Mathematics of Operational Research

Felix Fischer
fischerf@statslab.cam.ac.uk

May 18, 2015

Contents

List of Figures

1 Optimization

An *optimization problem* has the standard form

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h(x) = b \\ & x \in X. \end{aligned} \tag{1.1}$$

It consists of a vector $x \in \mathbb{R}^n$ of *decision variables*, an *objective function* $f : \mathbb{R}^n \to \mathbb{R}$, a *functional constraint* $h(x) = b$ where $h : \mathbb{R}^n \to \mathbb{R}^m$ and $b \in \mathbb{R}^m$, and a *regional constraint* $x \in X$ where $X \subseteq \mathbb{R}^n$. The set $X(b) = \{x \in X : h(x) = b\}$ is called the *feasible set*, and a problem is called *feasible* if $X(b)$ is non-empty and bounded if $f(x)$ is bounded from below on $X(b)$. A vector x^* is called optimal if it is in the feasible set and minimizes f among all vectors in the feasible set. The assumption that the functional constraint holds with equality is without loss of generality: an inequality constraint like $g(x) \leqslant b$ can be re-written as $g(x) + z = b$, where z is a new *slack variable* with the additional regional constraint $z \geqslant 0$.

1.1 Lagrangian Methods

A well-known method for solving constrained optimization problems is the method of Lagrange multipliers. The idea behind this method is to reduce constrained optimization to unconstrained optimization, and to take the (functional) constraints into account by augmenting the objective function with a weighted sum of them. To this end, define the *Lagrangian* associated with (1.1) as

$$L(x, \lambda) = f(x) - \lambda^T (h(x) - b), \tag{1.2}$$

where $\lambda \in \mathbb{R}^m$ is a vector of *Lagrange multipliers*.

The following result provides a condition under which minimizing the Lagrangian, subject only to the regional constraints, yields a solution to the original constrained problem. The result is easy to prove, yet extremely useful in practice.

THEOREM 1.1 (Lagrangian Sufficiency Theorem). *Let $x \in X$ and $\lambda \in \mathbb{R}^m$ such that $L(x, \lambda) = \inf_{x' \in X} L(x', \lambda)$ and $h(x) = b$. Then x is an optimal solution of (1.1).*

Proof. We have that

$$\begin{aligned} \min_{x' \in X(b)} f(x') &= \min_{x' \in X(b)} [f(x') - \lambda^T (h(x') - b)] \\ &\geqslant \min_{x' \in X} [f(x') - \lambda^T (h(x') - b)] \\ &= f(x) - \lambda^T (h(x) - b) = f(x). \end{aligned}$$

Equality in the first line holds because $h(x') - b = 0$ when $x' \in X(b)$. The inequality on the second line holds because the minimum is taken over a larger set. In the third line we finally use that x minimizes L and that $h(x) = b$. □

Two remarks are in order. First, a vector λ of Lagrange multipliers satisfying the conditions of the theorem is not guaranteed to exist in general, but it does exist for a large class of problems. Second, the theorem appears to be useful mainly for showing that a given solution x is optimal. In certain cases, however, it can also be used to find an optimal solution. Our general strategy in these cases will be to minimize $L(x, \lambda)$ for all values of λ, in order to obtain a minimizer $x^*(\lambda)$ that depends on λ, and then find λ^* such that $x^*(\lambda^*)$ satisfies the constraints. Let us apply this strategy to a concrete example.

EXAMPLE 1.2. Consider minimizing $x_1^2 + x_2^2$ subject to $a_1x_1 + a_2x_2 = b$ and $x_1, x_2 \geqslant 0$ for some $a_1, a_2, b \geqslant 0$. The Lagrangian is

$$L((x_1, x_2), \lambda) = x_1^2 + x_2^2 - \lambda(a_1x_1 + a_2x_2 - b),$$

and taking partial derivaties reveals that it has a unique stationary point at $(x_1, x_2) = (\lambda a_1/2, \lambda a_2/2)$. We now choose λ such that the constraint $a_1x_1 + a_2x_2 = b$ is satisfied at this point, which happens for $\lambda = 2b/(a_1^2 + a_2^2)$. Since $\partial^2 L/\partial^2 x_1^2 > 0$, $\partial^2 L/\partial^2 x_2^2 > 0$, and $\partial^2 L/(\partial x_1 \partial x_2) = 0$ for this value of λ, we have found a minimum with value $b^2/(a_1^2 + a_2^2)$ at $(x_1, x_2) = (a_1 b, a_2 b)/(a_1^2 + a_2^2)$.

More generally, to

$$\text{minimize } f(x) \text{ subject to } h(x) \leqslant b,\ x \in X, \tag{1.3}$$

we proceed as follows:

1. Introduce a vector z of slack variables to obtain the equivalent problem

$$\text{minimize } f(x) \text{ subject to } h(x) + z = b,\ x \in X,\ z \geqslant 0.$$

2. Compute the Lagrangian $L(x, z, \lambda) = f(x) - \lambda^T(h(x) + z - b)$.
3. Define the set
$$Y = \{\lambda \in \mathbb{R}^m : \inf_{x \in X, z \geqslant 0} L(x, z, \lambda) > -\infty\}.$$
4. For each $\lambda \in Y$, minimize $L(x, z, \lambda)$ subject only to the regional constraints, i.e., find $x^*(\lambda), z^*(\lambda)$ satisfying
$$L(x^*(\lambda), z^*(\lambda), \lambda) = \inf_{x \in X, z \geqslant 0} L(x, z, \lambda). \tag{1.4}$$
5. Find $\lambda^* \in Y$ such that $(x^*(\lambda^*), z^*(\lambda^*))$ is feasible, i.e., such that $x^*(\lambda^*) \in X$, $z^*(\lambda^*) \geqslant 0$, and $h(x^*(\lambda^*)) + z^*(\lambda^*) = b$. By Theorem 1.1, $x^*(\lambda^*)$ is optimal for (1.3).

1.2 The Lagrange Dual

Another useful concept that arises from the method of Lagrange multipliers is that of a dual problem. Denote by $\phi(b) = \inf_{x \in X(b)} f(x)$ the solution of (1.1), and define the (Lagrange) dual function $g : \mathbb{R}^m \to \mathbb{R}$ as the minimum value of the Lagrangian over X, i.e.,

$$g(\lambda) = \inf_{x \in X} L(x, \lambda).$$

Then, for all $\lambda \in \mathbb{R}^m$,

$$\inf_{x \in X(b)} f(x) = \inf_{x \in X(b)} L(x, \lambda) \geqslant \inf_{x \in X} L(x, \lambda) = g(\lambda), \tag{1.5}$$

i.e., the dual function provides a lower bound on the optimal value of (1.1). Since this holds for every value of λ, it is interesting to choose λ to make the lower bound as large as possible. This motivates the *dual problem* to

$$\begin{aligned} &\text{maximize} && g(\lambda) \\ &\text{subject to} && \lambda \in Y, \end{aligned}$$

where $Y = \{\lambda \in \mathbb{R}^m : g(\lambda) > -\infty\}$. In this context (1.1) is then referred to as the *primal problem*. Equation (1.5) is a proof of the *weak duality theorem*, which states that

$$\inf_{x \in X(b)} f(x) \geqslant \max_{\lambda \in Y} g(\lambda).$$

The primal problem (1.1) is said to satisfy *strong duality* if this holds with equality, i.e., if there exists λ such that

$$\phi(b) = g(\lambda).$$

If this is the case, then (1.1) can be solved using the method of Lagrangian multipliers. We can of course just try the method and see whether it works, as we did for Example 1.2. For certain important classes of optimization problems, however, it can be guaranteed that strong duality always holds.

1.3 Supporting Hyperplanes

A geometric interpretation of the dual function can be given in terms of ϕ. Fix $b \in \mathbb{R}^m$ and consider ϕ as a function of $c \in \mathbb{R}^m$. Further consider the hyperplane given by $\alpha : \mathbb{R}^m \to \mathbb{R}$ with

$$\alpha(c) = \beta - \lambda^T(b - c).$$

This hyperplane has intercept β at b and slope λ. We can now try to find $\phi(b)$ as follows:

1. For each λ, find $\beta_\lambda = \max\{\beta : \alpha(c) \leqslant \phi(c) \text{ for all } c \in \mathbb{R}^m\}$.
2. Choose λ to maximize β_λ.

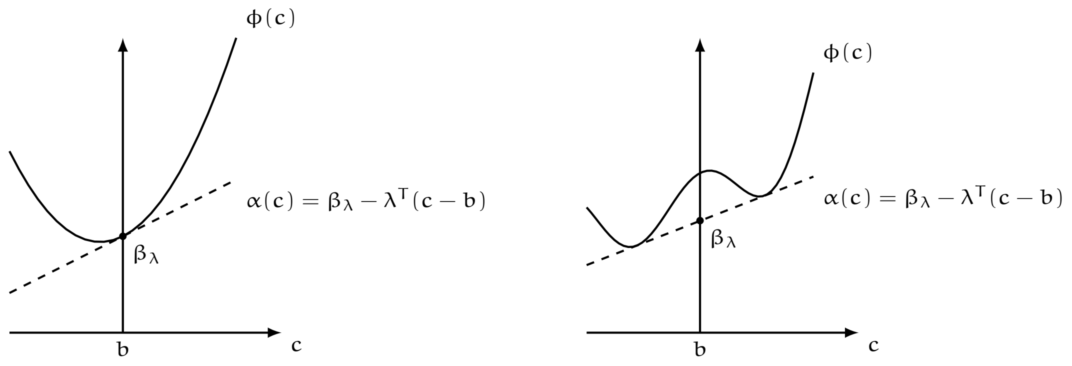

Figure 1.1: Geometric interpretation of the dual with optimal value $g(\lambda) = \beta_\lambda$. In the situation on the left strong duality holds, and $\beta_\lambda = \phi(b)$. In the situation on the right, strong duality does not hold, and $\beta_\lambda < \phi(b)$.

This approach is illustrated in Figure 1.1. We always have that $\beta_\lambda \leqslant \phi(b)$. In the situation on the left of Figure 1.1, this condition holds with equality because there is a tangent to ϕ at b. In fact,

$$\begin{aligned} g(\lambda) &= \inf_{x\in X} L(x,\lambda) \\ &= \inf_{c\in\mathbb{R}^m} \inf_{x\in X(c)} \big(f(x) - \lambda^T(h(x)-b)\big) \\ &= \inf_{c\in\mathbb{R}^m} \big(\phi(c) - \lambda^T(c-b)\big) \\ &= \sup\big\{\beta : \beta - \lambda^T(b-c) \leqslant \phi(c) \text{ for all } c \in \mathbb{R}^m\big\} \\ &= \beta_\lambda \end{aligned}$$

We again see the weak duality result as $\max_\lambda \beta_\lambda \leqslant \phi(b)$, but we also obtain a condition for strong duality. Call a hyperplane $\alpha : \mathbb{R}^m \to \mathbb{R}$ a *supporting hyperplane* to ϕ at b if $\alpha(c) = \phi(b) - \lambda^T(b-c)$ and $\phi(c) \geqslant \phi(b) - \lambda^T(b-c)$ for all $c \in \mathbb{R}^m$.

THEOREM 1.3. *The following are equivalent:*

1. *there exists a (non-vertical) supporting hyperplane to ϕ at b;*
2. *the problem satisfies strong duality.*

Proof. Suppose there exists a supporting hyperplane to ϕ at b. This means that there exists $\lambda \in \mathbb{R}^m$ such that for all $c \in \mathbb{R}^m$,

$$\phi(b) - \lambda^T(b-c) \leqslant \phi(c).$$

This implies that

$$\begin{aligned} \phi(b) &\leqslant \inf_{c\in\mathbb{R}^m} \big(\phi(c) - \lambda^T(c-b)\big) \\ &= \inf_{c\in\mathbb{R}^m} \inf_{x\in X(c)} \big(f(x) - \lambda^T(h(x)-b)\big) \\ &= \inf_{x\in X} L(x,\lambda) \\ &= g(\lambda). \end{aligned}$$

However, by (1.5) we have that $\phi(b) \geqslant g(\lambda)$. Hence $\phi(b) = g(\lambda)$, and strong duality holds.

Now suppose that the problem satisfies strong duality. Then there exists $\lambda \in \mathbb{R}^m$ such that for all $x \in X$,

$$\phi(b) \leqslant L(x, \lambda) = f(x) - \lambda^T(h(x) - b)$$

Minimizing the right hand side over $x \in X(c)$ yields that for all $c \in \mathbb{R}^m$

$$\phi(b) \leqslant \phi(c) - \lambda^T(c - b),$$

and hence

$$\phi(b) - \lambda^T(b - c) \leqslant \phi(c).$$

This describes a supporting hyperplane to ϕ at b. □

In the next lecture we will see that a supporting hyperplane exists for all $b \in \mathbb{R}^m$ if $\phi(b)$ is a convex function of b, and we will give sufficient conditions for this to be the case.

2 Convex and Linear Optimization

2.1 Convexity and Strong Duality

Let $S \subseteq \mathbb{R}^n$. S is called a *convex set* if for all $\delta \in [0,1]$, $x, y \in S$ implies that $\delta x + (1-\delta)y \in S$. A function $f : S \to \mathbb{R}$ is called *convex function* if for all $x, y \in S$ and $\delta \in [0,1]$, $\delta f(x) + (1-\delta)f(y) \geqslant f(\delta x + (1-\delta)y)$. A point $x \in S$ is called an *extreme point* of S if for all $y, z \in S$ and $\delta \in (0,1)$, $x = \delta y + (1-\delta)z$ implies that $x = y = z$. A point $x \in S$ is called an *interior point* of S if there exists $\epsilon > 0$ such that $\{y : \|y - x\|_2 \leqslant \epsilon\} \subseteq S$. The set of all interior points of S is called the *interior* of S.

We saw in the previous lecture that strong duality is equivalent to the existence of a supporting hyperplane. The following result establishes a sufficient condition for the latter.

Theorem 2.1 (Supporting Hyperplane Theorem). *Suppose that ϕ is convex and $b \in \mathbb{R}$ lies in the interior of the set of points where ϕ is finite. Then there exists a (non-vertical) supporting hyperplane to ϕ at b.*

The following result identifies a condition that guarantees convexity of ϕ.

Theorem 2.2. *Consider the optimization problem to*

$$\begin{aligned} \textit{minimize} \quad & f(x) \\ \textit{subject to} \quad & h(x) \leqslant b \\ & x \in X, \end{aligned}$$

and let ϕ be given by $\phi(b) = \inf_{x \in X(b)} f(x)$. Then, ϕ is convex when X, f, and h are convex.

Proof. Consider $b_1, b_2 \in \mathbb{R}^m$ such that $\phi(b_1)$ and $\phi(b_2)$ are defined, and let $\delta \in [0,1]$ and $b = \delta b_1 + (1-\delta)b_2$. Further consider $x_1 \in X(b_1)$, $x_2 \in X(b_2)$, and let $x = \delta x_1 + (1-\delta)x_2$. Then convexity of X implies that $x \in X$, and convexity of h that

$$\begin{aligned} h(x) &= h(\delta x_1 + (1-\delta)x_2) \\ &\leqslant \delta h(x_1) + (1-\delta)h(x_2) \\ &= \delta b_1 + (1-\delta)b_2 \\ &= b. \end{aligned}$$

Thus $x \in X(b)$, and by convexity of f,

$$\phi(b) \leqslant f(x) = f(\delta x_1 + (1-\delta)x_2) \leqslant \delta f(x_1) + (1-\delta)f(x_2).$$

This holds for all $x_1 \in X(b_1)$ and $x_2 \in X(b_2)$, so taking infima on the right hand side yields

$$\phi(b) \leqslant \delta\phi(b_1) + (1-\delta)\phi(b_2).$$ □

Observe that an equality constraint $h(x) = b$ is equivalent to constraints $h(x) \leqslant b$ and $-h(x) \leqslant -b$. In this case, the above result requires that X, f, h, and $-h$ are all convex, which in particular requires that h is linear.

2.2 Linear Programs

A *linear program* is an optimization problem in which the objective and all constraints are linear. It has the form

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & a_i^T x \geqslant b_i, & i \in M_1 \\ & a_i^T x \leqslant b_i, & i \in M_2 \\ & a_i^T x = b_i, & i \in M_3 \\ & x_j \geqslant 0, & j \in N_1 \\ & x_j \leqslant 0, & j \in N_2 \end{array}$$

where $c \in \mathbb{R}^n$ is a cost vector, $x \in \mathbb{R}^n$ is a vector of decision variables, and constraints are given by $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ for $i \in \{1, \ldots, m\}$. Index sets $M_1, M_2, M_3 \subseteq \{1, \ldots, m\}$ and $N_1, N_2 \subseteq \{1, \ldots, n\}$ are used to distinguish between different types of contraints.

