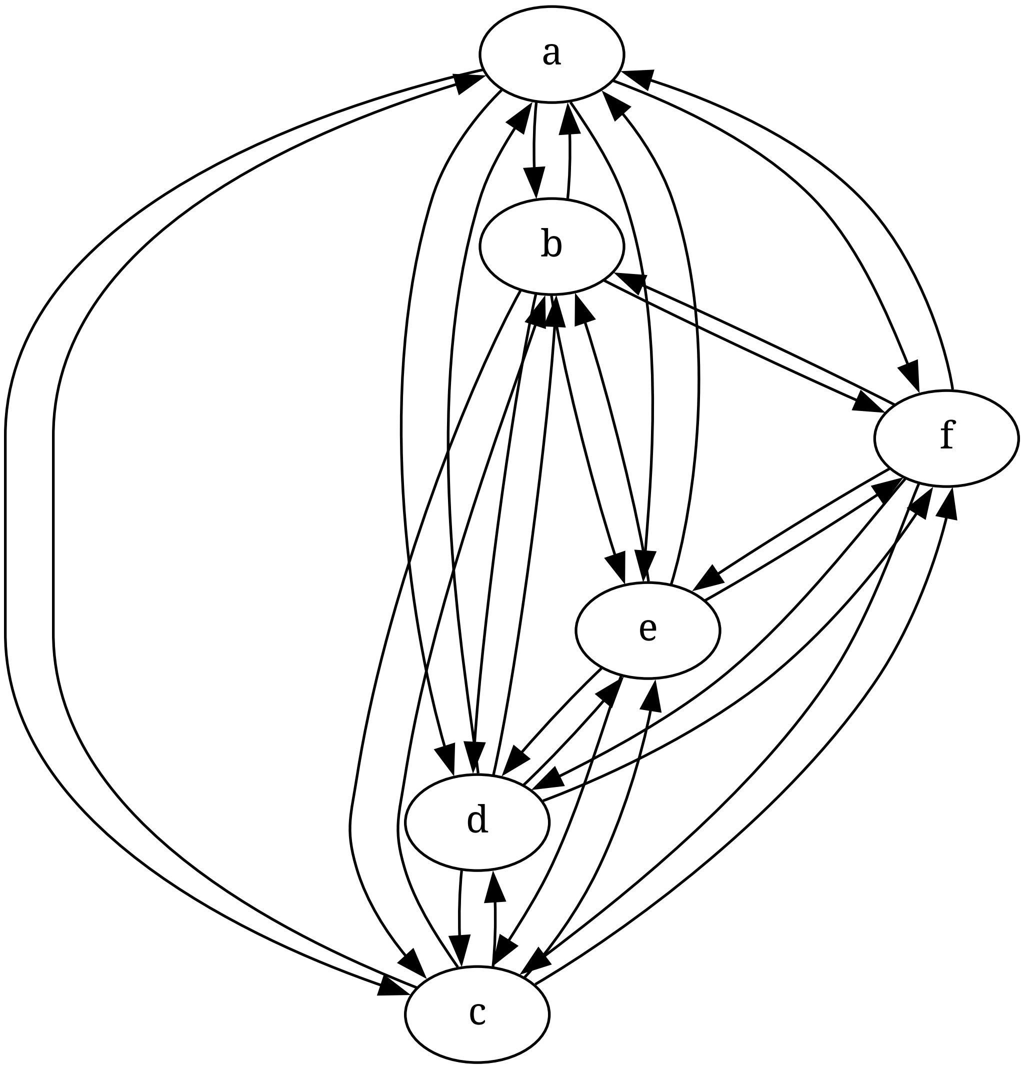


9 Directed graph

[2]:
```
# make the directed graph on the vertices I
def dg(I):
    dg = Digraph('di-graph')

    for i in I:
        dg.node(f"{i}")

    E = [ (i,j) for (i,j) in it.product(I,I) if i != j ] # if I.index(i) < I.
 ↪index(j) ]
#    E = [ (i,j) for (i,j) in it.product(I,I) if I.index(i) < I.index(j) ]
```

```
    for (i,j) in E:
        dg.edge(f"{i}",f"{j}")

    return dg

dg(['a','b','c','d','e','f'])
```

[2]:



