CSC2411 - Linear Programming and Combinatorial Optimization*
Lecture 1: Introduction to Optimization Problems and Mathematical Programming

Notes taken by Victor Glazer

January 13, 2005

Summary: Introduction to optimization problems in general and Mathematical Programming in particular. Convex sets and functions. Convex and Linear Programming.

Optimization Problems

Definition 1.1. An *optimization problem* consists of a set $\mathcal{D}$, called the *domain*, and a real-valued function $f : \mathcal{D} \to \mathbb{R}$, called the *objective function*. $f(x) \in \mathbb{R}$ represents the "profit" or "cost" associated with $x \in \mathcal{D}$.

Optimization problems come in two flavours: minimization problems and maximization problems. In a *maximization problem*, the goal is to find an $x \in \mathcal{D}$ such that $f(y) \leq f(x)$ for all $y \in \mathcal{D}$. In other words, we want a domain element which yields the greatest profit. In a *minimization problem*, on the other hand, the goal is to find an $x \in \mathcal{D}$ such that $f(x) \leq f(y)$ for all $y \in \mathcal{D}$. In this case we want a domain element which has the smallest cost.

In general, f may fail to have an optimum in $\mathcal{D}$. However, if $\mathcal{D}$ is finite then every f has both a minimum and a maximum in $\mathcal{D}$.

Let's look at a few examples.

1. Let p be a univariate polynomial. Where does p attain its maximum value, if we restrict it to the closed interval $[0, 1] \subset \mathbb{R}$?

   This is a maximization problem. The domain is $\mathcal{D} = [0, 1]$ and the objective function f is simply p itself.

* Lecture Notes for a course given by Avner Magen, Dept. of Computer Sciecne, University of Toronto.

2. What is the largest area enclosed by a two-dimensional (closed) curve of length one?

   This is a maximization problem. The domain $\mathcal{D}$ consists of all closed two-dimensional curves of unit length, and the objective function f maps a two-dimensional curve to the area enclosed by it.

3. Recall the classical Minimum Spanning Tree (MST) problem. Suppose that we are given an undirected graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$. The weight W of a subgraph G' of G with edge set $E' \subseteq E$ is just the sum of the individual weights of its edges, $W(G') \stackrel{def}{=} \sum_{e \in E'} w(e)$. A *path* in G is a sequence of vertices $v_1, \ldots, v_n \in V$, $n \leq |V|$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq n-1$. A *cycle* is a closed path, so that $v_1 = v_n$. G is *connected* if there is a path between every $u, v \in V, u \neq v$, and *acyclic* if it doesn't contain any cycles. A *spanning tree* T of G is a connected acyclic subgraph of G with vertex set V. What is the least-weight spanning tree of G?

   This is a minimization problem. The domain is $\mathcal{T}(G) \stackrel{def}{=} \{T : T \text{ is a spanning tree of } G\}$, and the objective function is the sum of the edge weights, W. Notice that here the domain is finite, unlike in the previous two examples, since $|\mathcal{T}(G)| \leq 2^{|E|}$. This problem is *tractable*, meaning that there are polynomial-time algorithms for it (where by *polynomial time* we mean *deterministic* polynomial time *in the worst case*). The two best-known ones are due to Prim [**?**] and Kruskal [**?**], whereas Chazelle's [**?**] is currently the fastest.

4. Another classical optimization problem is the Travelling Salesman Problem (TSP). We are given an undirected graph $G = (V, E)$, whose vertices represent cities, and a pairwise distance function $d : E \to \mathbb{R}$; $d(u, v)$ is the distance between cities $u, v \in V$. A *Hamiltonian cycle* in G is a cycle in which every $v \in V$ appears exactly once. The *length* L of a cycle $C = v_1, \ldots, v_n$ is the sum of the distances between adjacent vertices, $L(C) \stackrel{def}{=} \sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_n, v_1)$. What is the shortest Hamiltonian cycle in G?

   This is a minimization problem. The domain is $\mathcal{H}(G) \stackrel{def}{=} \{H : H \text{ is a Hamiltonian cycle in } G\}$ and the objective function is the cycle length L. As with the MST problem, the domain here is finite. Unlike the MST problem, however, TSP is *intractable*, since it is NP-hard (and therefore does not have a polynomial-time algorithm unless $P = NP$).