An equality constraint $a_i^T x = b_i$ is equivalent to the pair of constraints $a_i^T \leqslant b_i$ and $a_i^T x \geqslant b_i$, and a constraint of the form $a_i^T x \leqslant b_i$ can be rewritten as $(-a_i)^T x \geqslant -b_i$. Each occurrence of an unconstrained variable x_j can be replaced by $x_j^+ + x_j^-$, where x_j^+ and x_j^- are two new variables with $x_j^+ \geqslant 0$ and $x_j^- \leqslant 0$. We can thus write every linear program in the *general form*

$$\min\{\, c^T x : Ax \geqslant b, x \geqslant 0 \,\} \tag{2.1}$$

where $x, c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. Observe that constraints of the form $x_j \geqslant 0$ and $x_j \leqslant 0$ are just special cases of constraints of the form $a_i^T x \geqslant b_i$, but we often choose to make them explicit.

A linear program of the form

$$\min\{\, c^T x : Ax = b, x \geqslant 0 \,\} \tag{2.2}$$

is said to be in *standard form*. The standard form is of course a special case of the general form. On the other hand, we can also bring every general form problem into the standard form by replacing each inequality constraint of the form $a_i^T x \leqslant b_i$ or $a_i^T x \geqslant b_i$ by a constraint $a_i^T x + s_i = b_i$ or $a_i^T x - s_i = b_i$, where s_i is a new so-called *slack variable*, and an additional constraint $s_i \geqslant 0$.

The general form is typically used to discuss the theory of linear programming, while the standard form is often more convenient when designing algorithms for linear programming.

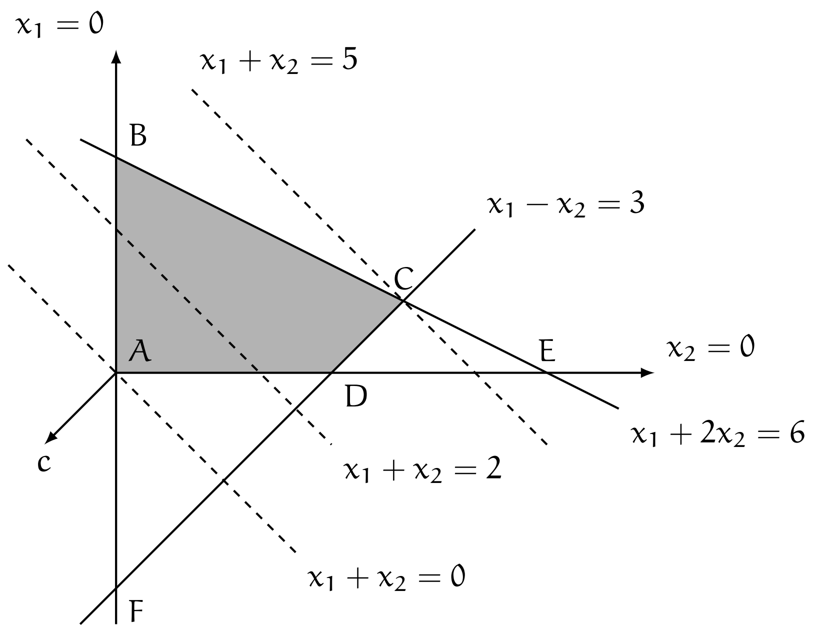

Figure 2.1: Geometric interpretation of the linear program of Example 2.3

Example 2.3. Consider the following linear program, which is illustrated in Figure 2.1:

$$\begin{array}{ll} \text{minimize} & -(x_1 + x_2) \\ \text{subject to} & x_1 + 2x_2 \leqslant 6 \\ & x_1 - x_2 \leqslant 3 \\ & x_1, x_2 \geqslant 0 \end{array}$$

Solid lines indicate sets of points for which one of the constraints is satisfied with equality. The feasible set is shaded. Dashed lines, orthogonal to the cost vector c, indicate sets of points for which the value of the objective function is constant. The optimal value over the feasible set is attained at point C.

2.3 Linear Program Duality

Consider problem (2.1) and introduce slack variables z to turn it into

$$\min\{ c^T x : Ax - z = b, x, z \geqslant 0 \}.$$

We have $X = \{(x, z) : x \geqslant 0, z \geqslant 0\} \subseteq \mathbb{R}^{m+n}$. The Lagrangian is given by

$$L((x,z),\lambda) = c^T x - \lambda^T(Ax - z - b) = (c^T - \lambda^T A)x + \lambda^T z + \lambda^T b$$

and has a finite minimum over X if and only if

$$\lambda \in Y = \{ \mu \in \mathbb{R}^m : c^T - \mu^T A \geqslant 0, \mu \geqslant 0 \}.$$

For $\lambda \in Y$, the minimum of $L((x,z),\lambda)$ is attained when both $(c^T - \lambda^T A)x = 0$ and $\lambda^T z = 0$, and thus

$$g(\lambda) = \inf_{(x,z)\in X} L((x,z),\lambda) = \lambda^T b.$$

We obtain the dual

$$\max\{\, b^T\lambda : A^T\lambda \leqslant c, \lambda \geqslant 0 \,\}. \tag{2.3}$$

The dual of (2.2) can be determined analogously as

$$\max\{\, b^T\lambda : A^T\lambda \leqslant c \,\}.$$

2.4 Complementary Slackness

An important relationship between primal and dual solutions is provided by conditions known as *complementary slackness.* Complementary slackness requires that slack does not occur simultaneously in a variable, of the primal or dual, and the corresponding constraint, of the dual or primal. Here, a variable is said to have slack if its value is non-zero, and an inequality constraint is said to have slack if it does not hold with equality. It is not hard to see that complementary slackness is a necessary condition for optimality. Indeed, if complementary slackness was violated by some variable and the corresponding contraint, reducing the value of the variable would reduce the value of the Lagrangian, contradicting optimality of the current solution. The following result formalizes this intuition.

THEOREM 2.4. *Let x and λ be feasible solutions for the primal* (2.1) *and the dual* (2.3)*, respectively. Then x and λ are optimal if and only if they satisfy complementary slackness, i.e., if*

$$(c^T - \lambda^T A)x = 0 \quad \textit{and} \quad \lambda^T(Ax - b) = 0.$$

Proof. If x and λ are optimal, then

$$\begin{aligned} c^T x &= \lambda^T b \\ &= \inf_{x'\in X} \left(c^T x' - \lambda^T(Ax' - b)\right) \\ &\leqslant c^T x - \lambda^T(Ax - b) \\ &\leqslant c^T x. \end{aligned}$$

Since the first and last term are the same, the two inequalities must hold with equality. Therefore, $\lambda^T b = c^T x - \lambda^T(Ax - b) = (c^T - \lambda^T A)x + \lambda^T b$, and thus $(c^T - \lambda^T A)x = 0$. Furthermore, $c^T x - \lambda^T(Ax - b) = c^T x$, and thus $\lambda^T(Ax - b) = 0$.

If on the other hand $(c^T - \lambda^T A)x = 0$ and $\lambda^T(Ax - b) = 0$, then

$$c^T x = c^T x - \lambda^T(Ax - b) = (c^T - \lambda^T A)x + \lambda^T b = \lambda^T b,$$

and by weak duality x and λ must be optimal. □

2.5 Shadow Prices

A more intuitive understanding of Lagrange multipliers can be obtained by again viewing (1.1) as a family of problems parameterized by $b \in \mathbb{R}^m$. As before, let $\phi(b) = \inf\{f(x) : h(x) = b, x \in \mathbb{R}^n\}$. It turns out that at the optimum, the Lagrange multipliers equal the partial derivatives of ϕ.

Theorem 2.5. *Suppose that f and h are continuously differentiable on $\mathbb{R}^n$, and that there exist unique functions $x^* : \mathbb{R}^m \to \mathbb{R}^n$ and $\lambda^* : \mathbb{R}^m \to \mathbb{R}^m$ such that for each $b \in \mathbb{R}^m$, $h(x^*(b)) = b$, $\lambda^*(b) \leqslant 0$ and $f(x^*(b)) = \phi(b) = \inf\{f(x) - \lambda^*(b)^T(h(x) - b) : x \in \mathbb{R}^n\}$. If x^* and λ^* are continuously differentiable, then*

$$\frac{\partial \phi}{\partial b_i}(b) = \lambda_i^*(b).$$

Proof. We have that

$$\begin{aligned}\phi(b) &= f(x^*(b)) - \lambda^*(b)^T(h(x^*(b)) - b)\\ &= f(x^*(b)) - \lambda^*(b)^T h(x^*(b)) + \lambda^*(b)^T b.\end{aligned}$$

Taking partial derivatives of each term,

$$\frac{\partial f(x^*(b))}{\partial b_i} = \sum_{j=1}^n \frac{\partial f}{\partial x_j}(x^*(b)) \frac{\partial x_j^*}{\partial b_i}(b),$$

$$\begin{aligned}\frac{\partial \lambda^*(b)^T h(x^*(b))}{\partial b_i} &= \lambda^*(b)^T \frac{\partial h(x^*(b))}{\partial b_i} + h(x^*(b)) \frac{\partial \lambda^*(b)^T}{\partial b_i}\\ &= \left(\sum_{j=1}^n \left(\lambda^*(b)^T \frac{\partial h}{\partial x_j}(x^*(b)) \right) \frac{\partial x_j^*}{\partial b_i}(b) \right) + h(x^*(b)) \frac{\partial \lambda^*(b)^T}{\partial b_i},\end{aligned}$$

$$\frac{\partial \lambda^*(b)^T b}{\partial b_i} = \lambda^*(b)^T \frac{\partial b}{\partial b_i} + b \frac{\lambda^*(b)^T}{\partial b_i}.$$

By summing and re-arranging,

$$\begin{aligned}\frac{\partial \phi(b)}{\partial b_i} = \sum_{j=1}^n &\left(\frac{\partial f}{\partial x_j}(x^*(b)) - \lambda^*(b)^T \frac{\partial h}{\partial x_j}(x^*(b)) \right) \frac{\partial x_j^*}{\partial b_i}(b)\\ &- (h(x^*(b)) - b) \frac{\partial \lambda^*(b)^T}{\partial b_i} + \lambda^*(b)^T \frac{\partial b}{\partial b_i}.\end{aligned}$$

The first term on the right-hand side is zero, because $x^*(b)$ minimizes $L(x, \lambda^*(b))$ and thus

$$\frac{\partial L(x^*(b), \lambda^*(b))}{\partial x_j} = \frac{\partial f}{\partial x_j}(x^*(b)) - \left(\lambda^*(b)^T \frac{\partial h}{\partial x_j}(x^*(b)) \right) = 0$$

for $j = 1, \dots, n$. The second term is zero as well, because $x^*(b)$ is feasible and thus $(h(x^*(b)) - b)_k = 0$ for $k = 1, \dots, m$, and the claim follows. □

This result continues to hold when the functional constraints are inequalities: if the ith constraint is not satisfied with equality, then $\lambda_i^* = 0$ by complementary slackness, and therefore also $\partial\lambda_i^*/\partial b_i = 0$.

In light of Theorem 2.5, Lagrange multipliers are also known as *shadow prices*, due to an economic interpretation of the problem to

$$\begin{array}{ll} \text{maximize} & f(x) \\ \text{subject to} & h(x) \leqslant b \\ & x \in X. \end{array}$$

Consider a firm that produces n different goods from m different raw materials. Vector $b \in \mathbb{R}^m$ describes the amount of each raw material available to the firm, vector $x \in \mathbb{R}^n$ the quantity produced of each good. Functions $h : \mathbb{R}^n \to \mathbb{R}^m$ and $f : \mathbb{R}^n \to \mathbb{R}$ finally describe the amounts of raw material required to produce, and the profit derived from producing, particular quantities of the goods. The goal in the above problem thus is to maximize the profit of the firm for given amounts of raw materials available to it. The *shadow price* of raw material i then is the price the firm would be willing to pay per additional unit of this raw material, which of course should be equal to the additional profit derived from it, i.e., to $\partial\phi(b)/\partial b_i$. In this context, complementary slackness corresponds to the basic economic principle that a particular raw material has a non-zero price if and only if it is scarce, in the sense that increasing its availability would increase profit.

3 The Simplex Method

3.1 Basic Solutions

In the LP of Example 2.3, the optimal solution happened to lie at an extreme point of the feasible set. This was not a coincidence. Consider an LP in general form,

$$\text{maximize } c^T x \text{ subject to } Ax \leqslant b,\ x \geqslant 0. \tag{3.1}$$

The feasible set of this LP is a convex polytope in $\mathbb{R}^n$, i.e., an intersection of half-spaces. Each level set of the objective function $c^T x$, i.e., each set $L_\alpha = \{x \in \mathbb{R}^n : c^T x = \alpha\}$ of points for which the value of the objective function is equal to some constant $\alpha \in \mathbb{R}$, is a k-dimensional flat for some $k \leqslant n$. The goal is to find the largest value of α for which $L_\alpha(f)$ intersects with the feasible set. If such a value exists, the intersection contains either a single point or an infinite number of points, and it is guaranteed to contain an extreme point of the feasible set. This fact is illustrated in Figure 3.1, and we will give a proof momentarily.

The geometric characterization of extreme points, as points that cannot be written as a convex combination of two different points, is somewhat hard to work with. We therefore use an alternative, algebraic characterization. To this end, consider the following LP in standard form, which can be obtained from (3.1) by introducing slack variables:

$$\text{maximize } c^T x \text{ subject to } Ax = b,\ x \geqslant 0, \tag{3.2}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Call a solution $x \in \mathbb{R}^n$ of the equation $Ax = b$ *basic* if at most m of its entries are non-zero, i.e., if there exists a set $B \subseteq \{1, \dots, n\}$ with $|B| = m$ such that $x_i = 0$ if $i \notin B$. The set B is then called *basis*, and variable x_i is called *basic* if $i \in B$ and *non-basic* if $i \notin B$. A basic solution x that also satisfies $x \geqslant 0$ is a *basic feasible solution* (BFS) of (3.2). We finally distinguish basic solutions that

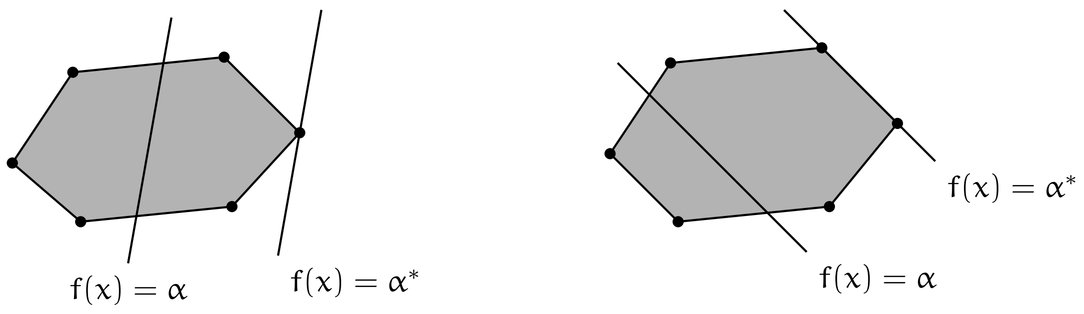

Figure 3.1: Illustration of linear programs with one optimal solution (left) and an infinite number of optimal solutions (right)

have exactly m non-zero entries from those that have strictly fewer, and refer to the latter as degenerate.

In what follows we will assume that (i) the rows of A are linearly independent and that (ii) every set of m columns of A are linearly independent. These assumptions are without loss of generality: if a set of rows are linearly dependent, one of the corresponding constraints can be removed without changing the feasible set; similarly, if a set of columns are linearly dependent, one of the corresponding variables can be removed.

3.2 Extreme Points and Optimal Solutions

It turns out that the extreme points of the feasible set are precisely the basic feasible solutions.

THEOREM 3.1. *A vector is a basic feasible solution of* $Ax = b$ *if and only if it is an extreme point of the set* $X(b) = \{x : Ax = b, x \geqslant 0\}$.