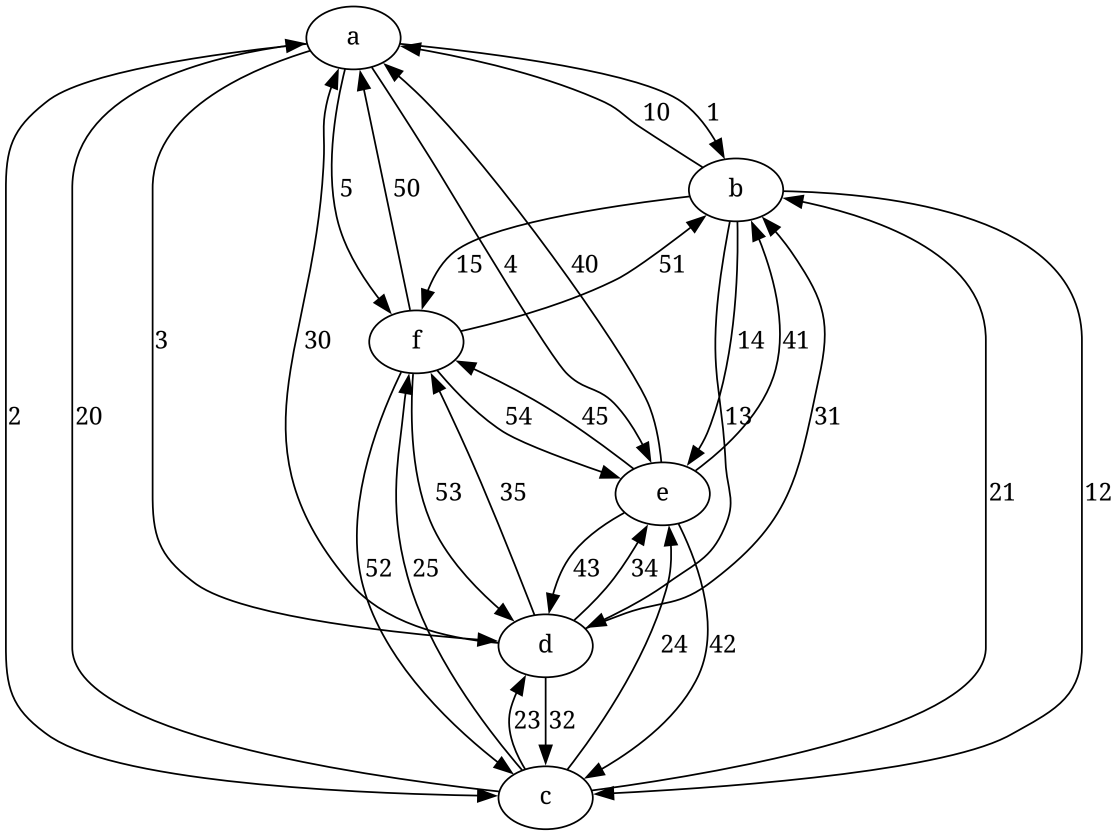
10 Weighted di-graph:

[3]:

```
# make the regular weighted directed graph with vertices I
# and weights determined by the function f
```

```
def wdg(I,f):
    wdg = Digraph('weighted di-graph')

    for i in I:
        wdg.node(f"{i}")

    E = [ (i,j) for (i,j) in it.product(I,I) if i != j ] # if I.index(i) < I.
 ↪index(j) ]
#    E = [ (i,j) for (i,j) in it.product(I,I) if I.index(i) < I.index(j) ]


    for (i,j) in E:
        wt = f"{f((I.index(i),I.index(j)))}"
        wdg.edge(f"{i}",f"{j}",wt)

    return wdg

wdg(['a','b','c','d','e','f'],lambda x: 10*x[0] + x[1])
```

[3]:



11 Example: MBTA

We can model the MBTA by viewing the stops as vertices, and the lines between stops as edges. In the following, the weights on the edges give the capacities of the individual lines.

Here we have created an *undirected graph* (we assume that the capacity between two stops doesn't depend on the direction traveled).

[5]:
```
import itertools as it

stops = [ "Park St",
          "Central Square",
          "Hynes",
          "Mass Ave",
          "Govt Center",
          "Downtown Crossing",
          "South Station",
          "State St",
          "HayMarket"
         ]
```

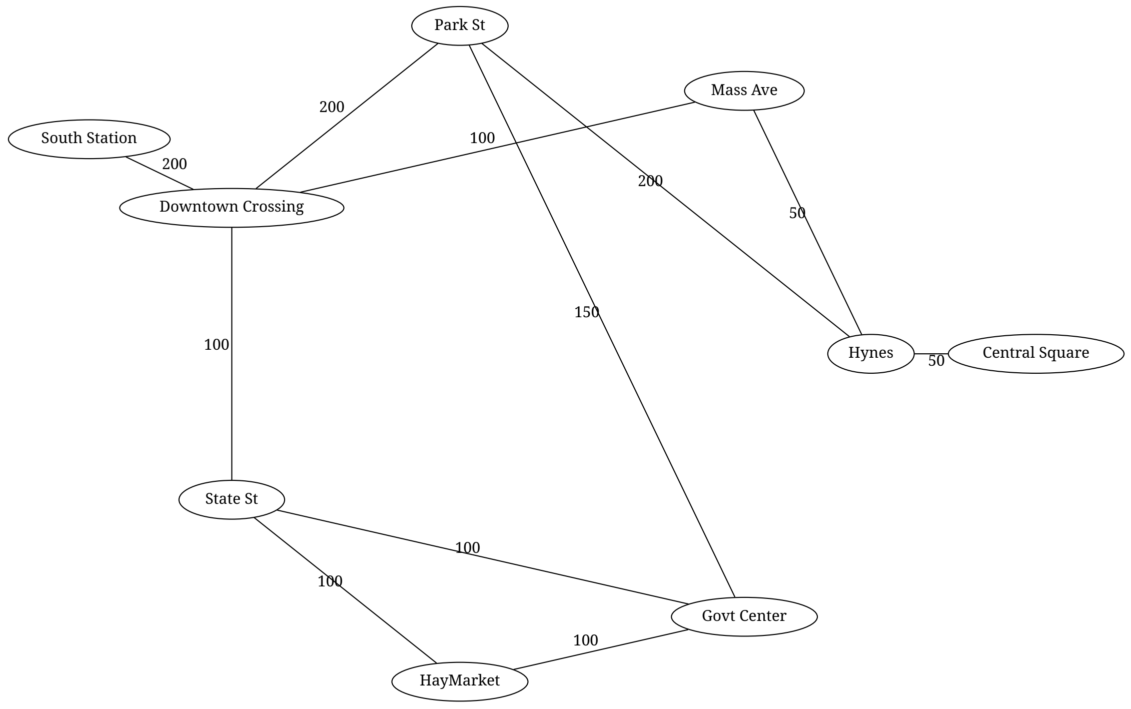
[8]:
```
# the edges are given as pairs, but should be interpreted
# as "unordered pairs" here.

# we use a dictionary here;
# the "edges" -- i.e. pairs of vertices -- are the keys.
# For a given edge, and the corresponding value
# is the "weight" attached to the edge (which here is the *capacity* of the␣
 ↪line).

connections = { ("Central Square","Hynes")             :  50,
                ("Hynes","Park St")                    : 200,
                ("Hynes","Park St")                    : 200,
                ("Hynes","Mass Ave")                   :  50,
                ("Mass Ave","Downtown Crossing")       : 100,
                ("Park St","Govt Center")              : 150,
                ("Park St","Downtown Crossing")        : 200,
                ("Govt Center","State St")             : 100,
                ("Govt Center","HayMarket")            : 100,
                ("Downtown Crossing","State St")       : 100,
                ("Downtown Crossing","South Station")  : 200,
                ("State St","HayMarket")               : 100
              }
```