5. A *flow network* is a directed graph $G = (V, E)$ together with a *capacity function* $c : E \to \mathbb{R}^{\geq 0}$ and two distinguished vertices $s, t \in V$; s is called the *source* and t is called the *sink*. For every $v \in V$, let $IN(v) \stackrel{def}{=} \{e \in E : e = (u, v) \text{ for some } u \in V\}$ and $OUT(v) \stackrel{def}{=} \{e \in E : e = (v, u) \text{ for some } u \in V\}$.

A *flow* in G is a function $f : E \to \mathbb{R}$ which satisfies the following three conditions:

(i) Non-negativity: for all $e \in E$, $f(e) \geq 0$

(ii) Capacity Constraints: for all $e \in E$, $f(e) \leq c(e)$

(iii) Flow conservation: for all $v \in V$, $\sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e)$

Intuitively, the capacity constraints ensure that the flow along a given edge does not exceed that edge's capacity, and the matter conservation constraints ensure that the flow entering a given vertex is equal to the flow exiting it. The *size* of a flow f is $||f|| \stackrel{def}{=} \sum_{e \in OUT(s)} f(e) - \sum_{e \in IN(s)} f(e)$. What is the largest flow in G?

This is a maximization problem. The domain is $\mathcal{F}(G) \stackrel{def}{=} \{f : f \text{ is a flow in } G\}$ and the objective function is the flow size $||\cdot||$. Although $\mathcal{F}(G)$ is infinite, the problem is tractable. One of the best-known algorithms — though not the most efficient — is due to Edmonds and Karp [**?**]; it fits into Ford and Fulkerson's *augmenting paths* framework [**?**]. The fastest algorithms currently known are of the push-relabel variety [**?**].

An interesting aspect of network flows is the *Max flow/Min cut* theorem. A *cut* in G is a partition of the vertex set V into disjoint sets S and T such that $S \cup T = V$, $s \in S$ and $t \in T$. The *capacity* C of a cut (S, T) is $C(S, T) \stackrel{def}{=} \sum_{e \in E(S,T)} c(e)$, where $E(S, T) \stackrel{def}{=} \{(u, v) \in E : u \in S, v \in T\}$. Denote the set of all cuts in G by $\mathcal{C}(G) \stackrel{def}{=} \{(S, T) : (S, T) \text{ is a cut in } G\}$. We call f a *maximum flow* in G if it has the largest size possible, so that $||f|| = \max\{||f'|| : f' \in \mathcal{F}(G)\}$, and (S, T) a *minimum cut* in G if it has the smallest size possible, so that $C(S, T) = \min\{C(S', T') : (S', T') \in \mathcal{C}(G)\}$.

Theorem 1.2. *The size of the maximum flow in G is equal to the capacity of the minimum cut in G.*

This is a special case of *Linear Programming* (LP) *duality*, a concept we will explore in greater detail later in the course.

Mathematical Programming

Although the above setting is very general, it is too abstract to be algorithmically interesting. We next consider *mathematical programming*, a more concrete special case.

Definition 1.3. A *mathematical program* consists of an objective function $f : \mathbb{R}^n \to \mathbb{R}$, a set of m *constraint functions* $\{g_i : \mathbb{R}^n \to \mathbb{R}\}_{i=1}^m$ and a *constant vector* $\vec{b} = (b_1, \ldots, b_m) \in \mathbb{R}^m$. The goal is to find an $\vec{x} \in \mathcal{D}$ which minimizes $f(\vec{x})$, where the domain $\mathcal{D} \subseteq \mathbb{R}^n$ is defined implicitly by the inequality constraints $g_i(\vec{x}) \leq b_i, 1 \leq i \leq m$, $\mathcal{D} = \bigcap_{i=1}^m \{\vec{x} \in \mathbb{R}^n : g_i(\vec{x}) \leq b_i\} = \{\vec{x} \in \mathbb{R}^n : g_i(\vec{x}) \leq b_i, 1 \leq i \leq m\}$.