Proof. Consider a BFS x and suppose that $x = \delta y + (1 - \delta)z$ for $y, z \in X(b)$ and $\delta \in (0, 1)$. Since $y \geqslant 0$ and $z \geqslant 0$, $x = \delta y + (1 - \delta)z$ implies that $y_i = z_i = 0$ whenever $x_i = 0$. This means in particular that $y - z$ has at most m non-zero entries. At the same time, $y, z \in X(b)$ implies that $Ay = b = Az$ and thus $A(y - z) = 0$. This yields a linear combination of at most m columns of A that is equal to zero, which by (ii) implies that $y = z$. Thus x is an extreme point of $X(b)$.

Now consider a feasible solution $x \in X(b)$ that is not a BFS. Let $i_1, \dots, i_r$ be the rows of x that are non-zero, and observe that $r > m$. This means that the columns $a^{i_1}, \dots, a^{i_r}$, where $a^i = (a_{1i}, \dots, a_{mi})^T$, have to be linearly dependent, i.e., there has to exist a collection of r non-zero numbers $y_{i_1}, \dots, y_{i_r}$ such that $y_{i_1}a^{i_1} + \dots + y_{i_r}a^{i_r} = 0$. Extending y to a vector in $\mathbb{R}^n$ by setting $y_i = 0$ if $i \notin \{i_1, \dots, i_r\}$, we have $Ay = y_{i_1}a^{i_1} + \dots + y_{i_r}a^{i_r}$ and thus $A(x \pm \epsilon y) = b$ for every $\epsilon \in \mathbb{R}$. Since x_i is non-zero whenever y_i is non-zero, we can choose $\epsilon > 0$ small enough such that $x \pm \epsilon y \geqslant 0$ and thus $x \pm \epsilon y \in X(b)$. Moreover $x = 1/2(x - \epsilon y) + 1/2(x + \epsilon y)$, so x is not an extreme point of $X(b)$. □

Moreover, when looking for an optimum, we can restrict our attention to the set of basic feasible solutions.

THEOREM 3.2. *If the linear program* (3.2) *is feasible and bounded, then it has an optimal solution that is a basic feasible solution.*

Proof. Let x be an optimal solution of (3.2). If x has exactly m non-zero entries, then it is a BFS and we are done. So suppose that x has r non-zero entries for $r > n$, and that it is not an extreme point of $X(b)$, i.e., that $x = \delta y + (1 - \delta)z$ for $y, z \in X(b)$ with $y \neq y$ and $\delta \in (0, 1)$. We will show that there must exist an optimal solution with strictly fewer than r non-zero entries; the claim then follows by induction.

Since $c^T x \geqslant c^T y$ and $c^T x \geqslant c^T z$ by optimality of x, and since $c^T x = \delta c^T y + (1-\delta) c^T z$, we must have that $c^T x = c^T y = c^T z$, so y and z are optimal as well. As in the proof of Theorem 3.1, $x_i = 0$ implies that $y_i = z_i = 0$, so y and z have at most r non-zero entries, which must occur in the same rows as in x. If y or z has strictly fewer than r non-zero entries, we are done. Otherwise let $x' = \delta' y + (1-\delta')z = z + \delta'(y-z)$, and observe that x' is optimal for every $\delta' \in \mathbb{R}$. Moreover, $y - z \neq 0$, and all non-zero entries of $y - z$ occur in rows where x is non-zero as well. We can thus choose $\delta' \in \mathbb{R}$ such that $x' \geqslant 0$ and such that x' has strictly fewer than r non-zero entries. □

Since there are only finitely many basic solutions, a naive approach to solving an LP would be to go over all basic solutions and pick one that optimizes the objective. The problem with this approach is that it would not in general be efficient, as the number of basic solutions may grow exponentially in the number of variables. We will now study a well-known method for solving linear programs, the simplex method, which explores the set of basic solutions in a more organized way.

3.3 The Simplex Tableau

One way to understand the simplex method is in terms of the so-called simplex tableau, which stores all the information required to explore the set of basic solutions.

Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $x \in \mathbb{R}^n$ such that $Ax = b$. Let B be a *basis*, i.e., a set $B \subseteq \{1, \dots, n\}$ with $|B| = m$, corresponding to a choice of m non-zero variables. Then we have

$$A_B x_B + A_N x_N = b,$$

where $A_B \in \mathbb{R}^{m\times m}$ and $A_N \in \mathbb{R}^{m\times(n-m)}$ respectively consist of the columns of A indexed by B and those not indexed by B, and x_B and x_N respectively consist of the rows of x indexed by B and those not indexed by B. Moreover, if x is a basic solution, then there is a basis B such that $x_N = 0$ and $A_B x_B = b$, and if x is a basic feasible solution, there is a basis B such that $x_N = 0$, $A_B x_B = b$, and $x_B \geqslant 0$.

For every x with $Ax = b$ and every basis B, we have that $x_B = A_B^{-1}(b - A_N x_N)$, and thus

$$\begin{aligned} f(x) = c^T x &= c_B^T x_B + c_N^T x_N \\ &= c_B^T A_B^{-1}(b - A_N x_N) + c_N^T x_N \\ &= c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N \end{aligned}$$

Suppose that we want to maximize $c^T x$ and find that

$$c_N^T - c_B^T A_B^{-1} A_N \leqslant 0 \quad \text{and} \quad A_B^{-1} b \geqslant 0. \tag{3.3}$$

Then, for any feasible $x \in \mathbb{R}^n$, it holds that $x_N \geqslant 0$ and therefore $f(x) \leqslant c_B^T A_B^{-1} b$. The basic solution x^* with $x_B^* = A_B^{-1} b$ and $x_N^* = 0$, on the other hand, is feasible and satisfies $f(x^*) = c_B^T A_B^{-1} b$. It must therefore be optimal.

If alternatively $(c_N^T - c_B^T A_B^{-1} A_N)_i > 0$ for some i, then we can increase the value of the objective by increasing $(x_N)_i$. Either this can be done indefinitely, which means that the maximum is unbounded, or the constraints force some of the variables in the basis to become smaller and we have to stop when the first such variable reaches zero. In that case we have found a new BFS and can repeat the process.

Assuming that the LP is feasible and has a bounded optimal solution, there exists a basis B^* for which (3.3) is satisfied. The basic idea behind the simplex method is to start from an initial BFS and then move from basis to basis until B^* is found. The information required for this procedure can conveniently be represented by the so-called *simplex tableau*. For a given basis B, it takes the following form:[1]

$$\begin{array}{r|c|c|c|}
 & \overbrace{\qquad B \qquad}^{m} & \overbrace{\qquad N \qquad}^{n-m} & \overbrace{\qquad\qquad}^{1} \\ \hline
m\{ & A_B^{-1}A_B = I & A_B^{-1}A_N & A_B^{-1}b \\ \hline
1\{ & c_B^T - c_B^T A_B^{-1} A_B = 0 & c_N^T - c_B^T A_B^{-1} A_N & -c_B^T A_B^{-1} b \\ \hline
\end{array}$$

The first m rows consist of the matrix A and the column vector b, multiplied by the inverse of A_B. It is worth pointing out that for any basis B, the LP with constraints $A_B^{-1}Ax = A_B^{-1}b$ is equivalent to the one with constraints $Ax = b$. The first n columns of the last row are equal to $c^T - \lambda^T A$ for $\lambda^T = c_B^T A_B^{-1}$. The vector λ can be interpreted as a solution, not necessarily feasible, to the dual problem. In the last column of the last row we finally have the value $-f(x)$, where x is the BFS with $x_B = A_B^{-1}b$ and $x_N = 0$.

We will see later that the simplex method always maintains feasibility of this solution x. As a consequence it also maintains complementary slackness for x and $\lambda^T = c_B^T A_B^{-1}$: since we work with an LP in standard form, $\lambda^T(Ax - b) = 0$ follows automatically from the feasibility condition, $Ax = b$; the condition $(c^T - \lambda^T A)x = 0$ holds because $x_N = 0$ and $c_B^T - \lambda^T A_B = c_B^T - c_B^T A_B^{-1} A_B = 0$. What it then means for (3.3) to become satisfied is that $c^T - \lambda^T A \leqslant 0$, i.e., that λ is a feasible solution for the dual. Optimality of x is thus actually a consequence of Theorem 2.4.

3.4 The Simplex Method in Tableau Form

Consider a tableau of the following form, where the basis can be identified by the identity matrix embedded in (a_{ij}):

$$\begin{array}{|c|c|}
\hline
\qquad (a_{ij}) \qquad & a_{i0} \\ \hline
a_{0j} & a_{00} \\ \hline
\end{array}$$

[1]The columns of the tableau have been permuted such that those corresponding to the basis appear on the left. This has been done just for convenience: in practice we will always be able to identify the columns corresponding to the basis by the embedded identity matrix.

The simplex method then proceeds as follows:

1. Find an initial BFS with basis B.
2. Check whether $a_{0j} \leqslant 0$ for every j. If yes, the current solution is optimal, so stop.
3. Choose j such that $a_{0j} > 0$, and choose $i \in \{i' : a_{i'j} > 0\}$ to minimize a_{i0}/a_{ij}. If $a_{ij} \leqslant 0$ for all i, then the problem is unbounded, so stop. If multiple rows minimize a_{i0}/a_{ij}, the problem has a degenerate BFS.
4. Update the tableau by multiplying row i by $1/a_{ij}$ and adding a $-(a_{kj}/a_{ij})$ multiple of row i to each row $k \neq i$. Then return to Step 2.

We will now describe the different steps of the simplex method in more detail and illustrate them using the LP of Example 2.3.

Finding an initial BFS

Finding an initial BFS is very easy when the constraints are of the form $Ax \leqslant b$ for $b \geqslant 0$. We can then write the constraints as $Ax + z = b$ for a vector z of slack variables with regional constraint $z \geqslant 0$, and obtain a BFS by setting $x = 0$ and $z = b$. This can alternatively be thought of as extending x to (x, z) and setting $(x_B, x_N) = (z, x) = (b, 0)$. We then have $A_B^{-1} = I$ and $c_B = 0$, and the entries in the tableau become A_N and c_N^T for the variables x_1 and x_2 that are not in the basis, and b and 0 in the last column. For the LP of Example 2.3 we obtain the following tableau, where rows and columns have been labeled with the names of the corresponding variables:

	x_1	x_2	z_1	z_2	a_{i0}
z_1	1	2	1	0	6
z_2	1	-1	0	1	3
a_{0j}	1	1	0	0	0

If the constraints do not have this convenient form, finding an initial BFS requires more work. We will discuss this case in the next lecture.

Choosing a pivot column

If $a_{0j} \leqslant 0$ for all $j \geqslant 1$, the current solution is optimal. Otherwise we can choose a column j such that $a_{0j} > 0$ as the pivot column and let the corresponding variable enter the basis. If multiple candidate columns exist, choosing any one of them will lead to a new basis, but we could for example break ties toward the column that maximizes a_{0j} or the one with the smallest index. The candidate variables in our example are x_1 and x_2, so let us choose x_1. The pivot operation will cause this variable to enter the basis.

Choosing the pivot row

If $a_{ij} \leqslant 0$ for all i, then the problem is unbounded and the objective can be increased by an arbitrary amount. Otherwise we choose a row $i \in \{i' : a_{i'j} > 0\}$ that minimizes a_{i0}/a_{ij}. This row is called the pivot row, and a_{ij} is called the pivot. If multiple rows minimize a_{i0}/a_{ij}, the problem has a degenerate BFS. In our example there is a unique choice, corresponding to variable z_2. The pivot operation will cause this variable to leave the basis.

Pivoting

The purpose of the pivoting step is to get the tableau into the appropriate form for the new BFS. For this, we multiply row i by $1/a_{ij}$ and add a $-(a_{kj}/a_{ij})$ multiple of row i to each row $k \neq i$, including the last one. Our choice of the pivot row as a row that minimizes a_{i0}/a_{ij} turns out to be crucial, as it guarantees that the solution remains feasible after pivoting. In our example, we need to subtract the second row from both the first and the last row, after which the tableau looks as follows:

	x_1	x_2	z_1	z_2	a_{i0}
z_1	0	3	1	-1	3
x_1	1	-1	0	1	3
a_{0j}	0	2	0	-1	-3

Note that the second row now corresponds to variable x_1, which has replaced z_2 in the basis.

We are now ready to choose a new pivot column. In our example, one further iteration yields the following tableau:

	x_1	x_2	z_1	z_2	a_{i0}
x_2	0	1	$\frac{1}{3}$	$-\frac{1}{3}$	1
x_1	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
a_{0j}	0	0	$-\frac{2}{3}$	$-\frac{1}{3}$	-5

This corresponds to the BFS where $x_1 = 4$, $x_2 = 1$, and $z_1 = z_2 = 0$, with an objective of -5. All entries in the last row are non-positive, so this solution is optimal.

3.5 Degeneracies and Cycling

In the absence of degeneracies, the value of the objective function increases in every iteration of the simplex method, and an optimal solution or a certificate for unboundedness is found after a finite number of steps. When the simplex method encounters a degenerate BFS, however, it may remain at the same BFS despite changing basis. This would obviously cause the value of the objective function to remain the same as well, and the simplex method may in fact cycle indefinitely through a number of bases that all represent the same BFS.

Such cycling can be avoided by a more careful choice of pivot rows and columns, and thus of the variables entering and leaving the basis. Bland's rule achieves this by fixing some ordering of the variables and then choosing, among all variables that could enter and leave in a given iteration, those that are minimal according to the ordering.

4 Advanced Simplex Procedures

4.1 The Two-Phase Simplex Method

The LP we solved in the previous lecture allowed us to find an initial BFS very easily. In cases where such an obvious candidate for an initial BFS does not exist, we use an additional phase I to find a BFS. In phase II we then proceed as in the previous lecture.

Consider the LP to

$$\begin{aligned} \text{maximize} \quad & -6x_1 - 3x_2 \\ \text{subject to} \quad & x_1 + x_2 \geqslant 1 \\ & 2x_1 - x_2 \geqslant 1 \\ & 3x_2 \leqslant 2 \\ & x_1, x_2 \geqslant 0, \end{aligned}$$

and introduce slack variables to obtain

$$\begin{aligned} \text{maximize} \quad & -6x_1 - 3x_2 \\ \text{subject to} \quad & x_1 + x_2 - z_1 = 1 \\ & 2x_1 - x_2 - z_2 = 1 \\ & 3x_2 + z_3 = 2 \\ & x_1, x_2, z_1, z_2, z_3 \geqslant 0. \end{aligned}$$

Unfortunately, the basic solution with $x_1 = x_2 = 0$, $z_1 = z_2 = -1$, and $z_3 = 2$ is not feasible. We can, however, add an *artificial variable* to the left-hand side of each constraint where the slack variable and the right-hand side have opposite signs, and then minimize the sum of the artificial variables starting from the obvious BFS where the artificial variables are non-zero instead of the corresponding slack variables. In the example, we

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 \\ \text{subject to} \quad & x_1 + x_2 - z_1 + y_1 = 1 \\ & 2x_1 - x_2 - z_2 + y_2 = 1 \\ & 3x_2 + z_3 = 2 \\ & x_1, x_2, z_1, z_2, z_3, y_1, y_2 \geqslant 0, \end{aligned}$$

and the goal of phase I is to solve this LP starting from the BFS where $x_1 = x_2 = z_1 = z_2 = 0$, $y_1 = y_2 = 1$, and $z_3 = 2$. If the original problem is feasible, we will be able to find a BFS where $y_1 = y_2 = 0$. This automatically gives us an initial BFS for the original problem.

In summary, the two-phase simplex method proceeds as follows:

1. Bring the constraints into equality form. For each constraint in which the slack variable and the right-hand side have opposite signs, or in which there is no slack variable, add a new artificial variable that has the same sign as the right-hand side.
2. Phase I: minimize the sum of the artificial variables, starting from the BFS where the absolute value of the artificial variable for each constraint, or of the slack variable in case there is no artificial variable, is equal to that of the right-hand side.
3. If some artificial variable has a positive value in the optimal solution, the original problem is infeasible; stop.
4. Phase II: solve the original problem, starting from the BFS found in phase I.

While the original objective is not needed for phase I, it is useful to carry it along as an extra row in the tableau, because it will then be in the appropriate form at the beginning of phase II. In the example, phase I therefore starts with the following tableau:

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
y_1	1	1	−1	0	0	1	0	1
y_2	2	−1	0	−1	0	0	1	1
z_3	0	3	0	0	1	0	0	2
II	−6	−3	0	0	0	0	0	0
I	3	0	−1	−1	0	0	0	2

Note that the objective for phase I is written in terms of the variables that are not in the basis. This can be obtained by first writing it in terms of y_1 and y_2, such that we have -1 in the columns for y_1 and y_2 and 0 in all other columns because we are *maximizing* $-y_1 - y_2$, and then adding the first and second row to make the entries for all variables in the basis equal to zero.