[58]:
```
# since we make a 'Graph' instead of a 'DiGraph'
# the edges are unordered

mbta = Graph('MBTA-stops',engine='circo')
```

```
#mbta = Graph('MBTA-stops')


for stop in stops:
    # make a node for each stop
    mbta.node(stop)

for f,t in connections.keys():
    # make a weighted edge for each connection
    wt = f"{connections[(f,t)]}"
    mbta.edge(f,t,wt)
```

[59]:
```
mbta.format='png'

mbta
#mbta.render()
```

[59]:



12 Maximum Flow

12.1 Definitions:

Let G be a weighted directed grap. An edge $e \in E$ is given by an ordered pair $e = (v, w)$ of vertices; we'll often write $e = (v \to w)$ for this edge.

As before, we say that a node v is a *source node* if the only edges in E involving v have the form

$e = (v \to w)$ – i.e. "no edges enter v"–, and we say that v is a *terminal node* if the only edges in E involving v have the form $e = (w \to v)$ – i.e. "no edges exit v".

We **always assume** that G has exactly one source node s and exactly one terminal node t.

12.2 Remark: how to make a directed graph from an undirected graph

Suppose that G is an undirected weighted graph with nodes V, edges E, and weight function f.

Also, suppose we have two chosen nodes $s \ne t$ in V.

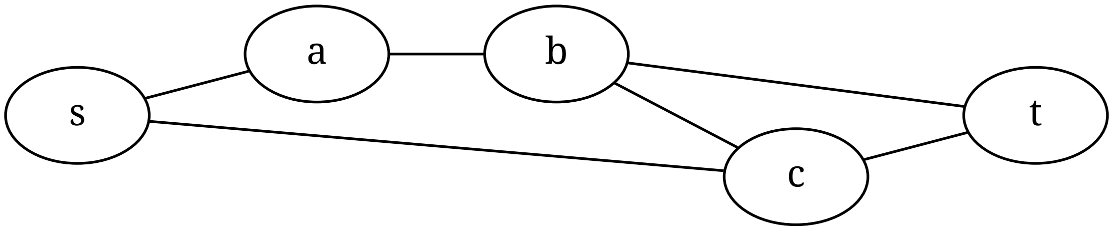
We consider the weighted directed graph $\widetilde{G}$ obtained from G by the following rules: - $\widetilde{G}$ has the same set of nodes V as does G - s is a source in $\widetilde{G}$, and we convert all edges $[s,v] = [v,s]$ in G to directed edges $(s \to v)$ in $\tilde{G}$. The weight $c(s \to v)$ is equal to $c([v,s])$. - t is a terminal node in $\widetilde{G}$, and we convert all edges $[t,v] = [v,t]$ in G to directed edges $(v \to t)$ in $\widetilde{G}$. The weight $c(v \to t)$ is equal to $c([v,t])$. - for any $[v,w]$ with $v, w \notin \{s,t\}$, there are edges in $\widetilde{G}$ in each direction: $(v \to w), (w \to v) \in \tilde{E}$. The weights $c(v \to w)$ and $c(w \to v)$ are both equal to $c([v,w])$.

This construction permits us to use some results about directed graphs to answer questions about undirected graphs, as we shall see below.

Example:

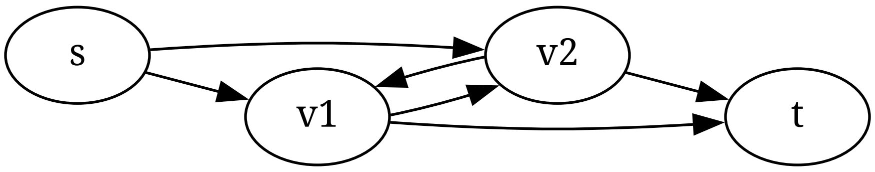
[14]:
```
g = Graph("undirected")

vertices = [ 's', 'a', 'b', 'c', 't' ]
edges = [('s','a'),
         ('s','c'),
         ('a','b'),
         ('b','c'),
         ('b','t'),
         ('c','t')
        ]

g.attr(rankdir='LR')

for v in vertices:
    g.node(v)

for (v,w) in edges:
    g.edge(v,w)

g
```

[14]:



[55]:
```
dg = Digraph("directed")
dg.attr(rankdir='LR')

for v in vertices:
    dg.node(v)

for (v,w) in edges:
    match (v,w):
        case ('s',w):
            dg.edge('s',w)
        case (v,'s'):
            dg.edge('s',v)
        case ('t',w):
            dg.edge(w,'t')
        case (v,'t'):
            dg.edge(v,'t')
        case _:
            dg.edge(v,w)
            dg.edge(w,v)

dg
```

[55]:



[16]:
```
{ 'a', 't' } - {'t'}
```

[16]:
```
{'a'}
```

13 Flows in directed graphs

As before, let G be a weighted, directed graph.