We usually write such programs as follows:

$$\begin{aligned} &\min f(\vec{x}) \quad \text{subject to} \\ &g_i(\vec{x}) \leq b_i \quad \text{for } i = 1 \ldots m \end{aligned}$$

Since the domain is restricted to be some subset of n-dimensional Euclidean space, this a more limited setting. Some problems, like example 2 above for instance, cannot be cast as mathematical programs. Mathematical programming is still too general for our purposes, however, since it allows for intractable problems. We prefer to concentrate on certain special cases involving "nice" f's and g's, for various notions of "niceness".

Convex Programming

First, we'll need some definitions.

Definition 1.4. A set $S \subseteq \mathbb{R}^n$ is *convex* if $\lambda\vec{x} + (1-\lambda)\vec{y} \in S$ for all $\vec{x}, \vec{y} \in S$ and $\lambda \in [0,1]$.

Geometrically, S is convex if every line segment joining two points in S is contained in S.

We can view the point $\lambda\vec{x} + (1-\lambda)\vec{x}$ as a "weighted average" of $\vec{x}$ and $\vec{y}$, the weights being λ and $(1-\lambda)$, which are non-negative and sum to 1. Such averages are called *convex combinations*. If S is convex then every convex combination of points in S also lies in S, so that S is "closed under taking convex combinations".

Examples of convex sets: $\varnothing$, the unit n-ball $\{\vec{x} \in \mathbb{R}^n : \|\vec{x}\|_2 \leq 1\}$, any affine subspace of $\mathbb{R}^n$, the first set depicted in Figure 1.

Examples of non-convex sets: $\mathbb{R}^n \setminus \{\vec{0}\}$, the second set depicted in Figure 1.

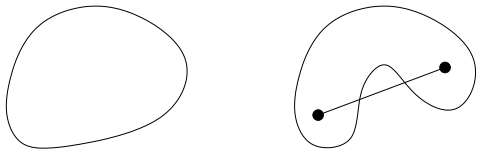

Figure 1: A convex set (left) and one that isn't (right).

Definition 1.5. Let $S \subseteq \mathbb{R}^n$ be a convex set. A function $f : S \to \mathbb{R}$ is *convex* if $f(\lambda\vec{x} + (1-\lambda)\vec{y}) \leq \lambda f(\vec{x}) + (1-\lambda) f(\vec{y})$ for every $\vec{x}, \vec{y} \in S$ and $\lambda \in [0,1]$. Notice that $\vec{z} = \lambda\vec{x} + (1-\lambda)\vec{y} \in S$ by convexity of S, so it makes sense to evaluate f on $\vec{z}$.

In other words, f is a convex function if the image under f of every convex combination of points in S is bounded above by the convex combination of their images.

Geometrically, f is convex if its *epigraph*, $epi(f) \stackrel{def}{=} \{(\vec{x}, y) \in \mathbb{R}^{n+1} : \vec{x} \in S, y \geq f(\vec{x})\}$, is a convex set. For univariate functions, this means that the line segment joining any two points on the graph of f lies on or above the graph.

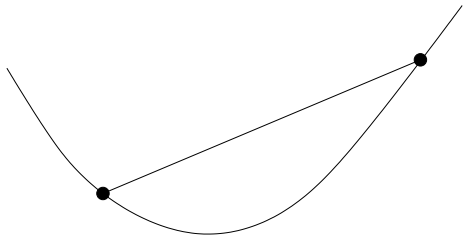

Figure 2: A convex function.

Examples of convex functions: linear functions (see below), norms (ditto), x^k for $x \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$, the function depicted in Figure 2.

Examples of functions which *aren't* convex: $f(x) = \sqrt{x}, x \in \mathbb{R}^{\geq 0}$.

To verify that $\sqrt{x}$ is *not* convex, note that

$$\sqrt{\frac{1}{2}0 + \frac{1}{2}100} = \sqrt{50} \approx 7.07 > 5 = \frac{1}{2}\sqrt{0} + \frac{1}{2}\sqrt{100}.$$

Recall that a *norm* is any function $||\cdot|| : \mathbb{R}^n \to \mathbb{R}$ which satisfies the following three conditions:

(i) $||\vec{x}|| > 0$ for all $\vec{x} \in \mathbb{R}^n \setminus \{\vec{0}\}$, and $f(\vec{0}) = 0$

(ii) $||c\vec{x}|| = |c|\ ||\vec{x}||$ for all $\vec{x} \in \mathbb{R}^n$ and $c \in \mathbb{R}$