Phase I now proceeds by pivoting on a_{21} to get

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
	0	$\frac{3}{2}$	−1	$\frac{1}{2}$	0	1	$-\frac{1}{2}$	$\frac{1}{2}$
	1	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{2}$
	0	3	0	0	1	0	0	2
II	0	−6	0	−3	0	0	3	3
I	0	$\frac{3}{2}$	−1	$\frac{1}{2}$	0	0	$-\frac{3}{2}$	$\frac{1}{2}$

and on a_{14} to get

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
	0	3	−2	1	0	2	−1	1
	1	1	−1	0	0	1	0	1
	0	3	0	0	1	0	0	2
II	0	3	−6	0	0	6	0	6
I	0	0	0	0	0	−1	−1	0

Note that we could have chosen a_{12} as the pivot element in the second step, and would have obtained the same result.

This ends phase I as $y_1 = y_2 = 0$, and we have found a BFS for the original problem with $x_1 = z_2 = 1$, $z_3 = 2$, and $x_2 = z_1 = 0$. After dropping the columns for y_1 and y_2 and the row corresponding to the objective for phase I, the tableau is in the right form for phase II:

x_1	x_2	z_1	z_2	z_3	
0	3	−2	1	0	1
1	1	−1	0	0	1
0	3	0	0	1	2
0	3	−6	0	0	6

By pivoting on a_{12} we obtain the following tableau, corresponding to an optimal solution of the original problem with $x_1 = 2/3$, $x_2 = 1/3$, and value -5:

x_1	x_2	z_1	z_2	z_3	
0	1	$-\frac{2}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$
1	0	$-\frac{1}{3}$	$-\frac{1}{3}$	0	$\frac{2}{3}$
0	0	2	−1	1	1
0	0	−4	−1	0	5

It is worth noting that the problem we have just solved is the dual of the LP in Example 2.3, which we solved in the previous lecture, augmented by the constraint $3x_2 \leqslant 2$. Ignoring the column and row corresponding to z_3, the slack variable for this new constraint, the final tableau is essentially the negative of the transpose of the final tableau we obtained in the previous lecture. This makes sense because the additional constraint is not tight in the optimal solution, as we can see from the fact that $z_3 \neq 0$.

4.2 The Dual Simplex Method

The (primal) simplex method maintains feasibility of the primal solution along with complementary slackness and seeks feasibility of the dual solution. Alternatively one could maintain feasibility of the dual solution and complementary slackness and seek feasibility of the primal solution. This is known as the dual simplex method.

One situation where the dual simplex method can be useful is when an initial feasible solution for the dual is easier to find than one for the primal. Consider the following LP, to which we have already added slack variables z_1 and z_2:

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 3x_2 + 4x_3 \\ \text{subject to} \quad & x_1 + 2x_2 + x_3 - z_1 = 3 \\ & 2x_1 - x_2 - 3x_3 - z_2 = 4 \\ & x_1, x_2, x_3, z_1, z_2 \geqslant 0. \end{aligned}$$

The primal simplex algorithm would have to use two phases, since the solution where $z_1 = -3$ and $z_2 = -4$ is not feasible. On the other hand, $c \geqslant 0$, so the dual solution with $\lambda_1 = \lambda_2 = 0$ satisfies $c^T - \lambda^T A \geqslant 0$ and is therefore feasible. We obtain the following tableau:

-1	-2	-1	1	0	-3
-2	1	3	0	1	-4
2	3	4	0	0	0

In the dual simplex algorithm the pivot is selected by picking a row i such that $a_{i0} < 0$ and a column $j \in \{j' : a_{ij'} < 0\}$ that minimizes $-a_{0j}/a_{ij}$. Pivoting then works just like in the primal algorithm. In the example we can pivot on a_{21} to obtain

0	$-\frac{5}{2}$	$-\frac{5}{2}$	1	$-\frac{1}{2}$	-1
1	$-\frac{1}{2}$	$-\frac{3}{2}$	0	$-\frac{1}{2}$	2
0	4	7	0	1	-4

and then on a_{12} to obtain

0	1	1	$-\frac{2}{5}$	$\frac{1}{5}$	$\frac{2}{5}$
1	0	-1	$-\frac{1}{5}$	$-\frac{2}{5}$	$\frac{11}{5}$
0	0	3	$\frac{8}{5}$	$\frac{1}{5}$	$-\frac{28}{5}$

We have reached the optimum of $28/5$ with $x_1 = 11/5$, $x_2 = 2/5$, and $x_3 = 0$.

It is worth pointing out that for problems in which all constraints are inequality constraints, the optimal dual solution can also be read off from the final tableau. For problems of this type, the last $n - m$ columns of the extended constraint matrix A correspond to the slack variables and therefore contain values 1 or -1 on the diagonal and 0 everywhere else. For the same reason, the last $n - m$ columns of the vector c^T are 0. The values of the dual variables, each of them with opposite sign of the slack variable in the corresponding constraint, thus appear in the last $n - m$ columns of the vector $(c^T - \lambda^T A)$ in the last row of the final tableau. In our example, we have $\lambda_1 = 8/5$ and $\lambda_2 = 1/5$.

4.3 Gomory's Cutting Plane Method

Another situation where the dual simplex method can be useful is when we need to add constraints to an already solved LP. While such constraints can make the primal solution infeasible, they do not affect feasibility of the dual solution. We can therefore simply add the constraint and continue running the dual LP algorithm from the current solution until the primal solution again becomes feasible. The need to add constraints to an LP for example arises naturally in Gomory's cutting plane approach for solving integer programs (IPs). An IP is a linear program with the additional requirement that variables should be integral.

Assume that for a given IP we have already found an optimal (fractional) solution x^* with basis B, and let a_{ij} denote the entries of the final tableau, i.e., $a_{ij} = (A_B^{-1}A_j)_i$ and $a_{i0} = (A_B^{-1}b)_i$. If x^* is not integral, there has to be a row i such that a_{i0} is not integral, and for every feasible solution x,

$$x_i + \sum_{j \in N} \lfloor a_{ij} \rfloor x_j \leqslant x_i + \sum_{j \in N} a_{ij} x_j = a_{i0}.$$

The inequality holds because x is feasible, i.e., $x \geqslant 0$, the equality follows from the properties of the final tableau. If x is integral, the left-hand side is integral as well, and the inequality must still hold if the right-hand side is rounded down. Thus,

$$x_i + \sum_{j \in N} \lfloor a_{ij} \rfloor x_j \leqslant \lfloor a_{i0} \rfloor.$$

This inequality is satisfied by every (integral) feasible solution, but not by the current solution x^*, for which $x_i^* = a_{i0}$. It corresponds to a so-called *cutting plane*, a hyperplane that separates the current solution x^* from the feasible set. The idea behind the cutting plane method is to iteratively add cutting planes and solve the resulting linear programs using the dual simplex algorithm. As it turns out, this always leads to an optimal integral solution after a finite number of steps.

Consider again the final tableau on Page 22, and assume that we are now looking for an integral solution. By the first row, and assuming that all variables are integral and non-negative,

$$x_2 + x_3 - 1z_1 + 0z_2 \leqslant x_2 + x_3 - \frac{2}{5}z_1 + \frac{1}{5}z_2 = \frac{2}{5},$$

and in fact

$$x_2 + x_3 - z_1 \leqslant 0.$$

If we turn this into an equality constraint using a new slack variable, add it to the tableau, and bring it into the right form by subtracting the first constraint from it, we obtain

0	1	1	$-\frac{2}{5}$	$\frac{1}{5}$	0	$\frac{2}{5}$
1	0	-1	$-\frac{1}{5}$	$-\frac{2}{5}$	0	$\frac{11}{5}$
0	0	0	$-\frac{3}{5}$	$-\frac{1}{5}$	1	$-\frac{2}{5}$
0	0	3	$\frac{8}{5}$	$\frac{1}{5}$	0	$-\frac{28}{5}$

After one more round of the dual simplex algorithm we reach the optimal integral solution with $x_1 = 3$ and $x_2 = x_3 = 0$:

0	1	1	−1	0	1	0
1	0	−1	1	0	−2	3
0	0	0	3	1	−5	2
0	0	3	1	0	1	−6

We will return to IPs, and learn about a different method for solving them that often works better in practice, in a later lecture.

5 Complexity of Problems and Algorithms

We have seen that the simplex algorithm inspects basic feasible solutions and is guaranteed to find an optimal solution after a finite number of steps. We have also observed, however, that the number of basic feasible solutions is generally exponential in n, and going over all of them would take a long time. It is therefore an interesting question whether there really are cases where the simplex algorithm has to look at a significant fraction of the set of all basic feasible solutions. If this was the case, we could then ask whether a similar property holds for every algorithm that solves the linear programming problem.

5.1 Asymptotic Complexity

Formally, an instance of an optimization problem is given by its input. In the case of linear programming, for example, this input consists of two vectors $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m \times n}$. If each real value is represented using at most k bits, the whole instance can be described by a string of $(mn + m + n)k$ bits. We will refer to this parameter as the *input size*.

A sensible way to define the complexity of a problem is via the complexity of the fastest algorithm that solves it. The latter is typically measured in terms of the number of arithmetic or bit-level operations as a function of the input size, ignoring lower-order terms resulting from details of the implementation. The following notation is useful in this context: given two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, write

- $f(n) = O(g(n))$ if there exist constants c and n_0 such that for every $n \geqslant n_0$, $f(n) \leqslant cg(n)$,
- $f(n) = \Omega(g(n))$ if there exist constants c and n_0 such that for every $n \geqslant n_0$, $f(n) \geqslant cg(n)$, and
- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

In other words, $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ mean that the asymptotic growth of $f(n)$ is respectively bounded from above or below by $g(n)$, up to a constant factor. Gaussian elimination for example shows that solving a linear system $Ax = b$ with $A \in \mathbb{R}^{n \times n}$ has arithmetic complexity $O(n^3)$. The same bound can also be shown to hold for bit complexity.

5.2 P, NP, and Polynomial-Time Reductions

In computational complexity theory, *efficient computation* is typically associated with running times that are at most *polynomial* in the size of the input. In many situations of interest, and also in this course, it suffices to study complexity-theoretic questions for *decisions problems*, i.e., problems where the answer is just a single bit. An example of a decision problem in the context of linear programming would be the following: given a linear program and a number $k \in \mathbb{R}$, does the optimal solution of the linear program have value less than k? Formally, a decision problem can be described by a language $L \subseteq \{0,1\}^*$, containing precisely the instances for which the answer is 1 in some encoding as strings of bits.

One might expect the answer to the question whether a particular problem can be solved efficiently to depend a lot on the details of the computational model one is using. Quite surprisingly, this turns out not to be the case: all computational models that are known to be physically realizable can simulate each other, and a particular model, the *Turing machine*, can simulate all others with polynomial overhead. A Turing machine has a finite number of states, finite control, and a readable and writable tape that can store intermediary results as strings of bits. The Turing machine is started with the input written on the tape. It then runs for a certain number of steps, and when it halts the output is inferred from the state or the contents of some designated part of the tape. In the context of decision problems, a Turing machine is said to *accept* input $x \in \{0,1\}^*$ if it halts with output 1.

The most important open problem in complexity theory is concerned with the relationship between the complexity classes P and NP. P is the class of decision problems that can be solved in *polynomial time*. Formally, a function $f : \{0,1\}^* \to \{0,1\}^*$ is computable in polynomial time if there exists a Turing machine M and $k \in \mathbb{N}$ with the following property: for every $x \in \{0,1\}^*$, if M is started with input x, then after $O(|x|^k)$ steps it halts with output $f(x)$. NP is the class of decision problems for which a given solution can be verified in polynomial time. Formally, $L \subseteq \{0,1\}^*$ is in NP if there exists a Turing machine M and $k \in \mathbb{N}$ with the following property: for every $x \in \{0,1\}^*$, $x \in L$ if and only if there exists a certificate $y \in \{0,1\}^*$ with $|y| = O(|x|^k)$ such that M accepts (x, y) after $O(|x|^k)$ steps. The name NP, for *nondeterministic polynomial time*, derives from an alternative definition as the class of decision problems solvable in polynomial time by a nondeterministic Turing machine. A nondeterministic Turing machine is a Turing machine that can make a non-deterministic choice at each step of its computation and is required to accept $x \in L$ only for some sequence of these choices. Finding a solution is obviously at least as hard as verifying a solution described by a certificate. Most people believe that it must be strictly harder, i.e., that $P \neq NP$.

The relative complexity of different decision problems can be captured in terms of *reductions*. Intuitively, a reduction from one problem to another transforms every instance of the former into an equivalent instance of the latter, where equivalence means that both of them yield the same decision. For this transformation to preserve the complexity of the original problem, the reduction should of course have less power than is required to actually solve the original problem. In our case it makes sense to use reductions that can be computed in polynomial time. A decision problem $L \subseteq \{0,1\}^*$ is called *polynomial-time reducible* to a decision problem $K \subseteq \{0,1\}^*$, denoted $L \leqslant_p K$,

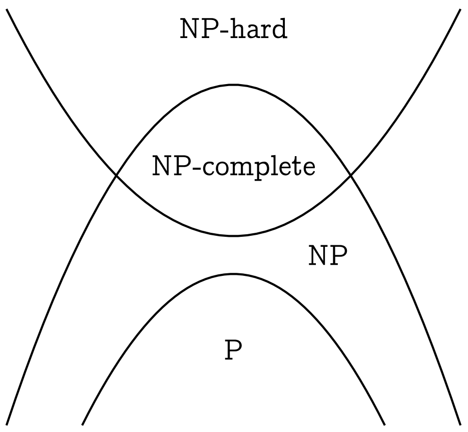

Figure 5.1: Relationship between P, NP, and the sets of NP-hard and NP-complete problems. It is not known whether the intersection between P and the set of NP-complete problems is empty. If it is not, then P = NP.

if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ computable in polynomial time such that for every $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in K$. A problem K is called *NP-hard* if for every problem L in NP, $L \leqslant_p K$. A problem is called *NP-complete* if it is both in NP and NP-hard. The relation $\leqslant_p$ is transitive. NP-complete problems are thus the hardest problems in NP, in the sense that membership of any NP-complete problem in P would imply that P = NP. The existence of NP-complete problems is less obvious, but holds nonetheless. Figure 5.1 illustrates the relationship between P and NP.

What is nice about the asymptotic worst-case notions of complexity considered above is that they do not require any assumptions about low-level details of the implementation or about the type of instances we will encounter in practice. We do, however, have to be a bit careful in interpreting results that use these notions. The fact that a problem is in P does not automatically mean that it can always be solved efficiently in practice, as the constant overhead hidden in the asymptotic notation might be prohibitively large. In fact, it does not even have to be the case that an algorithm with a polynomial worst-case running time is better in practice than an algorithm whose worst-case running time is exponential. Experience has shown, however, that for problems in P one is usually able to find algorithms that are fast in practice. On the other hand, NP-hardness of a problem does not mean that it can never be solved in practice, and we will consider approaches for solving NP-hard optimization problems in a later lecture. NP-hardness is still a very useful concept because it can help to direct efforts away from algorithms that are always efficient and toward algorithms with good practical performance.

5.3 Some NP-Complete Problems

The first problem ever shown to be NP-complete is the *Boolean satisfiability problem* (SAT), which asks whether a given Boolean formula is satisfiable. A Boolean formula consists of a set of clauses $C_i \subseteq X$ for $i = 1, \dots, m$, where $X = \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ is a set of literals. It is called satisfiable if there exists a set $S \subseteq X$ such that $|S \cap \{x_j, \bar{x}_j\}| \leqslant 1$ for all $j = 1, \dots, n$ and $|S \cap C_i| \geqslant 1$ for all $i = 1, \dots, m$. Since the set S can serve as a certificate, it is easy to see that SAT is in NP. NP-hardness can be shown by encoding the operation of an arbitrary nondeterministic Turing machine as a Boolean formula.

THEOREM 5.1 (Cook, 1971; Levin, 1973). *Boolean satisfiability is NP-complete.*

An instance of the 0—1 *integer programming problem* consists of a matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$, and asks whether there exists a vector $x \in \{0, 1\}^n$ such that $Ax \geqslant b$. Note that this is the feasibility problem associated with a special case of the integer programs we encountered in the previous lecture, in the context of the cutting plane method.