A *flow* from s to t is a function, $f : E \to \mathbb{R}_{>0}$ that satisfies conditions **(A)** and **(B)**:

(A). $f(e) \le w(e)$ for each edge $e \in E$ in the graph G, where $w(e)$ is the value of the weight function (the *capacity*) of the edge e.

> If $e = (u \to v)$ we write $f(e) = f(u \to v) = f(u, v)$ to cut down on the number of parenthesis!

(B). *conservation law* holds for each node $v \in V$ with $v \notin \{s,t\}$:

$$\sum_{u \text{ such that } (u \to v) \text{ is an edge}} f(u \to v) = \sum_{u: \text{ such that } (v \to u) \text{ is an edge}} f(v \to u)$$

Note that the RHS is the sum of the values of f for edges coming "into v", and the LHS is the sum of the values of f for edges going "out of v".

14 Flow value

The *value of the flow f* is given by

$$|f| = \sum_{u \in V \text{ such that } (s \to u) \text{ is an edge}} f(s \to u)$$

where s is the "source" node.

14.1 Lemma:

Let G be a directed graph with source s and terminus t. Then the value $|f|$ of a flow on G can also be expressed using t rather than s, as follows:

$$|f| = \sum_{u \in V \text{ such that } (u \to t) \text{ is an edge}} f(u \to t).$$

14.2 Proof:

The following two expressions coincide, since each represents the sum $\sum_{e \in E} f(e)$:

$$\sum_{v \in V} \sum_{u \text{ such that } (u \to v) \in E} f(u \to v) = \sum_{v \in V} \sum_{u \text{ such that } (v \to u) \in E} f(v \to u).$$

Subtracting the RHS from the LHS we find that

$$0 = \sum_{v \in V} \left(\sum_{u \text{ such that } (u \to v) \in E} f(u \to v) - \sum_{u \text{ such that } (v \to u) \in E} f(v \to u) \right).$$

For any $v \in V$ with $v \neq s, t$, the terms in the above sum corresponding to v sum to zero by the conservation law for flows. When $v = s$, there no edges of the form $(u \to s)$ and when $v = t$ there are no edges of the form $(t \to u)$.

We thus find that

$$0 = (-1) \cdot \sum_{u \text{ such that } (s \to u)} f(s \to u) + \sum_{u \text{ such that } (u \to t)} f(u \to t)$$

and the Lemma follows at once.

14.3 Flow as a linear program

We are interested in the question: how to maximize the flow from the source to the terminus of the network?

Note that f is just determined by an assignment of a real number to each edge $e = (u \to v) \in E$. Thus f amounts to a vector whose length is the number $\#E$ of edges in E.

Our goal is: `maximize` $|f|$

subject to $0 \leq f(u \to v) \leq c(u \to v)$ for each edge $u \to v \in E$

and subject to the **conservation laws**:

$$0 = \sum_{u \text{ such that } (u \to v) \in E} f(u \to v) - \sum_{u \text{ such that } (v \to u) \in E} f(v \to u) \text{ for each node } v \neq s, t.$$

We have just described a linear program; an optimal solution to this linear program is known as `max flow`.

Our goal in these notes is to consider the dual of this linear program, and to observe that information from the dual tells us something useful about the `max flow` problem.

We pause to formulate the following:

15 A remark about dual linear programs

Consider the linear program $\mathcal{L}$:

`maximize` $\mathbf{cx}$

subject to: *inequality constraint(s)* $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$

equality constraint $B\mathbf{x} = \mathbf{0}$.

Then $\mathcal{L}$ isn't in standard form, but nevertheless we'd like to describe the dual without having to make a complicated "change of variables."

Now each row $\mathbf{r}$ of the matrix B determines an equality constraint $\mathbf{rx} = 0$, which is equivalent to two inequality constraints

$$\mathbf{rx} \leq 0 \quad \text{and} \quad \mathbf{rx} \geq 0$$

If the rows of B are $\mathbf{r}_1, \ldots, \mathbf{r}_s$, and if A is an $r \times n$ matrix, then we form a new $(r + s) \times n$ matrix A_1 and vector $\mathbf{b_1} \in \mathbb{R}^{r+2s}$ as follows:

$$A_1 = \begin{bmatrix} A \\ \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_s \end{bmatrix} \quad \text{and} \quad \mathbf{b_1} = \begin{bmatrix} \mathbf{b} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

The condition $A_1\mathbf{x} \le \mathbf{b_1}$ is equivalent to the pair of conditions $A\mathbf{x} \le \mathbf{b}$ and $B\mathbf{x} \le \mathbf{0}$.

Consider a "dual vector" of the following form:

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_r & p_1 & \cdots & p_s \end{bmatrix}^T$$

where the $y_1, \dots, r_r$ are labelled by the rows of A and $p_1, \dots, p_s$ are labelled by the rows of B.

15.0.1 Lemma:

The linear program $\mathcal{L}'$ dual to $\mathcal{L}$ can be described as follows: `minimize` $\mathbf{b_1}^T \cdot \mathbf{y}$ subject to:

- $A_1^T\mathbf{y} \ge \mathbf{c}$,
- $y_i \ge 0$ for $1 \le i \le r$, and $p_i \in \mathbb{R}$ for $1 \le i \le s$.