(iii) $||\vec{x} + \vec{y}|| \leq ||\vec{x}|| + ||\vec{y}||$ for all $\vec{x}, \vec{y} \in \mathbb{R}^n$ (triangle inequality)

Claim 1.6. *Norms are convex functions.*

Proof. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a norm, $\vec{x}, \vec{y} \in \mathbb{R}^n$ and $\lambda \in [0, 1] \subseteq \mathbb{R}$. We have:

$$\begin{aligned} ||\lambda\vec{x} + (1-\lambda)\vec{y}|| &\leq ||\lambda\vec{x}|| + ||(1-\lambda)\vec{y}|| \quad \text{(by the triangle inequality)} \\ &= |\lambda|\ ||\vec{x}|| + |(1-\lambda)|\ ||\vec{y}|| \quad \text{(by property (ii) of norms)} \\ &= \lambda||\vec{x}|| + (1-\lambda)||\vec{y}|| \quad \text{(since } \lambda, 1-\lambda \geq 0\text{)}. \end{aligned}$$

■

We are now ready to define Convex Programming.

Definition 1.7. A mathematical program

$$\begin{aligned} &\min f(\vec{x}) \quad \text{subject to} \\ &g_i(\vec{x}) \leq b_i \quad \text{for } i = 1 \ldots m \end{aligned}$$

is *convex* if both the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}, 1 \leq i \leq m$ are convex.

Here the domain is a convex set, as we will show in a moment. We'll need the following two lemmas.

Lemma 1.8. *The intersection $S = \bigcap_{i=1}^{m} S_i$ of a collection of m convex sets $\{S_i\}_{i=1}^{m}$ is itself convex.*

Proof. Let $\vec{x}, \vec{y} \in S = \bigcap_{i=1}^{m} S_i$ and $\lambda \in [0,1]$. Since $\vec{z} = \lambda\vec{x} + (1-\lambda)\vec{y} \in S_i$ for all $1 \leq i \leq m$ (by convexity of the individual S_i's), $\vec{z} \in \bigcap_{i=1}^{m} S_i = S$. ■

Lemma 1.9. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a convex function and $b \in \mathbb{R}$ be a constant. Then the set $S = \{\vec{x} \in \mathbb{R}^n : g(\vec{x}) \leq b\} \subseteq \mathbb{R}^n$ is convex.*

Proof. Let $\vec{x}, \vec{y} \in S$ and $\lambda \in [0,1]$. We want to show that $\lambda\vec{x} + (1-\lambda)\vec{y} \in S$, i.e. $g(\lambda\vec{x} + (1-\lambda)\vec{y}) \leq b$. We have:

$$\begin{aligned} g(\lambda\vec{x} + (1-\lambda)\vec{y}) &\leq \lambda g(\vec{x}) + (1-\lambda)g(\vec{y}) \quad \text{(by convexity of } g) \\ &\leq \max\{g(\vec{x}), g(\vec{y})\} \quad \text{(because } \lambda + (1-\lambda) = 1) \\ &\leq b \quad \text{(since } \vec{x}, \vec{y} \in S). \end{aligned}$$

■

The following theorem is an immediate consequence of lemmas 1.8 and 1.9.

Theorem 1.10. *The domain $\mathcal{D} = \bigcap_{i=1}^{m} \mathcal{D}_i, \mathcal{D}_i = \{\vec{x} \in \mathbb{R}^n : g_i(\vec{x}) \leq b_i\}$ of a convex program*

$$\begin{aligned} &\min f(\vec{x}) \quad \textit{subject to} \\ &g_i(\vec{x}) \leq b_i \quad \textit{for } i = 1 \ldots m \end{aligned}$$

is a convex set.

Let $\mathcal{D}$ be a non-empty subset of $\mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ be a function. Recall that $\vec{x} \in \mathcal{D}$ is a *local minimum* of f in $\mathcal{D}$ if $f(\vec{x})$ is the smallest value of f in some n-ball contained in $\mathcal{D}$ centered at $\vec{x}$. More formally, there exists an $\epsilon \in \mathbb{R}^{>0}$ such that $f(\vec{x}) \leq f(\vec{y})$ for all $\vec{y} \in \mathcal{D}$ which satisfy $\|\vec{x} - \vec{y}\|_2 \leq \epsilon$ (where $||\cdot||_2$ denotes the Euclidean norm, see Example 1.12). If $f(\vec{x}) \leq f(\vec{y})$ for all $\vec{y} \in \mathcal{D}$, then $\vec{x}$ is a *global minimum* of f in $\mathcal{D}$. Local and global maxima are defined similarly.