THEOREM 5.2 (Karp, 1972). *0—1 integer programming is NP complete.*

Proof. Membership in NP is again easy to see. NP-hardness can be shown by a reduction from SAT. Consider a Boolean formula with literals $X = \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ and clauses C_i, $i = 1, \dots, m$, and assume without loss of generality that $|C_i \cap \{x_j, \bar{x}_j\}| \leqslant 1$ for all $i = 1, \dots, m$ and $j = 1, \dots, n$. Now let $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$ be given by

$$a_{ij} = \begin{cases} 1 & \text{if } x_j \in C_i \\ -1 & \text{if } \bar{x}_j \in C_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } i = 1, \dots, m \text{ and } j = 1, \dots, n,$$

$$b_i = 1 - |\{ j : \bar{x}_j \in C_i\}| \qquad \text{for } i = 1, \dots, m.$$

Intuitively, this integer program represents each Boolean variable by a binary variable, and each clause by a constraint that requires its literals to sum up to at least 1. To this end, the left hand side of the contraint contains x_j if the corresponding Boolean variable occurs as a positive literal in the clause, and $(1 - x_j)$ if it occurs as a negative literal. The above form is then obtained by moving all constants to the right hand side. It is now easy to see that there exists $x \in \{0, 1\}^n$ such that $Ax \geqslant b$ if and only if the Boolean formula is satisfiable. □

The last problem we consider is the *traveling salesman problem* (TSP). For a given matrix $A \in \mathbb{N}^{n \times n}$ and a number $k \in \mathbb{N}$, it asks whether there exists a permutation $\sigma \in S_n$ such that $a_{\sigma(n)\sigma(1)} + \sum_{i=1}^{n-1} a_{\sigma(i)\sigma(i+1)} \leqslant k$. If the entries of the matrix A are interpreted as pairwise distances among a set of locations, we are looking for a tour with a given maximum length that visits every location exactly once and returns to the starting point. The special case where A is a symmetric binary matrix and $k = 0$ is also known as the Hamiltonian cycle problem.

THEOREM 5.3 (Karp, 1972). *TSP is NP-complete, even if $A \in \{0, 1\}^{n \times n}$ symmetric and $k = 0$.*

6 The Complexity of Linear Programming

6.1 A Lower Bound for the Simplex Method

The complexity of the simplex method depends on two factors, the number of steps in each round and the number of iterations. It is not hard to see that the tableau form requires $O(mn)$ arithmetic operations in each round. We will now describe an instance of the linear programming problem, and a specific pivot rule, such that the simplex method requires an exponential number of iterations to find the optimal solution. For this, we construct a polytope with an exponential number of vertices, and a so-called *spanning path* that traverses all of the vertices, in such a way that consecutive vertices are adjacent and a certain linear objective strictly increases along the path. This shows that the simplex method requires an exponential number of iterations in the worst case, for the specific pivoting rule that follows the spanning path.

Consider the unit cube in $\mathbb{R}^n$, given by the constraints

$$0 \leqslant x_i \leqslant 1 \quad \text{for } i = 1, \dots, n.$$

The unit cube has 2^n vertices, because either one of the two constraints $0 \leqslant x_i$ and $x_i \leqslant 1$ can be active for each dimension i. Further consider a spanning path of the unit cube constructed inductively as follows. In dimension 1, the path moves from $x_1 = 0$ to $x_1 = 1$. In dimension k, the path starts with $x_k = 0$ and traverses the spanning path for dimensions x_1 to x_{k-1}, which exists by the induction hypothesis. It then moves to the adjacent vertex with $x_1 = 1$, and traverses the spanning path for dimensions x_1 to x_{k-1} in the reverse direction. This construction is illustrated of the left of Figure 6.1.

Now assume that we are trying to minimize the objective $-x_n$, and observe that so far it decreases only once, namely in the middle of the path. This can easily be fixed. Let $\epsilon \in (0, 1/2)$, and consider the perturbed unit cube with constraints

$$\begin{aligned} \epsilon \leqslant x_1 &\leqslant 1, \\ \epsilon x_{i-1} \leqslant x_i &\leqslant 1 - \epsilon x_{i-1} \quad \text{for } i = 2, \dots, n \end{aligned} \tag{6.1}$$

An example is shown on the right of Figure 6.1. It is easily verified that x_n now increases strictly along the path described above. We obtain the following result.

THEOREM 6.1. *Consider the linear programming problem of minimizing $-x_n$ subject to* (6.1). *Then there exists a pivoting rule and an initial basic feasible solution such that the simplex method requires $2^n - 1$ iterations before it terminates.*

Observe that each of the numbers in the description of the perturbed unit cube can be represented using $O(\log \epsilon^{-n}) = O(n)$ bits, the number of iterations is therefore also exponential in the input size.

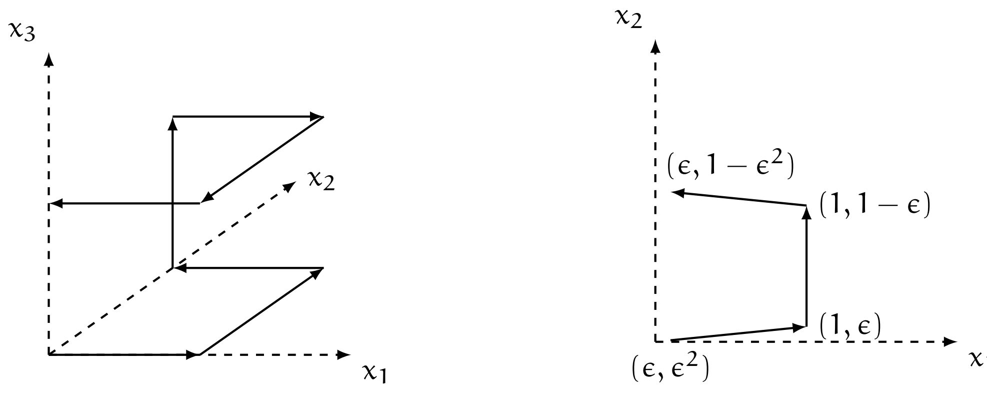

Figure 6.1: Spanning paths of the three-dimensional unit cube (left) and of the perturbed two-dimensional unit cube with $\epsilon = 1/10$ (right)

Interestingly, the first and last vertices of the spanning paths constructed above are adjacent, which means that a different pivoting rule could reach the optimal solution in a single step. However, similar worst-case instances have been constructed for many other pivot rules, and no pivot rule is known to guarantee a polynomial worst-case running time. The *diameter* of a polytope, i.e., the maximum number of steps necessary to get from any vertex to any other vertex, provides a lower bound of the number of iterations of the simplex method that is independent of the pivoting rule. The Hirsch conjecture, which states that the diameter of a polytope in dimension d with n facets cannot be greater than $n-d$, was disproved in 2010. Whether the diameter is bounded by a polynomial function of n and d remains open.

In practice, the performance of the simplex method is often much better, usually linear in the number of constraints. However, it is not clear how the intuition of a good average-case performance could be formalized, because this would require a natural probability distribution over instances of the linear programing problem. This is a problem that applies more generally to the average-case analysis of algorithms.

6.2 The Idea for a New Method

Again consider the linear program (2.2) and its corresponding dual:

$$\min\{\, c^T x : Ax = b, x \geqslant 0 \,\}$$
$$\max\{\, b^T \lambda : A^T \lambda \leqslant c \,\}.$$

By strong duality, each of these problems has a bounded optimal solution if and only if the following set of linear constraints is feasible:

$$c^T x = b^T \lambda, \quad Ax = b, \quad x \geqslant 0, \quad A^T \lambda \leqslant c.$$

We can thus concentrate on the following decision problem: given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, is the set $\{x \in \mathbb{R}^n : Ax \geqslant b\}$ non-empty? We will now consider a method for solving this problem, known as the *ellipsoid method*.

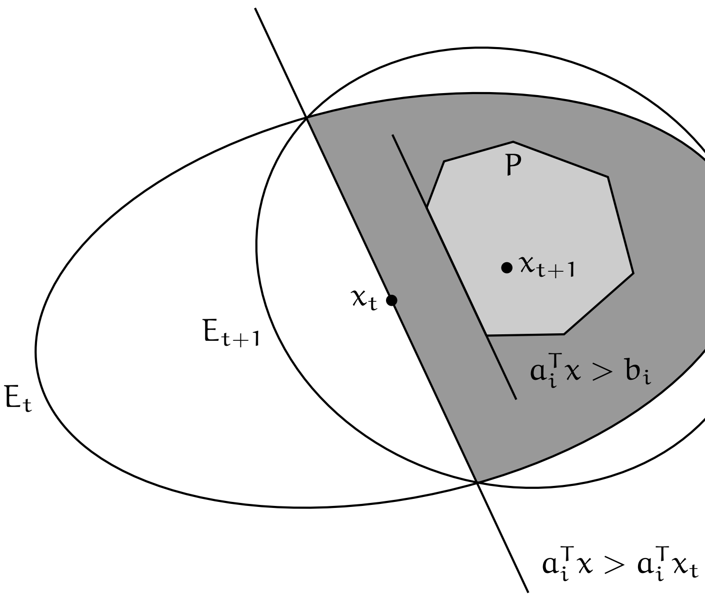

Figure 6.2: A step of the ellipsoid method where $x_t \notin P$ but $x_{t+1} \in P$. The polytope P and the half-ellipsoid that contains it are shaded.

We need some definitions. A symmetric matrix $D \in \mathbb{R}^{n\times n}$ is called *positive definite* if $x^T D x > 0$ for all non-zero $x \in \mathbb{R}^n$. A set of vectors $E \subseteq \mathbb{R}^n$ given by

$$E = E(z, D) = \{ x \in \mathbb{R}^n : (x - z)^T D^{-1}(x - z) \leqslant 1 \}$$

for a positive definite symmetric matrix $D \in \mathbb{R}^{n\times n}$ and a vector $z \in \mathbb{R}^n$ is called an *ellipsoid* with center z. If $D \in \mathbb{R}^{n\times n}$ is non-singular and $b \in \mathbb{R}^n$, then the mapping $S : \mathbb{R}^n \to \mathbb{R}^n$ given by $S(x) = Dx + b$ is called an *affine transformation.* We further write $S(L)$ for the *image* of $L \subseteq \mathbb{R}^n$ under S, i.e., $S(L) = \{y \in \mathbb{R}^n : y = S(x) \text{ for some } x \in \mathbb{R}^n\}$. The *volume* of a set $L \subseteq \mathbb{R}^n$ if finally defined as $\text{Vol}(L) = \int_{x\in L} dx$.

Let $P = \{x \in \mathbb{R}^n : Ax \geqslant b\}$ for some $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^m$. To decide whether P is non-empty, the ellipsoid method generates a sequence $\{E_t\}$ of ellipsoids E_t with centers x_t. If $x_t \in P$, then P is non-empty and the method stops. If $x_t \notin P$, then one of the constraints is violated, i.e., there exists a row j of A such that $a_j^T x_t < b_j$. Therefore, P is contained in the half-space $\{x \in \mathbb{R}^n : a_j^T x \geqslant a_j^T x_t\}$, and in particular in the intersection of this half-space with E_t, which we will call a *half-ellipsoid.*

The following is the key result underlying the ellipsoid method. It states that there exists a new ellipsoid E_{t+1} that contains the half-ellipsoid and whose volume is only a fraction of the volume of E_t. This situation is illustrated in Figure 6.2.

THEOREM 6.2. *Let* $E = E(z, D)$ *be an ellipsoid in* $\mathbb{R}^n$ *and* $a \in \mathbb{R}^n$ *non-zero. Consider the half-space* $H = \{x \in \mathbb{R}^n : a^T x \geqslant a^T z\}$, *and let*

$$z' = z + \frac{1}{n+1}\frac{Da}{\sqrt{a^T D a}},$$
$$D' = \frac{n^2}{n^2-1}\left(D - \frac{2}{n+1}\frac{Daa^T D}{a^T D a}\right).$$

Then D' *is symmetric and positive definite, and therefore* $E' = E(z', D')$ *is an ellipsoid. Moreover,* $E \cap H \subseteq E'$ *and* $\mathrm{Vol}(E') < e^{-1/(2(n+1))}\mathrm{Vol}(E)$.

If the procedure is repeated, it will either find a point in P or generate smaller and smaller ellipsoids containing P. In the next lecture, this procedure will be turned into an algorithm by observing that the volume of P must either be zero or larger than a certain threshold that depends on the size of the description of P.

We now sketch the proof of Theorem 6.2. We use the following lemma about affine transformations, which is not hard to prove.

LEMMA 6.3. *Let* $S : \mathbb{R}^n \to \mathbb{R}^n$ *be an affine transformation given by* $S(x) = Dx + b$ *and let* $L \subseteq \mathbb{R}^n$. *Then,* $\mathrm{Vol}(S(L)) = |\det(D)|\mathrm{Vol}(L)$.

Proof sketch of Theorem 6.2. We prove the theorem for $E = \{x \in \mathbb{R}^n : x^T x \leqslant 1\}$ and $H = \{x \in \mathbb{R}^n : x_1 \geqslant 0\}$. Since every pair of an ellipsoid and a hyperplane as in the statement of the theorem can be obtained from E and H via some affine transformation, the general case then follows by observing that affine transformations preserve inclusion and, by Lemma 6.3, relative volume of sets.

Let $e_1 = (1, 0, \ldots, 0)^T$. Then,

$$\begin{aligned} E' &= E\left(\frac{e_1}{n+1}, \frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^T\right)\right) \\ &= \left\{x \in \mathbb{R}^n : \frac{n^2-1}{n^2}\sum_{i=1}^{n} x_i^2 + \frac{1}{n^2} + \frac{2(n+1)}{n^2}x_1(x_1-1) \leqslant 1\right\}. \end{aligned}$$

Consider an arbitrary $x \in E \cap H$, and observe that $0 \leqslant x_1 \leqslant 1$ and $\sum_{i=1}^{n} x_i^2 \leqslant 1$. It is easily verified that $x \in E'$ and thus $E \cap H \subseteq E'$.

Now consider the affine transformation $F : \mathbb{R}^n \to \mathbb{R}^n$ given by

$$F(x) = \frac{e_1}{n+1} + \left(\frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^T\right)\right)^{\frac{1}{2}} x.$$

It is not hard to show that $E' = F(E)$. Therefore, by Lemma 6.3,

$$\begin{aligned} \frac{\mathrm{Vol}(E')}{\mathrm{Vol}(E)} &= \sqrt{\det\left(\frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^T\right)\right)} \\ &= \left(\frac{n^2}{n^2-1}\right)^{\frac{n}{2}}\left(1 - \frac{2}{n+1}\right)^{\frac{1}{2}} = \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{\frac{n-1}{2}} \\ &= \left(1 - \frac{1}{n+1}\right)\left(1 + \frac{1}{n^2-1}\right)^{\frac{n-1}{2}} < e^{-\frac{1}{n+1}}\left(e^{\frac{1}{n^2-1}}\right)^{\frac{n-1}{2}} = e^{-\frac{1}{2(n+1)}}, \end{aligned}$$

where the strict inequality follows by using twice that $1 + a < e^a$ for all $a \neq 0$. □

A more detailed description of the ellipsoid method and an overview of the proof of correctness will be given in the next lecture.

7 The Ellipsoid Method

Consider a polytope $P = \{x \in \mathbb{R}^n : Ax \geqslant b\}$, given by a matrix $A \in \mathbb{Z}^{m\times n}$ and a vector $b \in \mathbb{Z}^m$. Assume for now that P is bounded and either empty or full-dimensional. Here, P is called *full-dimensional* if $\mathrm{Vol}(P) > 0$. The ellipsoid method takes the following steps to decide whether P is non-empty:

1. Let U be the largest absolute value among the entries of A and b, and define
   $\nu_0 = 0, \quad D_0 = n(nU)^{2n}I, \quad E_0 = E(\nu_0, D_0),$
   $V = (2n)^n(nU)^{n^2}, \quad \nu = n^{-n}(nU)^{-n^2(n+1)},$
   $t^* = \lceil 2(n+1)\log(V/\nu)\rceil.$
2. For $t = 0, \dots, t^*$, do the following:
   (a) If $t = t^*$ then stop; P is empty.
   (b) If $x_t \in P$ then stop; P is non-empty.
   (c) Find a violated constraint, i.e., a row j such that $a_j^T x_t < b_j$.
   (d) Let $E_{t+1} = E(x_{t+1}, D_{t+1})$ with

$$x_{t+1} = x_t + \frac{1}{n+1}\frac{D_t a_j}{\sqrt{a_j^T D_t a_j}},$$
$$D_{t+1} = \frac{n^2}{n^2-1}\left(D_t - \frac{2}{n+1}\frac{D_t a_j a_j^T D_t}{a_j^T D_t a_j}\right).$$

The ellipsoid method is a so-called *interior point method*, because it traverses the interior of the feasible set rather than following its boundary.