Sketch of proof:

In fact, $\mathcal{L}$ can be put in standard form using a slightly more complicated matrix $\widetilde{A_1}$ than A_1: namely, instead of simply including each row $\mathbf{r}_i$, one should instead include two rows: one equal to $\mathbf{r}_i$ and another equal to $-\mathbf{r}_i$. And $\widetilde{\mathbf{b}_i}$ should end with $2s$ zeros.

Then the condition $\widetilde{A_1}\mathbf{x} \le \widetilde{\mathbf{b_1}}$ is equivalent to the pair of conditions $A\mathbf{x} \le \mathbf{b}$ and $B\mathbf{x} = \mathbf{0}$.

Taking transposes, the transpose of $\widetilde{A_1}$ leads to dual variables $y_1, \dots, y_r, p_{1,+}, p_{1,-}, \dots, p_{s,+}, p_{s,0}$ which satisfy $y_i \ge 0, p_{i,\pm} \ge 0$.

Now, one sets $p_i = p_{i,+} - p_{i,-}$, and a bit of thought leads to the conclusion of the lemma.

15.1 Linear program for flows

Lets write V^o for the set of all vertices *except* s, t; thus

$$V^o = V \setminus \{s, t\}$$

and $\#V^o = \#V - 2$.

Now consider the linear program:

`maximize` $|f|$

subject to:

inequality constraints: $f(u \to v) \le c(u, v)$ for $(u \to v) \in E$, and

$f(u \to v) \ge 0$ for $(u \to v) \in E$,

equality constraints:

$$\sum_{u \text{ such that } (u \to v) \in E} f(u \to v) - \sum_{u \text{ such that } (v \to u) \in E} f(v \to u) = 0 \quad \text{for} \quad v \in V^o.$$

15.2 Max flow and min cut

If $\mathcal{L}$ is the "flow" linear program determined by a directed graph, then the inequality constraint $B\mathbf{y} \geq \mathbf{e}$ for the dual linear program $\mathcal{L}'$ is given by a totally unimodular matrix B, and the vector $\mathbf{e}$ has integer entries.

According to the preceding Theorem, if $\mathcal{L}'$ has an optimal solution, it must have an optimal solution with integer coefficients.

Recall that the variable vector $\mathbf{y}$ for $\mathcal{L}'$ has entries $d(u \to v)$ for edges $(u \to v)$ in the graph, and entries $p(v)$ for nodes $v \in V^o$ i.e. nodes $v \neq t, s$.

We introduce value $p(s) = 1$ and $p(t) = 0$, and we note with these notations, the constraints of $\mathcal{L}'$ (described above) may be reformulated as follows:

$d(u \to v) - p(u) + p(v) \geq 0 \quad \text{for all} \quad (u \to v) \in E.$

$p(s) = 1$ and $p(t) = 0$

$p(u) \in \mathbf{R}$

$d(u \to v) \geq 0$

Suppose now that we have chosen an optimal solution with all $d(u \to v), p(v) \in \mathbb{Z}$.

This solution minimizes the quantity

$$(\heartsuit) \quad \sum_{u \to v} d(u \to v) c(u \to v).$$

In fact, there is an optimal integral solution p, d as above with the following properties:

$d(u \to v) \in \{0, 1\}$ for each edge $u \to v$.
$p(u) \in \{0, 1\}$ for each node $u \in V$.
$d(u \to v) = 0$ whenever $p(u) = p(v)$.
$d(u \to v) = 1$ if $p(u) = 1$ and $p(v) = 0$.

Observe that p determines a *partition* of the nodes. Indeed, we can view nodes u for which $p(u) = 0$ as being "grouped with t" and nodes for which $p(u) = 1$ as being "grouped with s".

===========================

15.3 The dual linear program

Consider the dual to the linear program just described; note that we have some equality constraints.

As discussed a above, we are led to the dual linear program with $\#E + \#V^o$ dual variables; we write these as

$d(u \to v)$ for $(u \to v) \in E$

and

$p(u)$ for $u \in V^o$

Note - $d(u \to v) \geq 0$ since these are the variables corresponding to rows of the primal inequality constraint matrix, while

- $p(u) \in \mathbb{R}$ since these are the variables corresponding to rows of the primal equality constraint matrix.

The dual objective function has entries $c(u \to v)$ for the edges, and 0 for the $u \in V^o$.

We now find that the dual program is given as follows:

$$\texttt{minimize} \sum_{(u\to v)\in E} c(u \to v)d(u \to v)$$

subject to:

$d(s \to v) + p(v) \geq 1$ for $(s \to v) \in E$, $v \neq t$,

$d(u \to v) - p(u) + p(v) \geq 0$ for all $(u \to v) \in E$, provided $\{u, v\} \cap \{s, t\} = \emptyset$,

$d(v \to t) - p(v) \geq 0$ for all $(v \to t) \in E$, $v \neq s$,

$p(u), d(v \to w) \geq 0$ for every $u \in V^o$ and every $(v \to w) \in E$.

One sees that $(\heartsuit)$ represents the sum of capacities of all edges which connect a node in the s-group with a node in the t-group. This leads to the `min-cut` description:

15.4 Theorem (`min cut`)

Consider the primal linear program "maximize the flow value" for a directed graph, and consider the dual linear program. An optimal solution to the dual program is obtained by considering all partitions of the nodes V into two connected groups – one group containing s and one containing t – and computing the sum of the capacities of all edges connecting the s-group to the t-group. The minimal such sum is the optimal value.