One reason general optimization problems are so difficult is that there may be many local optima, some far from a global optimum. This means that locally optimal decisions do not necessarily result in a solution which is globally optimal, or even close to one. However, this is not an issue in Convex Programming, as the next theorem demonstrates.

Theorem 1.11. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then every local minimum $\vec{x} \in \mathcal{D}$ of f in a convex set $\mathcal{D} \subseteq \mathbb{R}^n$ is a global minimum of f in $\mathcal{D}$.*

Proof. Let $\vec{z} \in \mathcal{D}$ be an arbitrary point in the domain and $\vec{x} \in \mathcal{D}$ be a local minimum of f in $\mathcal{D}$ (so that $f(\vec{x}) \leq f(\vec{y})$ for all $\vec{y} \in \mathcal{D}$ such that $||\vec{x} - \vec{y}|| \leq \epsilon$, where $\epsilon > 0$ is some real constant). Consider any convex combination $\vec{y} = \lambda\vec{x} + (1-\lambda)\vec{z}$, $\lambda \in [0,1]$

of $\vec{x}$ and $\vec{z}$ which is "close" to $\vec{x}$, so that $||\vec{x} - \vec{y}|| \leq \epsilon$. Since $\vec{y} \in \mathcal{D}$ by convexity of $\mathcal{D}$ and $\vec{x}$ is a local minimum of f, we have

$$\begin{aligned} f(\vec{x}) \leq f(\vec{y}) &= f(\lambda\vec{x} + (1-\lambda)\vec{z}) \\ &\leq \lambda f(\vec{x}) + (1-\lambda) f(\vec{z}) \text{ (by convexity of } f) \end{aligned}$$

so that $(1-\lambda)f(\vec{x}) \leq (1-\lambda)f(\vec{z})$. Since $1-\lambda \geq 0$, this shows that $f(\vec{x}) \leq f(\vec{z})$. ■

See Figure 3 for a picture of the proof.

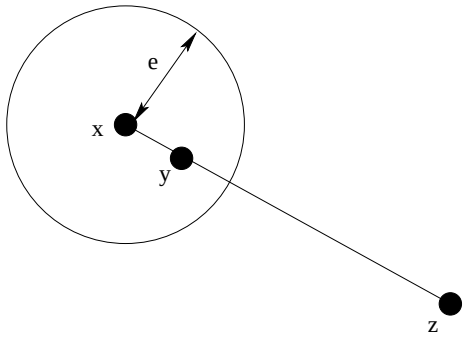

Figure 3: A graphical view of Theorem 1.11.

Example 1.12. Recall that the ℓ_2 or *Euclidean* norm of $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is $\|\vec{x}\|_2 \stackrel{def}{=} \sqrt{\sum_{j=1}^n x_j^2}$, whereas its ℓ_1 or *Manhattan* norm is $\|\vec{x}\|_1 \stackrel{def}{=} \sum_{j=1}^n |x_j|$. What vector $\vec{x} = (x_1, x_2) \in \mathbb{R}^2$ in the first quadrant has the smallest Euclidean norm, subject to the constraint that its Manhattan norm is 1? Since $\vec{x}$ lies in the first quadrant, so that $\sum_{j=1}^2 |x_j| = \sum_{j=1}^2 x_j$, we get the following program:

$$\begin{aligned} \min \sqrt{x_1^2 + x_2^2} \quad \text{subject to} \\ x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_1 + x_2 &= 1 \end{aligned}$$

However, this program doesn't quite have the right form, so we split the equality constraint and flip the $\geq$ constraints, obtaining

$$\begin{aligned} \min \sqrt{x_1^2 + x_2^2} \quad \text{subject to} \\ -x_1 &\leq 0 \\ -x_2 &\leq 0 \\ x_1 + x_2 &\leq 1 \\ -x_1 - x_2 &\leq -1 \end{aligned}$$

The objective function is convex since it's a norm (see Claim 1.6), and the constraints are convex because they're linear (see Definition 1.13).