7.1 Proof of Correctness

Observe that E_0 is a ball centered at the origin. Given Theorem 6.2, and assuming that (i) $P \subseteq E_0$ and $\mathrm{Vol}(E_0) < V$ and that (ii) P is either empty or $\mathrm{Vol}(P) > \nu$, correctness of the ellipsoid method is easy to see: it either finds a point in P, thereby proving that P is non-empty, or an ellipsoid $E_{t^*} \supseteq P$ with $\mathrm{Vol}(E_{t^*}) < e^{-t^*/2(n+1)}\mathrm{Vol}(E_0) < (\nu/V)\mathrm{Vol}(E_0) < \nu$, in which case P must be empty.

We now show that the above assumptions hold, starting with the inclusion of P in E_0 and the volume of E_0. We use the following lemma.

Lemma 7.1. *Let* $A \in \mathbb{Z}^{m\times n}$ *and* $b \in \mathbb{R}^m$. *Let* U *be the largest absolute value among the entries of* A *and* b. *Then every extreme point* x *of the polytope* $P = \{x' \in \mathbb{R}^n : Ax' \geqslant b\}$ *satisfies* $-(nU)^n \leqslant x_i \leqslant (nU)^n$ *for all* $i = 1, \dots, n$.

Proof. Any extreme point x can be written as $x = \hat{A}^{-1}\hat{b}$ for some invertible submatrix $\hat{A} \in \mathbb{Z}^{n\times n}$ of A and subvector $\hat{b} \in \mathbb{R}^n$ of b, corresponding to n linearly independent constraints that are active at x. By Cramer's rule,

$$x_i = \frac{\det \hat{A}^i}{\det \hat{A}},$$

where $\hat{A}^i$ is the matrix obtained by replacing the ith column of $\hat{A}$ by $\hat{b}$. Then, for $i = 1, \dots, n$,

$$\left|\det \hat{A}^i\right| = \left|\sum_{\sigma\in S_n} (-1)^{|\sigma|} \prod_{j=1}^{n} \hat{a}^i_{j,\sigma(j)}\right| \leqslant \sum_{\sigma\in S_n} \prod_{i=1}^{n} |\hat{a}^i_{j,\sigma(j)}| \leqslant n!U^n \leqslant (nU)^n,$$

where $|\sigma|$ is the number of inversions of permutation $\sigma \in S_n$, i.e., the number of pairs i, j such that $i < j$ and $\sigma(i) > \sigma(j)$. Moreover, $\det(\hat{A}) \neq 0$ since $\hat{A}$ is invertible, and $|\det(\hat{A})| \geqslant 1$ since all entries of A are integers. Therefore, $|x_i| \leqslant (nU)^n$ for all $i = 1, \dots, n$. □

If P is bounded, it is therefore contained in a cube with side length $2(nU)^n$. The ball E_0 contains this cube and is itself contained in a cube of volume $V = (2n)^n(nU)^{n^2}$, and thus $P \subseteq E_0$ and $\mathrm{Vol}(E_0) \leqslant V$.

We now turn to the lower bound on the volume of P in the case when it is non-empty.

LEMMA 7.2. *Consider a full-dimensional and bounded polytope* $P = \{x \in \mathbb{R}^n : Ax \geqslant b\}$*, where* $A \in \mathbb{Z}^{m\times n}$ *and* $b \in \mathbb{Z}^m$ *and all entries have absolute value at most* U*. Then* $\mathrm{Vol}(P) > n^{-n}(nU)^{-n^2(n+1)}$.

Proof sketch. If P is full-dimensional and bounded and has at least one extreme point, it has $n + 1$ extreme points $v^0, \dots, v^n$ that do not lie on a common hyperplane. Let

$$Q = \left\{ x \in \mathbb{R}^n : x = \sum_{k=0}^{n} \lambda_k v^k, \sum_{k=0}^{n} \lambda_k = 1, \lambda_k \geqslant 0 \right\}.$$

Clearly, $Q \subseteq P$ and thus $\mathrm{Vol}(Q) \leqslant \mathrm{Vol}(P)$. It can now be shown that

$$\mathrm{Vol}(Q) = \frac{1}{n!}\left|\det \begin{pmatrix} 1 & \cdots & 1 \\ v^0 & \cdots & v^n \end{pmatrix}\right|.$$

The ith coordinate of v^k is a rational number p^k_i/q^k_i, and by the same argument as in the proof of Lemma 7.1, $|q^k_i| \leqslant (nU)^n$ and $|p^k_i| \geqslant 1$. Therefore,

$$\begin{aligned} \mathrm{Vol}(P) \geqslant \mathrm{Vol}(Q) &\geqslant \frac{1}{n!}\left|\frac{1}{\prod_{i=1}^n \prod_{k=0}^n q^k_i}\right| \\ &> \frac{1}{n^n}\frac{1}{\prod_{i=1}^n \prod_{k=0}^n (nU)^n} = n^{-n}(nU)^{-n^2(n+1)}. \end{aligned}$$

□

So far we have assumed that the polytope P is bounded and full-dimensional. We finally lift these assumptions. By Lemma 7.1, all extreme points of P lie in the set $P_B = \{x \in P : |x_i| \leqslant (nU)^n \text{ for all } i = 1, \dots, n\}$. Moreover, P is non-empty if and only if it has an extreme point. We can therefore test for non-emptiness of P_B instead of P, and P_B is a bounded polytope.

For a polytope P that is not full-dimensional, it is not the case that $\mathrm{Vol}(P) < \nu$ implies $P = \emptyset$, and the ellipsoid method can fail. The following result shows, however, that we can slightly perturb P to obtain a polytope that is either empty or full-dimensional.

LEMMA 7.3. *Let* $P = \{x \in \mathbb{R}^n : Ax \geqslant b\}$*, where* $A \in \mathbb{Z}^{m \times n}$ *and* $b \in \mathbb{Z}^m$ *and all entries have absolute value at most* U*. Let*

$$P_\epsilon = \{\, x \in \mathbb{R}^n : Ax \geqslant b - \epsilon e\}$$

where

$$\epsilon = \frac{1}{2(n+1)} \left((n+1)U\right)^{-(n+1)}$$

and $e^T = (1, \dots, 1)$*. Then,* $P_\epsilon = \emptyset$ *if and only if* $P = \emptyset$*, and either* $P_\epsilon = \emptyset$ *or* $\mathrm{Vol}(P) > 0$*.*

Proof. We first show that emptiness of P implies emptiness of P_ϵ. If P is empty, then the linear program $\min\{0^T x : Ax \geqslant b\}$ is infeasible and its dual $\max\{\lambda^T b : \lambda^T A = 0^T, \lambda \geqslant 0\}$ is unbounded. There thus has to exist a basic feasible solution λ to the $n+1$ constraints $\lambda^T A = 0^T$, $\lambda^T b = 1$, and $\lambda \geqslant 0$, and, by Lemma 7.1, $\lambda_i \leqslant ((n+1)U)^{n+1}$ for all i. Since λ is a BFS, at most $n+1$ of its components are non-zero, and therefore $\sum_i^m \lambda_i \leqslant (n+1)\left((n+1)U\right)^{n+1}$ and $\lambda^T(b - \epsilon e) = 1 - \epsilon \sum_{i=1}^m \lambda_i \geqslant \frac{1}{2} > 0$. This means that the dual remains unbounded, and the primal infeasible, if we replace b by $b - \epsilon e$, and thus $P_\epsilon = \emptyset$.

It remains to be shown that P_ϵ is full-dimensional if P is non-empty. For this, consider $x \in P$ and let

$$Y = \left\{ y \in \mathbb{R}^n : x_i - \frac{\epsilon}{nU} \leqslant y_i \leqslant x_i + \frac{\epsilon}{nU} \text{ for all } i = 1, \dots, n \right\}.$$

It is easy to see that Y has volume $(2\epsilon/(nU))^n > 0$ and that $Y \subseteq P_\epsilon$. Thus P_ϵ must be full-dimensional. □

The general case of polytopes P that potentially are unbounded and not full-dimensional can thus be handled by computing the bounded polytope P_B, perturbing it, and then running the ellipsoid method on the resulting polytope.

7.2 The Complexity of the Ellipsoid Method

For a bounded and full-dimensional polytope P given by a matrix A and vector b with integer entries bounded by U, the ellipsoid method decides whether P is empty or not in

$O(n \log(V/\nu)) = O(n^4 \log(nU))$ iterations. It can further be shown that $O(n^6 \log(nU))$ iterations suffice even when P might be unbounded or not full-dimensional.

For the ellipsoid method to have a polynomial running time, however, the number of operations in each iteration also has to be bounded by a polynomial function of n and $\log U$. A potential problem is that the computation of the new ellipsoid involves taking a square root. This means that in general calculations cannot be done exactly, and intermediate results have to be stored with sufficiently many bits to ensure that errors don't accumulate. It turns out that the algorithm can be made to work, with the same asymptotic number of iterations as above, when only $O(n^3 \log U)$ bits are used for each intermediate value. The proof of this result is very technical.

The ellipsoid method has high theoretical significance, because it provided the first polynomial-time algorithm for linear programming and can also be applied to larger classes of convex optimization problems. In practice, however, both the simplex method and a different interior point method, *Karmarkar's algorithm*, tend to be much faster. It turns out that the latter also has a better worst-case performance than the ellipsoid method.

8 Graphs and Flows

Lectures 8 through 11 will be concerned with flow problems on graphs and networks.

A directed *graph*, or *network*, $G = (V, E)$ consists of a set V of *vertices* and a set $E \subseteq V \times V$ of *edges*. When the relation E is symmetric, G is called an undirected graph, and we can write edges as unordered pairs $\{i, j\} \in E$ for $i, j \in V$. The *degree* of vertex $i \in V$ in graph G is the number $|\{j \in V : (i, j) \in E \text{ or } (j, i) \in E\}|$ of other vertices connected to it by an edge. A *walk* from $u \in V$ to $w \in V$ is a sequence of vertices $v_1, \dots, v_k \in V$ such that $v_1 = u$, $v_k = w$, and $(v_i, v_{i+1}) \in E$ for $i = 1, \dots, k-1$. In a directed graph, we can also consider an undirected walk where $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$ for $i = 1, \dots, k-1$. A walk is a *path* if $v_1, \dots, v_k$ are pairwise distinct, and a *cycle* if furthermore $v_1 = v_k$. A graph that does not contain any cycles is called *acyclic*. A graph is called *connected* if for every pair of vertices $u, v \in V$ there is an undirected path from u to v. A *tree* is a graph that is connected and acyclic. A graph $G' = (V', E')$ is a subgraph of graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. In the special case where G' is a tree and $V' = V$, it is called a *spanning tree* of G.

8.1 Minimum Cost Flows

Consider a network $G = (V, E)$ with $|V| = n$, and let $b \in \mathbb{R}^n$. Here, b_i denotes the amount of flow that enters or leaves the network at vertex $i \in V$. If $b_i > 0$, we say that i is a *source* supplying b_i units of flow. If $b_i < 0$, we say that i is a *sink* with a demand of $|b_i|$ units of flow. Further let $C, \underline{M}, \overline{M} \in \mathbb{R}^{n \times n}$, where c_{ij} denotes the cost associated with one unit of flow on edge $(i, j) \in E$, and $\underline{m}_{ij}$ and $\overline{m}_{ij}$ respectively denote lower and upper bounds on the flow across this edge. The minimum cost flow problem then asks for flows x_{ij} that conserve the flow at each vertex, respect the upper and lower bounds, and minimize the overall cost. Formally, $x \in \mathbb{R}^{n \times n}$ is a *minimum cost flow* of G if it is an optimal solution of the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ \text{subject to} \quad & b_i + \sum_{j:(j,i) \in E} x_{ji} = \sum_{j:(i,j) \in E} x_{ij} && \text{for all } i \in V, \\ & \underline{m}_{ij} \leqslant x_{ij} \leqslant \overline{m}_{ij} && \text{for all } (i,j) \in E. \end{aligned}$$

Note that $\sum_{i \in V} b_i = 0$ is required for any feasible flows to exist, and we make this assumption in the following. We further assume without loss of generality that the network G is connected. Otherwise the problem can be decomposed into several smaller problems that can be solved independently. An important special case is that of *uncapacitated flows*, where $\underline{m}_{ij} = 0$ and $\overline{m}_{ij} = \infty$ for all $(i, j) \in E$.

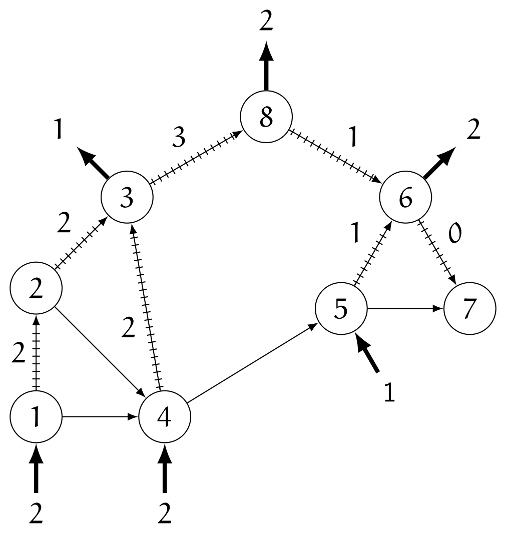

Figure 8.1: A flow network with a spanning tree T indicated by hatched edges. Since the network is uncapacitated, we have to set $L = E \setminus T$ and $U = \emptyset$, and thus flows are zero for edges not in T. Flows for the edges in T can be determined inductively starting from the leafs. Note that the resulting spanning tree solution is feasible.

The minimum cost flow problem is a linear programming problem, with constraints of the form $Ax = b$ where

$$a_{ik} = \begin{cases} 1 & \text{kth edge starts at vertex } i, \\ -1 & \text{kth edge ends at vertex } i, \\ 0 & \text{otherwise.} \end{cases}$$

Given this rather simple structure, we may hope that minimum cost flow problems are easier to solve than general linear programs. Indeed, we will see that basic feasible solutions of a minimum cost flow problem take a special form, and will obtain an algorithm that exploits this form.

8.2 Spanning Tree Solutions

Consider a minimum cost flow problem for a connected network $G = (V, E)$. A solution x to this problem is called *spanning tree solution* if there exists a spanning tree (V, T) of G and two sets $L, U \subseteq E$ with $L \cap U = \emptyset$ and $L \cup U = E \setminus T$ such that $x_{ij} = \underline{m}_{ij}$ if $(i,j) \in L$ and $x_{ij} = \overline{m}_{ij}$ if $(i,j) \in U$. For every choice of T, L and U, the flow conservation constraints uniquely determine the values x_{ij} for $(i,j) \in T$. An example is shown in Figure 8.1.

It is not hard to show that the basic solutions of a minimum cost flow problem are precisely its spanning tree solutions.

Theorem 8.1. *A flow vector is a basic solution of a minimum cost flow problem if and only if it is a spanning tree solution.*

8.3 The Network Simplex Method

We will now derive a variant of the simplex method, the network simplex method, that works directly with spanning tree solutions. The network simplex method maintains a feasible solution for the primal and a corresponding dual solution, but unlike the simplex method does not guarantee that these two solutions satisfy complementary slackness. Rather, it uses a separate condition to either establish both dual feasibility and complementary slackness, and thus optimality, or identify a new spanning tree solution.