Using strong duality for linear programs, one sees immediately

15.5 Corollary

The maximum value of $|f|$ for a flow is found by computing the `min cut`.

15.6 Example

Let's return to the MBTA example introduced earlier. More precisely, let's find the maximum hourly flow between Park Street and Downtown Crossing.

Well, by duality it is the same to solve the `min cut` problem. Thus, we need to cut paths between a "Park street" group and a "Downtown Crossing" group.

We re-interpret the *undirected* graph as a directed graph with source "Park St" and terminus "Downtown Crossing", as discussed before.

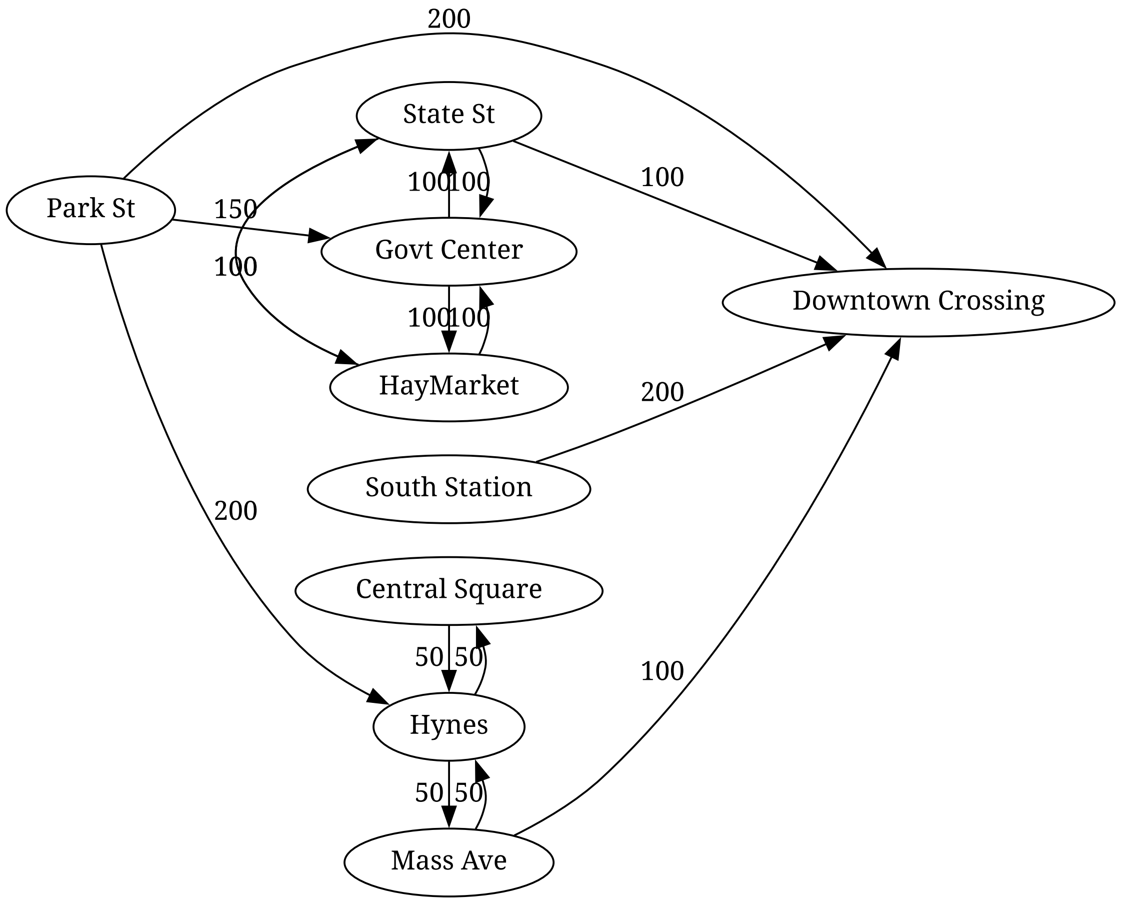
[54]:
```
a='a'

match ('c','b'):
  case ('a',_):
      print('aaa')
  case _:
      print('bbb')
```

bbb

[56]:
```
stops = [ "Park St",
          "Central Square",
          "Hynes",
          "Mass Ave",
          "Govt Center",
          "South Station",
          "State St",
          "HayMarket",
          "Downtown Crossing"
         ]

# the edges are given as pairs, but should be interpreted
# as "unordered pairs" here.
connections = { ("Central Square","Hynes")             :  50,
                ("Hynes","Park St")                    : 200,
                ("Hynes","Mass Ave")                   :  50,
                ("Mass Ave","Downtown Crossing")       : 100,
                ("Park St","Govt Center")              : 150,
                ("Park St","Downtown Crossing")        : 200,
                ("Govt Center","State St")             : 100,
                ("Govt Center","HayMarket")            : 100,
                ("Downtown Crossing","State St")       : 100,
                ("Downtown Crossing","South Station") : 200,
                ("State St","HayMarket")               : 100
               }

mbta = Digraph('MBTA-stops')
mbta.attr(rankdir='LR')

mbta.node("Park St")

source = "Park St"
term   = "Downtown Crossing"

mbta.node(source)

with mbta.subgraph() as c:
    c.attr(rank='same')

    for stop in set(stops) - set([source,term]):
        # make a node for each stop
        c.node(stop)

mbta.node(term)
```

```
for f,t in connections.keys():
    wt = f"{connections[(f,t)]}"

    # make a weighted edge for each connection
    if f == source:
        mbta.edge(f,t,wt)
    elif t == source:
        mbta.edge(t,f,wt)
    elif f == term:
        mbta.edge(t,f,wt)
    elif t == term:
        mbta.edge(f,t,wt)
    else:
        mbta.edge(f,t,wt)
        mbta.edge(t,f,wt)

mbta.format='png'

mbta
```