Linear Programming

Linear Programming is an important special case of Convex Programming which has been studied extensively over the past sixty years or so.

Definition 1.13. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *linear* if

(i) $f(\vec{x} + \vec{y}) = f(\vec{x}) + f(\vec{y})$ and

(ii) $f(\alpha\vec{x}) = \alpha f(\vec{x})$

for all $\vec{x}, \vec{y} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$.

Observation 1.14. *Every linear function $f : \mathbb{R}^n \to \mathbb{R}$ can be expressed as*

$$f(\vec{x}) = \langle \vec{c}, \vec{x} \rangle = \sum_{j=1}^{n} c_j x_j,$$

where $c_j = f(\vec{e}_j)$ and $\vec{e}_j$ is the j^{th} standard basis vector (which has a 1 in the j^{th} coordinate and 0's everywhere else).

Claim 1.15. *Linear functions are convex.*

Proof. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a linear function, $\vec{x}, \vec{y} \in \mathbb{R}^n$ and $\lambda \in [0, 1]$. Then

$$\begin{aligned} f(\lambda\vec{x} + (1-\lambda)\vec{y}) &= f(\lambda\vec{x}) + f((1-\lambda)\vec{y}) && \text{(by linearity property (i))} \\ &= \lambda f(\vec{x}) + (1-\lambda) f(\vec{y}) && \text{(by linearity property (ii))} \\ &\leq \lambda f(\vec{x}) + (1-\lambda) f(\vec{y}), \end{aligned}$$

■

In other words, for linear functions the image of a convex combination of two points in the domain is actually equal to the convex combination of their images, as opposed to just being upper-bounded by it.

Definition 1.16. A convex program

$$\begin{aligned} &\min f(\vec{x}) \quad \text{subject to} \\ &g_i(\vec{x}) \leq b_i \quad \text{for } i = 1 \dots m \end{aligned}$$

is *linear* if both the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}, 1 \leq i \leq m$ are linear, and can therefore be expressed as $f(\vec{x}) = \sum_{j=1}^{n} c_j x_j$,

$g_i(\vec{x}) = \sum_{j=1}^{n} a_{ij}x_j$ (see Observation 1.14). The program then becomes

$$\begin{aligned} &\min \sum_{j=1}^{n} c_j x_j \quad \text{subject to} \\ &\sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad \text{for } i = 1 \ldots m \end{aligned}$$

Example 1.17. (the diet problem) A farmer wants his cow to be as skinny as possible while still keeping her healthy. There are n different food types available, the j^{th} food containing $c_j \in \mathbb{R}$ calories per kilogram, $1 \leq j \leq n$, and $a_{ij} \in \mathbb{R}$ milligrams of vitamin i per kilogram, $1 \leq i \leq m$. The cow requires at least $b_i \in \mathbb{R}$ milligrams of vitamin i to stay healthy. Given that the goal is to minimize caloric intake while having enough of each vitamin, how should she be fed?

Letting x_j be the number of kilograms of food j the cow is fed, we get the following linear program:

$$\begin{aligned} &\min \sum_{j=1}^{n} c_j x_j \quad \text{subject to} \\ &x_j \geq 0 \quad \text{for } j = 1 \ldots n \\ &\sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad \text{for } i = 1 \ldots m \end{aligned}$$

Example 1.18. (**network flows redux)** Recall the network flows problem from example 5. As we already pointed out, fairly efficient specialized algorithms for this problem are known. However, it can also be attacked using the general methods of Linear Programming, as we now show.

Letting x_e be the flow along edge $e \in E$, $f(e)$, we get the following linear program:

$$\begin{aligned} \max \sum_{e \in OUT(s)} x_e - \sum_{e \in IN(s)} x_e \quad &\text{subject to} \\ x_e \geq 0 \quad &\text{for } e \in E \\ x_e \leq c(e) \quad &\text{for } e \in E \\ \sum_{e \in OUT(v)} x_e = \sum_{e \in IN(v)} x_e \quad &\text{for } v \in V \setminus \{s, t\} \end{aligned}$$