The Lagrangian of the minimum cost flow problem is

$$\begin{aligned} L(x,\lambda) &= \sum_{(i,j)\in E} c_{ij}x_{ij} - \sum_{i\in V}\lambda_i\Bigg(\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} - b_i\Bigg) \\ &= \sum_{(i,j)\in E}(c_{ij} - \lambda_i + \lambda_j)x_{ij} + \sum_{i\in V}\lambda_i b_i \end{aligned} \tag{8.1}$$

Let $\bar{c}_{ij} = c_{ij} - \lambda_i + \lambda_j$ be the *reduced cost* of edge $(i,j) \in E$. Dual feasibility requires that $\bar{c}_{ij} \geqslant 0$ whenever $\overline{m}_{ij} = \infty$, and holds trivially if all edges are subject to finite capacities. Minimizing $L(x,\lambda)$ subject to the regional constraints $\underline{m}_{ij} \leqslant x_{ij} \leqslant \overline{m}_{ij}$ for $(i,j) \in E$ further yields the following complementary slackness conditions:

$$\begin{aligned} \bar{c}_{ij} > 0 \quad &\text{implies} \quad x_{ij} = \underline{m}_{ij}, \\ \bar{c}_{ij} < 0 \quad &\text{implies} \quad x_{ij} = \overline{m}_{ij}, \text{ and} \\ \underline{m}_{ij} < x_{ij} < \overline{m}_{ij} \quad &\text{implies} \quad \bar{c}_{ij} = 0. \end{aligned}$$

Assume that x is a basic feasible solution associated with sets T, U, and L. Then the system of equations

$$\lambda_{|V|} = 0, \qquad \lambda_i - \lambda_j = c_{ij} \quad \text{for all } (i,j) \in T$$

has a unique solution, which in turn allows us to compute $\bar{c}_{ij}$ for all edges $(i,j) \in E$. Note that $\bar{c}_{ij} = 0$ for all $(i,j) \in T$ by construction, so the third complementary slackness condition is always satisfied.

Pivoting

If $\bar{c}_{ij} \geqslant 0$ for all $(i,j) \in L$ and $\bar{c}_{ij} \leqslant 0$ for all $(i,j) \in U$, dual feasibility and the first two complementary slackness are satisfied as well, meaning that the solution is optimal. Otherwise, consider an edge (i,j) that violates these conditions, and observe that this edge and the edges in T forms a unique cycle C. Since (i,j) is the only edge in C with non-zero reduced cost, we can decrease the objective by pushing flow along C to increase x_{ij} if $\bar{c}_{ij}$ is negative and decrease x_{ij} if $\bar{c}_{ij}$ is positive. Doing so will change the flow on all edges in C by the same amount, with the direction of the change depending on whether a specific edge is oriented in the same or the opposite direction as (i,j).

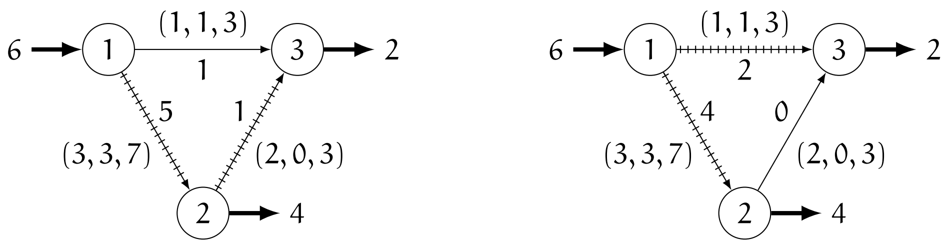

Figure 8.2: Flow network before and after a pivoting step. Edge (i,j) is labeled with the vector $(c_{ij}, \underline{m}_{ij}, \overline{m}_{ij})$ and the current flow x_{ij}, and spanning trees are indicated by hatched edges. In the situation shown on the left, we have $\lambda_3 = 0$, $\lambda_2 = c_{23} + \lambda_3 = 2$, and $\lambda_1 = c_{12} + \lambda_2 = 5$, and thus $\bar{c}_{13} = c_{13} - \lambda_1 + \lambda_3 = -4$. If we push one unit of flow around the cycle $1,3,2,1$, the flow on $(2,3)$ reaches the lower bound of $\underline{m}_{23} = 0$ and we obtain a new spanning tree with edges $(1,2)$ and $(1,3)$. The new situation is shown on the right. Now, $\lambda_3 = 0$, $\lambda_1 = c_{13} + \lambda_3 = 1$, and $\lambda_2 = \lambda_1 - c_{12} = -2$, and thus $\bar{c}_{23} = c_{23} - \lambda_2 + \lambda_3 = 4$. Since this is positive and $x_{23} = \underline{m}_{23}$, we have found an optimal solution.

Let $\underline{B}, \overline{B} \subseteq C$ respectively denote the sets of edges whose flow is to decrease or increase, and let

$$\delta = \min\left\{ \min_{(k,\ell)\in\underline{B}} \{x_{k\ell} - \underline{m}_{k\ell}\},\ \min_{(k,\ell)\in\overline{B}} \{\overline{m}_{k\ell} - x_{k\ell}\} \right\}.$$

be the maximum amount of flow that can be pushed along C. If $\delta = \infty$, the problem is unbounded. If $\delta = 0$ or if the minimum is attained for more than one edge, the problem is degenerate. Otherwise, pushing δ units of flow along C yields a unique edge $(k,\ell) \in C$ whose flow is either $\underline{m}_{k\ell}$ or $\overline{m}_{k\ell}$. If $(k,\ell) \in T$, we obtain a new BFS with spanning tree $(T \setminus \{(k,\ell)\}) \cup \{(i,j)\}$. If instead $(k,\ell) = (i,j)$, we obtain a new BFS where (i,j) has moved from U to L, or vice versa. An example of the pivoting step is given in Figure 8.2.

In the absence of degeneracies the value of the objective function decreases in every iteration of the network simplex method, and an optimal solution or a certificate of unboundedness is found after a finite number of iterations. If a degenerate solution is encountered it will still be possible to identify a new spanning tree or even a new BFS, but extra care may be required to ensure convergence. This is beyond the scope of this course.

Finding an initial feasible spanning tree solution

Consider a minimum cost flow problem for a network (V,E) and assume without loss of generality that $\underline{m}_{ij} = 0$ for all $(i,j) \in E$. If this is not the case, we can instead consider the problem obtained by setting $\underline{m}_{ij}$ to zero, $\overline{m}_{ij}$ to $\overline{m}_{ij} - \underline{m}_{ij}$, and replacing b_i by $b_i - \underline{m}_{ij}$ and b_j by $b_j + \underline{m}_{ij}$. A solution with flows x_{ij} for the new problem then corresponds to a solution with flows $x_{ij} + \underline{m}_{ij}$ for the original problem.

We now modify the problem such that the set of optimal solutions remains the same, assuming that the problem was feasible, but an initial feasible spanning tree solution is easy to find. For this, we introduce a dummy vertex $d \notin V$ and uncapacitated dummy edges $E' = \{(i, d) : i \in V, b_i \geqslant 0\} \cup \{(d, i) : i \in V, b_i < 0\}$ with cost equal to $\sum_{(i,j)\in E} c_{ij}$. It is easy to see that a dummy edge has positive flow in some optimal solution of the new problem if and only if the original problem is infeasible. Furthermore, a feasible spanning tree solution is now easily obtained by letting $T = E'$, $x_{id} = b_i$ for all $i \in V$ with $b_i > 0$, $x_{di} = -b_i$ for all $i \in V$ with $b_i < 0$, and $x_{ij} = 0$ otherwise.

8.4 Integrality of Optimal Solutions

Since the network simplex method does not require any divisions, any finite optimal solution it obtains for a problem with integer constants is also integral.

THEOREM 8.2. *Consider a minimum cost flow problem that is feasible and bounded. If* b_i *is integral for all* $i \in V$ *and* $\underline{m}_{ij}$ *and* $\overline{m}_{ij}$ *are integral for all* $(i, j) \in E$*, then there exists an integral optimal solution. If* c_{ij} *is integral for all* $(i, j) \in E$*, then there exists an integral optimal solution to the dual.*

9 Transportation and Assignment Problems

We will now consider several special cases of the minimum cost flow problem: the transportation problem, the assignment problems, the maximum flow problem, and the shortest path problem.

9.1 The Transportation Problem

In the *transportation problem* we are given a set of *suppliers* $i = 1, \dots, n$ producing s_i units of a good and a set of *consumers* $j = 1, \dots, m$ with demands d_j such that $\sum_{i=1}^{n} s_i = \sum_{j=1}^{m} d_j$. The cost of transporting one unit of the good from supplier i to consumer j is c_{ij}, and the goal is to match supply and demand while minimizing overall transportation cost. This can be formulated as an uncapacitated minimum cost flow problem on a *bipartite network*, i.e., a network $G = (S \uplus C, E)$ with $S = \{1, \dots, n\}$, $C = \{1, \dots, m\}$, and $E \subseteq S \times C$. As far as optimal solutions are concerned, edges not contained in E are equivalent to edges with a very large cost. We can thus restrict our attention to the case where $E = S \times C$, known as the *Hitchcock transportation problem*:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = d_j \quad \text{for all } j = 1, \dots, m \\
& \sum_{j=1}^{m} x_{ij} = s_i \quad \text{for all } i = 1, \dots, n \\
& x_{ij} \geqslant 0 \quad \text{for all } i, j.
\end{aligned}$$

It turns out that transportation problems already capture the full expressiveness of minimum cost flow problems.

Theorem 9.1. *Every minimum cost flow problem with finite capacities or non-negative costs has an equivalent transportation problem.*

Proof. Consider a minimum cost flow problem on a network $G = (V, E)$ with supplies or demands b_i, capacities $\underline{m}_{ij}$ and $\overline{m}_{ij}$, and costs c_{ij}. When constructing an initial feasible tree solution in the previous lecture, we saw that we can assume without loss of generality that $\underline{m}_{ij} = 0$ for all i, j. We can further assume that all capacities are finite: if some edge has infinite capacity but costs are non-negative then setting the capacity of this edge to a large enough number, for example $\sum_{i \in V} |b_i|$, does not affect the optimal solution of the problem.

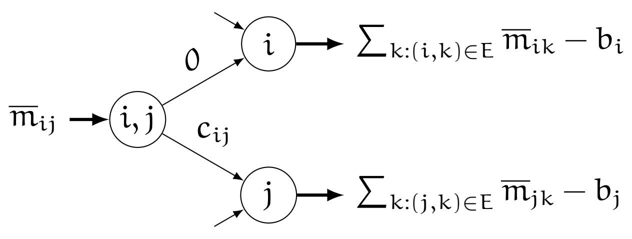
$\overline{m}_{ij}$
i,j
0
c_{ij}
i
j
$\sum_{k:(i,k)\in E} \overline{m}_{ik} - b_i$
$\sum_{k:(j,k)\in E} \overline{m}_{jk} - b_j$

Figure 9.1: Representation of flow conservation constraints by a transportation problem

We now construct a transportation problem as follows. For every vertex $i \in V$, we add a sink vertex with demand $\sum_k \overline{m}_{ik} - b_i$. For every edge $(i,j) \in E$, we add a source vertex with supply $\overline{m}_{ij}$, an edge to vertex i with cost $c_{ij,i} = 0$, and an edge to vertex j with cost $c_{ij,j} = c_{ij}$. The situation is shown in Figure 9.1.

We now claim that there exists a direct correspondence between feasible flows of the two problems, and that these flows have the same costs. To see this, let the flows on edges (ij,i) and (ij,j) be $\overline{m}_{ij} - x_{ij}$ and x_{ij}, respectively. The total flow into vertex i then is $\sum_{k:(i,k)\in E}(\overline{m}_{ik} - x_{ik}) + \sum_{k:(k,i)\in E} x_{ki}$, which must be equal to $\sum_{k:(i,k)\in E} \overline{m}_{ik} - b_i$. This is the case if and only if $b_i + \sum_{k:(k,i)\in E} x_{ki} - \sum_{k:(i,k)\in E} x_{ik} = 0$, which is the flow conservation constraint for vertex i in the original problem. $\square$

9.2 The Network Simplex Method in Tableau Form

When solving a transportation problem using the network simplex method, it is convenient to write it down in a tableau of the following form, where λ_i for $i = 1, \dots, n$ and μ_j for $j = 1, \dots, m$ are the dual variables corresponding to the flow conservation constraints for suppliers and consumers, respectively:

	μ_1	$\cdots$	μ_m	
λ_1	x_{11} $\boxed{c_{11}}$	$\cdots$ $\boxed{\cdots}$	x_{1m} $\boxed{c_{1m}}$	s_1
$\vdots$	$\vdots$ $\boxed{\vdots}$	$\ddots$ $\boxed{\ddots}$	$\vdots$ $\boxed{\vdots}$	$\vdots$
λ_n	x_{n1} $\boxed{c_{n1}}$	$\cdots$ $\boxed{\cdots}$	x_{nm} $\boxed{c_{nm}}$	s_n
	d_1	$\cdots$	d_m	

Consider the Hitchcock transportation problem given by the following tableau:

$\boxed{5}$	$\boxed{3}$	$\boxed{4}$	$\boxed{6}$	8
$\boxed{2}$	$\boxed{7}$	$\boxed{4}$	$\boxed{1}$	10
$\boxed{5}$	$\boxed{6}$	$\boxed{2}$	$\boxed{4}$	9
6	5	8	8	

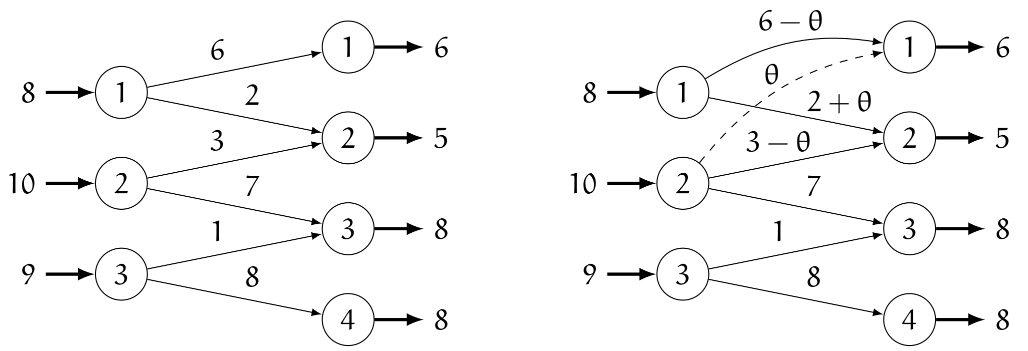

Figure 9.2: Initial basic feasible solution of a transportation problem (left) and a cycle along which the overall cost can be decreased (right)

An initial BFS can be found by iteratively considering pairs (i, j) of supplier i and customer j, increasing x_{ij} until either the supply s_i or the demand d_j is satisfied, and moving to the next supplier in the former case or to the next customer in the latter. Since $\sum_i s_i = \sum_j d_j$, this process is guaranteed to find a feasible solution, and the corresponding spanning tree consists of the pairs (i, j) that have been visited. If at some point both the supply and the demand are satisfied at the same time, the resulting solution is degenerate. In the example, we can start by setting $x_{11} = \min\{s_1, d_1\} = 6$, moving to customer 2 and setting $x_{12} = 2$, moving to supplier 2 and setting $x_{22} = 3$, and so forth. The resulting spanning tree and flows are shown on the left of Figure 9.2.