[56]:



In this case, the `min cut` is achieved by cutting the following connections:

- Park St to Downtown Crossing (capacity 200)
- Downtown Crossing to State (capacity 100)
- Hynes to Mass Ave (capacity 50)

This gives the partition:

Park St group:

{ Park St, Hynes, State, Central, Govt Centerr, Haymarket}

Downtown Crossing group:

{ Downtown Crossing, Mass Ave, South Station }

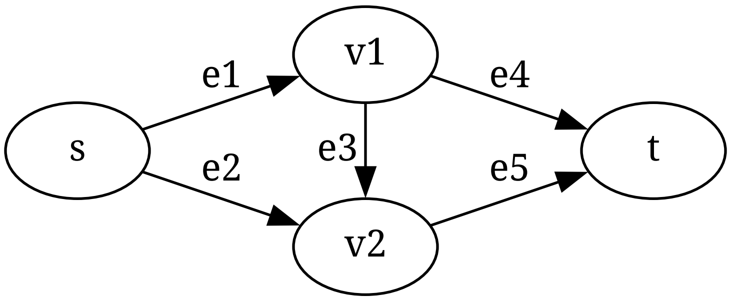
The `min cut` value is 350, and it follows by duality that the maximum flow between Park St and Downtown Crossing is 350 people per hour.

16 A small-ish example

Rather than confirm the inequalities for the dual program in general, let's see them for a simple example, since it mostly illustrates the main ideas.

[27]:
```python
simple = Digraph()
simple.attr(rankdir='LR')

extremal_vertices = [ 's', 't' ]
interior_vertices = [ 'v1', 'v2' ]

edges = { ('s' ,'v1'): 'e1',
          ('s' ,'v2'): 'e2',
          ('v1','v2'): 'e3',
          ('v1','t' ): 'e4',
          ('v2','t' ): 'e5'
        }

for v in extremal_vertices:
    simple.node(v)

with simple.subgraph() as c:
    c.attr(rank='same')
    for v in interior_vertices:
        c.node(v)

for (v,w) in edges.keys():
    simple.edge(v,w,edges[(v,w)])

simple
```

[27]:



For this graph, the `max-flow` linear program may be described as follows:

`maximize` $\mathbf{c} \cdot \mathbf{f} = f_1 + f_2$ where $\mathbf{c} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix}$.

subject to:

$f_i \leq e_i$ for $1 \leq i \leq 5$

$f_1 - f_3 - f_4 = 0$

$f_2 + f_3 - f_5 = 0$

$f_i \geq 0$ for $1 \leq i \leq 5$

In matrix form, the inequality constraints are described by:

$$A = \mathbf{I}_5, \quad b = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{bmatrix}, \quad A\mathbf{f} \leq b.$$

The equality constraints are described by $B\mathbf{f} = \mathbf{0}$ where

$$B = \begin{bmatrix} 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \end{bmatrix}$$

Now, the dual program has variables $\mathbf{y} = \begin{bmatrix} \mathbf{d} \\ \mathbf{p} \end{bmatrix}$, where $\mathbf{d} \in \mathbb{R}^5$ and $\mathbf{p} \in \mathbb{R}^2$.

The objective function is determined by $\begin{bmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & 0 & 0 \end{bmatrix}$,

and the inequality constraint is given by

$A_1^T \cdot \mathbf{y} \geq \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix}^T$

where

$$A_1 = \begin{bmatrix} & & I_5 & & \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \end{bmatrix}$$

and where I_5 is the 5×5 identity matrix.

Thus, we minimize $\sum_{i=1}^5 d_i e_i$ subject to

$d_1 + p_1 \geq 1$

$d_2 + p_2 \geq 1$

$d_3 - p_1 + p_2 \geq 0$

$d_4 - p_1 \geq 0$

$d_5 - p_2 \geq 0$

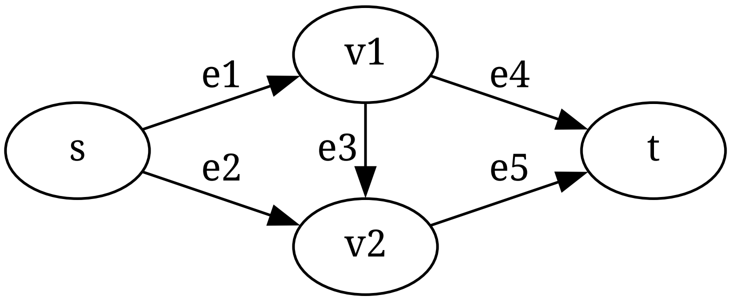
$d_i \geq 0$ all i.

We can now observe an interesting property of the solutions of the dual linear program to the flow program.

Let's compute solutions for a bunch of *randomly generated* capacities e_1, e_2, e_3, e_4, e_5.

[32]:
```
simple
```

[32]:



[35]:
```
## we can generate random "capacities" as follows--

[ np.random.rand(5) for _ in range(10) ]
```

[35]:
```
[array([0.15, 0.07, 0.97, 0.53, 0.33]),
 array([0.75, 0.05, 1.00, 0.87, 0.40]),
 array([0.33, 0.86, 0.63, 0.44, 0.18]),
 array([0.43, 0.80, 0.68, 0.25, 0.68]),
 array([0.02, 0.02, 0.53, 0.84, 0.92]),
 array([0.89, 0.85, 0.75, 0.80, 0.02]),
 array([0.78, 0.30, 0.01, 0.25, 0.68]),
 array([0.63, 0.39, 0.10, 0.05, 0.71]),
 array([0.37, 0.92, 0.13, 0.82, 0.84]),
 array([0.61, 0.58, 0.13, 0.28, 0.24])]
```

[36]:
```
import numpy as np
from scipy.optimize import linprog
```

```python
from pprint import pprint

float_formatter = "{:.2f}".format
np.set_printoptions(formatter={'float_kind':float_formatter})

def sbv(index,size):
    return np.array([1.0 if i == index else 0.0 for i in range(size)])

# describe the matrices for the primal linear program

B = np.array([[1,0,-1,-1,0],[0,1,1,0,-1]])

A1 = np.concatenate([np.identity(5),B],axis=0)

c=np.array([1,1,0,0,0])

# bounds on the dual variables.
# the first five correspond to rows of the inequality constraint matrix
# for the primal program
bounds = np.array(5*[(0,None)] +  2*[(None,None)])

def f():
    # randomly generate capacities
    b = 2*np.random.rand(5)

    e = np.concatenate([b,np.array([0,0])])

    # solve the dual linear program
    res = linprog(e,A_ub=(-1)*A1.T,b_ub=(-1)*c)
    return {'capacities': b,
            'value': res.fun,
            'result': res.x}


pprint([ f() for i in range(10) ])
```

```
[{'capacities': array([1.27, 1.96, 1.09, 0.52, 1.53]),
  'result': array([0.00, 0.00, 0.00, 1.00, 1.00, 1.00, 1.00]),
  'value': 2.0528753372855206},
 {'capacities': array([0.24, 1.39, 0.78, 0.62, 1.50]),
  'result': array([1.00, 1.00, 0.00, -0.00, -0.00, 0.00, 0.00]),
  'value': 1.6202503170337812},
 {'capacities': array([1.37, 1.09, 1.77, 0.91, 1.05]),
  'result': array([0.00, 0.00, 0.00, 1.00, 1.00, 1.00, 1.00]),
  'value': 1.9573931033311718},
 {'capacities': array([0.90, 0.98, 1.21, 1.41, 1.86]),
  'result': array([1.00, 1.00, 0.00, -0.00, -0.00, 0.00, 0.00]),
  'value': 1.8849717362468992},
{'capacities': array([0.76, 1.44, 0.55, 1.63, 1.21]),
 'result': array([1.00, 0.00, 0.00, -0.00, 1.00, 0.00, 1.00]),
 'value': 1.9755419720117828},
{'capacities': array([0.19, 0.98, 0.23, 0.13, 1.26]),
 'result': array([1.00, 1.00, 0.00, -0.00, -0.00, -0.00, 0.00]),
 'value': 1.1744028598872696},
{'capacities': array([1.14, 0.59, 0.19, 1.06, 1.33]),
 'result': array([1.00, 1.00, 0.00, -0.00, -0.00, -0.00, 0.00]),
 'value': 1.7335864251422437},
{'capacities': array([1.18, 0.04, 0.30, 0.89, 1.26]),
 'result': array([1.00, 1.00, 0.00, -0.00, -0.00, -0.00, 0.00]),
 'value': 1.2194308835921672},
{'capacities': array([0.79, 1.49, 0.14, 1.68, 1.20]),
 'result': array([1.00, 0.00, 0.00, -0.00, 1.00, 0.00, 1.00]),
 'value': 1.9933844893418846},
{'capacities': array([1.91, 1.33, 0.89, 1.13, 1.41]),
 'result': array([0.00, -0.00, 0.00, 1.00, 1.00, 1.00, 1.00]),
 'value': 2.533189388639286}]
```

We see in every case that the result – i.e. the solution to the dual problem – gives an optimal solution for which $d_i, p_j \in \{0,1\}$ for all i, j.

This is true quite generally, as we are now going to explain.

16.1 Totally unimodular matrices

Definition A square matrix A is *unimodular* if

- every entry A_{ij} is an integer, and
- $\det A = \pm 1$.

Definition The $m \times n$ matrix A is *totally unimodular* if every square submatrix of A is unimodular or is singular (has determinant 0).

Note in particular that the 1×1 square submatrices of a unimodular matrix A must be 0 or ± 1.

In particular $\begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix}$ is *unimodular* but not *totally unimodular*.

Proposition If the $n \times n$ matrix A is unimodular, and if $\mathbf{b} \in \mathbb{Z}^m$, then the equation

$$A\mathbf{x} = \mathbf{b}$$

has a solution $\mathbf{x} \in \mathbb{Z}$.

Proof Since A has integer entries, the formula for the *inverse matrix* A^{-1} shows that A^{-1} is $\dfrac{1}{\det A}B$ for some $n \times n$ matrix with integer entries; since A is unimodular, A^{-1} has integer entries.

The solution to the matrix equation is $\mathbf{x} = A^{-1}\mathbf{b}$; since A^{-1} and $\mathbf{b}$ have integer entries, also $\mathbf{x}$ has integer entries.

We are going to invoke (but not prove!) the following important result:

Theorem Consider a linear program in standard form given by $(\mathbf{c}, A, \mathbf{b})$. Suppose that $\mathbf{b} \in \mathbb{Z}^r$, that A is *totally unimodular*, and that there is at least one optimal solution. Then there is an optimal solution $\mathbf{x} \in \mathbf{Z}^n$ – i.e. an *integral* optimal solution.