To determine the values of the dual variables λ_i for $i = 1, \ldots, 3$ and μ_j for $j = 1, \ldots, 4$, observe that $\lambda_i - \mu_j = c_{ij}$ must be satisfied for all $(i, j) \in T$. By setting $\lambda_1 = 0$, we obtain a system of 6 linear equalities with 6 variables, which has a unique solution. It will finally be convenient to write down $\lambda_i - \mu_j$ for $(i, j) \notin T$, which we do in the upper right corner of the respective cells. The tableau now looks as follows:

	−5	−3	0	−2	
0	6 $\boxed{5}$	2 $\boxed{3}$	0 $\boxed{4}$	2 $\boxed{6}$	8
4	9 $\boxed{2}$	3 $\boxed{7}$	7 $\boxed{4}$	6 $\boxed{1}$	10
2	7 $\boxed{5}$	5 $\boxed{6}$	1 $\boxed{2}$	8 $\boxed{4}$	9
	6	5	8	8	

If $c_{ij} \geqslant \lambda_i - \mu_j$ for all $(i, j) \notin T$, the current flow would be optimal. In our example this condition is violated, for example, for $i = 2$ and $j = 1$. Edge $(2, 1)$ forms a unique cycle with the spanning tree T, and we would like to increase x_{21} by pushing flow along this cycle. Due to the special structure of the network, doing so will alternately increase and decrease the flow for edges along the cycle. In particular, increasing x_{21} by θ will increase x_{12} and decrease x_{11} and x_{22} by the same amount. The situation is shown on the right of Figure 9.2. Increasing x_{21} by the maximum amount of $\theta = 3$ and re-computing the values of the dual variables λ_1 and μ_j, we obtain the following tableau:

	-5	-3	-7	-9
0	3 $\boxed{5}$	5 $\boxed{3}$	7 $\boxed{4}$	9 $\boxed{6}$
-3	3 $\boxed{2}$	0 $\boxed{7}$	7 $\boxed{4}$	6 $\boxed{1}$
-5	0 $\boxed{5}$	-2 $\boxed{6}$	1 $\boxed{2}$	8 $\boxed{4}$

Now, $c_{24} < \lambda_2 - \mu_4$, and we can increase x_{24} by 7 to obtain the following tableau, which satisfies $c_{ij} \geqslant \lambda_i - \mu_j$ for all $(i, j) \notin T$ and therefore yields an optimal solution:

	-5	-3	-2	-4
0	3 $\boxed{5}$	5 $\boxed{3}$	2 $\boxed{4}$	4 $\boxed{6}$
-3	3 $\boxed{2}$	0 $\boxed{7}$	-1 $\boxed{4}$	7 $\boxed{1}$
0	5 $\boxed{5}$	3 $\boxed{6}$	8 $\boxed{2}$	1 $\boxed{4}$

9.3 The Assignment Problem

An instance of the assignment problem is given by n agents and n jobs, and costs c_{ij} for assigning job j to agent i. The goal is to assign exactly one job to each agent at a minimum overall cost, i.e., to

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{subject to} \quad & x_{ij} \in \{0, 1\} \quad \text{for all } i, j = 1, \dots, n \\
& \sum_{j=1}^{n} x_{ij} = 1 \quad \text{for all } i = 1, \dots, n \\
& \sum_{i=1}^{n} x_{ij} = 1 \quad \text{for all } j = 1, \dots, n
\end{aligned} \tag{9.1}$$

Except for the integrality constraints, this problem is a special case of the Hitchcock transportation problem. All basic solutions of the *LP relaxation* of this problem, which is obtained by replacing the integrality constraint $x_{ij} \in \{0, 1\}$ by $0 \leqslant x_{ij} \leqslant 1$, are spanning tree solutions and therefore integral. Thus, both the network simplex method and the general simplex method yield an optimal solution of the original problem when applied to the LP relaxation. This is not necessarily the case, for example, for the ellipsoid method.

10 Maximum Flows and Perfect Matchings

10.1 The Maximum Flow Problem

Consider a flow network (V, E) with a single source 1, a single sink n, and finite capacities $\overline{m}_{ij} = C_{ij}$ for all $(i, j) \in E$. We will also assume for convenience that $\underline{m}_{ij} = 0$ for all $(i, j) \in E$. The *maximum flow problem* then asks for the maximum amount of flow that can be sent from vertex 1 to vertex n, i.e., the goal is to

$$\begin{aligned} &\text{maximize} \quad \delta \\ &\text{subject to} \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} \delta & \text{if } i = 1 \\ -\delta & \text{if } i = n \\ 0 & \text{otherwise} \end{cases} \\ &\qquad 0 \leqslant x_{ij} \leqslant C_{ij} \quad \text{for all } (i,j) \in E. \end{aligned} \tag{10.1}$$

To see that this is again a special case of the minimum cost flow problem, set $c_{ij} = 0$ for all $(i, j) \in E$, and add an additional edge $(n, 1)$ with infinite capacity and cost $c_{n1} = -1$. Since the new edge $(n, 1)$ has infinite capacity, any feasible flow of the original network is also feasible for the new network. Cost is clearly minimized by maximizing the flow across the edge $(n, 1)$, which by the flow conservation constraints for vertices 1 and n maximizes flow through the original network. This kind of problem is known as a *circulation problem*, because there are no sources or sinks but flow merely circulates in the network.

10.2 The Max-Flow Min-Cut Theorem

Consider a flow network $G = (V, E)$ with capacities C_{ij} for all $(i, j) \in E$. A *cut* of G is a partition of V into two sets, and the capacity of a cut is defined as the sum of capacities of all edges across the partition. Formally, for $S \subseteq V$, the capacity of the cut $(S, V \setminus S)$ is given by

$$C(S) = \sum_{(i,j)\in E\cap(S\times(V\setminus S))} C_{ij}.$$

Assume that x is a feasible flow vector that sends δ units of flow from vertex 1 to vertex n. It is easy to see that δ is bounded from above by the capacity of any cut S with $1 \in S$ and $n \in V \setminus S$. Indeed, for $X, Y \subseteq V$, let

$$f(X, Y) = \sum_{(i,j)\in E\cap(X\times Y)} x_{ij}.$$

Then, for any $S \subseteq V$ with $1 \in S$ and $n \in V \setminus S$,

$$\begin{aligned}
\delta &= \sum_{i \in S} \left(\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} \right) \\
&= f(S,V) - f(V,S) \qquad (10.2) \\
&= f(S,S) + f(S, V \setminus S) - f(V \setminus S, S) - f(S,S) \\
&= f(S, V \setminus S) - f(V \setminus S, S) \leqslant f(S, V \setminus S) \leqslant C(S).
\end{aligned}$$

The following result states that this upper bound in fact tight, i.e., that there exists a flow of size equal to the minimum capacity of a cut that separates vertex 1 from vertex n.

THEOREM 10.1 (Max-flow min-cut theorem). *Let δ be the optimal solution of* (10.1) *for a network* (V, E) *with capacities* C_{ij} *for all* $(i,j) \in E$. *Then,*

$$\delta = \min\{\, C(S) : S \subseteq V, 1 \in S, n \in V \setminus S \,\}.$$

Proof. It remains to be shown that there exists a cut that separates vertex 1 from vertex n and has capacity equal to δ. Consider a feasible flow vector x. A path $P = v_0, v_1, \dots, v_k$ is called an *augmenting path* for x if $x_{v_{i-1}v_i} < C_{v_{i-1}v_i}$ or $x_{v_i v_{i-1}} > 0$ for every $i = 1, \dots, k$. If there exists an augmenting path from vertex 1 to vertex n, then we can push flow along the path, by increasing the flow on every forward edge and decreasing the flow on every backward edge along the path by the same amount, such that all constraints remain satisfied and the amount of flow from 1 to n increases.

Now assume that x is optimal, and let

$$S = \{1\} \cup \{\, i \in V : \text{there exists an augmenting path for } x \text{ from } 1 \text{ to } i \,\}.$$

By optimality of x, $n \in V \setminus S$. Moreover,

$$\delta = f(S, V \setminus S) - f(V \setminus S, S) = f(S, V \setminus S) = C(S).$$

The first equality holds by (10.2). The second equality holds because $x_{ij} = 0$ for every $(i,j) \in E \cap ((V \setminus S) \times S)$. The third equality holds because $x_{ij} = C_{ij}$ for every $(i,j) \in E \cap (S \times (N \setminus S))$. □

10.3 The Ford-Fulkerson Algorithm

The *Ford-Fulkerson algorithm* attempts to find a maximum flow by repeatedly pushing flow along an augmenting path, until such a path can no longer be found:

1. Start with a feasible flow vector x.
2. If there is no augmenting path from 1 to n, then stop.
3. Otherwise pick some augmenting path from 1 to n, and push a maximum amount of flow along this path without violating any constraints. Then go to Step 2.

Assume that all capacities are integral and that we start with an integral flow vector, e.g., the flow vector x such that $x_{ij} = 0$ for all $(i,j) \in E$. It is then not hard to see that the flow vector always remains integral and overall flow increases by at least one unit in each iteration. The algorithm is therefore guaranteed to find a maximum flow after a finite number of iterations. It can in fact be shown that $O(|E| \cdot |V|)$ iterations suffice if only augmenting paths with a minimum number of edges are used. Such an augmenting path can for example be found using breadth-first search, which requires $O(|E|)$ steps and leads to an overall running time of $O(|E|^2 \cdot |V|)$.

10.4 Max-Flow Min-Cut from Strong Duality

Consider the following formulation of the maximum flow problem as a minimum cost flow problem, which we have already discussed above:

$$\begin{aligned} \text{minimize} \quad & -x_{n1} \\ \text{subject to} \quad & \sum_{j:(i,j)\in E'} x_{ij} - \sum_{j:(j,i)\in E'} x_{ji} = 0 \quad \text{for all } i \in V \\ & 0 \leqslant x_{ij} \leqslant C_{ij} \quad \text{for all } (i,j) \in E \\ & x_{n1} \geqslant 0, \end{aligned}$$

where $E' = E \cup \{(n,1)\}$. The Lagrangian (8.1) becomes

$$L(x,\lambda) = (-1 - \lambda_n + \lambda_1)x_{n1} - \sum_{(i,j)\in E} (\lambda_i - \lambda_j)x_{ij},$$

which has a bounded minimum where $x_{n1} > 0$ only if $\lambda_1 - \lambda_n = 1$. We know from the general case that one of the dual variables can be set arbitrarily, so we let $\lambda_1 = 1$ and obtain $\lambda_n = 0$. For a fixed λ, $L(x,\lambda)$ is minimized by setting $x_{ij} = 0$ whenever $\lambda_i - \lambda_j < 0$ and $x_{ij} = C_{ij}$ whenever $\lambda_i - \lambda_j > 0$, and thus

$$g(\lambda) = \inf_x L(x,\lambda) = -\sum_{(i,j)\in E} \max(\lambda_i - \lambda_j, 0)C_{ij}.$$

By introducing new variables $d_{ij} \geqslant \max(\lambda_i - \lambda_j, 0)$ for $(i,j) \in E$, we obtain

$$g(\lambda) \geqslant -\sum_{(i,j)\in E} d_{ij}C_{ij},$$

with equality if $d_{ij} = \max(\lambda_i - \lambda_j, 0)$. We can thus maximize $g(\lambda)$ by minimizing $\sum_{(i,j)\in E} d_{ij}C_{ij}$ subject to $d_{ij} \geqslant \lambda_i - \lambda_j$ and $d_{ij} \geqslant 0$ for all $(i,j) \in E$, and obtain the following dual of (10.1):

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in E} d_{ij}C_{ij} \\ \text{subject to} \quad & d_{ij} - \lambda_i + \lambda_j \geqslant 0 \quad \text{for all } (i,j) \in E \\ & d_{ij} \geqslant 0 \quad \text{for all } (i,j) \in E \\ & \lambda_1 = 1, \quad \lambda_n = 0. \end{aligned}$$

It can be shown that this dual has an optimal solution in which $\lambda_i \in \{0, 1\}$ for all $i \in V$. By the complementary slackness conditions, the set $S = \{i \in V : \lambda_i = 1\}$ must then be a minimum cut, and the max-flow min-cut theorem follows from strong duality.

10.5 The Bipartite Matching Problem

A *matching* of a graph (V, E) is a set of edges that do not share any vertices, i.e., a set $M \subseteq E$ such for all $(s, t), (u, v) \in M$, $u \neq s \neq v$ and $u \neq t \neq v$. Matching M is called perfect if it covers every vertex, i.e., if $|M| = |V|/2$.

A graph is k-regular if every vertex has degree k. Using maximum flows it is easy to show that every k-regular bipartite graph, for $k \geqslant 1$, has a perfect matching. For this, consider a k-regular bipartite graph $(L \uplus R, E)$, orient all edges from L to R, and add two new vertices s and t and new edges (s, i) and (j, t) for every $i \in L$ and $j \in R$. Finally set the capacity of every new edge to 1, and that of every original edge to infinity. We can now send $|L|$ units of flow from s to t by setting the flow to 1 for every new edge and to $1/k$ for every original edge. By Theorem 8.2, there must exist an integral solution with the same value, and it is easy to see that such a solution corresponds to a perfect matching.

This result is a special case of a well-known characterization of the bipartite graphs that have a perfect matching. It should not come as a surprise that this characterization can be obtained from the max-flow min-cut theorem as well.

Theorem 10.2 (Hall's Theorem). *A bipartite graph* $G = (L \uplus R, E)$ *with* $|L| = |R|$ *has a perfect matching if and only if* $|N(X)| \geqslant |X|$ *for every* $X \subseteq L$*, where* $N(X) = \{j \in R : i \in X, (i, j) \in E\}$.

Proof. The direction from left to right is obvious: in a perfect matching, every vertex in X is matched to a different vertex in $N(X)$.

For the direction from right to left, assume that G does not have a perfect matching and again consider the graph with additional vertices s and t described above. The maximum flow from s to t is this graph must be smaller than $|L|$, so by the max-flow min-cut theorem there has to exist a cut $S \subseteq L \uplus R \cup \{s\}$ with $s \in S$ and $C(S) < |L|$. Let $L_S = L \cap S$, $R_S = R \cap S$, and $L_T = L \setminus S$. Since $C(S)$ is finite, $i \in S$ implies that $j \in S$ for every $(i, j) \in E$. On the one hand, this means that $N(L_S) \subseteq R_S$. On the other, the capacity of the cut must thus come precisely from the edges in $\{s\} \times L_T$ and $R_S \times \{t\}$. Each of these edges has capacity 1, so $C(S) = |L_T| + |R_S|$, and we obtain

$$|N(L_S)| \leqslant |R_S| = C(S) - |L_T| < |L| - |L_T| = |L_S|. \qquad \square$$

11 Shortest Paths and Minimum Spanning Trees

Consider a network (V, E) with associated costs c_{ij} for each edge $(i, j) \in E$, corresponding for example to the physical distance between vertices i and j or the cost of establishing a link between them. The *single-pair shortest path problem* then asks for a (directed) path from a given source $s \in V$ to a given destination $t \in V$ that has minimum cost, where the cost of a path is the sum of costs of its edges. The shortest path problem has numerous applications in transportation and communications, and also occurs frequently as a subproblem of more complex problems. It is a special case of the minimum cost flow problem, but can be solved more efficiently using specialized algorithms.

11.1 The Bellman Equations

It will be instructive to consider a destination $t \in V$ and simultaneously look for shortest paths from any vertex $i \in V \setminus \{t\}$ to t. This problems is sometimes called the *single-destination* shortest path problem, and is equivalent to the minimum cost flow problem on the same network where one unit of flow is to be routed from each vertex $i \in V \setminus \{t\}$ to t, i.e., the one with supply $b_i = 1$ at every vertex $i \in V \setminus \{t\}$ and demand $b_t = -(|V| - 1)$ at vertex t.

Let λ_i for $i \in V$ be the dual solution corresponding to an optimal spanning tree solution of this flow problem, and recall that for every edge $(i, j) \in E$ with $x_{ij} > 0$,

$$\lambda_i = c_{ij} + \lambda_j.$$

By setting $\lambda_t = 0$ and adding these equalities along a path from i to t, we see that λ_i is equal to the length of a shortest path from i to t. Moreover, since $b_i = 1$ for all $i \in V \setminus \{t\}$, and given $\lambda_t = 0$, the dual problem is to

$$\text{maximize} \sum_{i \in V \setminus \{t\}} \lambda_i \quad \text{subject to } \lambda_i \leqslant c_{ij} + \lambda_j \text{ for all } (i, j) \in E.$$

In an optimal solution, λ_i will thus be as large as possible subject to the constraints, i.e., it will satisfy the so-called *Bellman equations*

$$\lambda_i = \min_{j:(i,j) \in E} (c_{ij} + \lambda_j) \quad \text{for all } i \in V \setminus \{t\},$$

with $\lambda_t = 0$. The intuition behind these equalities is that in order to find a shortest path from i to t, one should choose the first edge (i, j) on the path in order to minimize the sum of the length of this edge and that of a shortest path from j to t. This situation is illustrated in Figure 11.1.

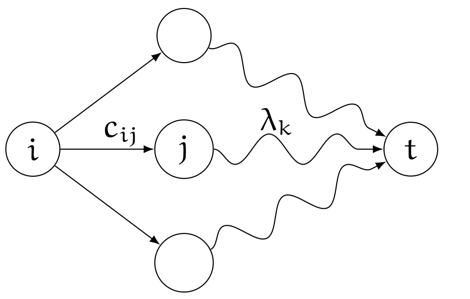

Figure 11.1: Illustration of the Bellman equations for the shortest path problem

11.2 The Bellman-Ford Algorithm

Let $\lambda_i(k)$ be the length of a shortest path from i to t that uses at most k edges. Then, $\lambda_t(k) = 0$ for all $k \geqslant 0$, and

$$\lambda_i(0) = \infty \quad \text{and}$$
$$\lambda_i(k) = \min_{j:(i,j)\in E} (c_{ij} + \lambda_j(k-1))$$

for all $i \in V \setminus \{t\}$ and $k \geqslant 1$.

The algorithm that successively computes $\lambda_i(k)$ for all i and larger and larger values of k is known as the *Bellman-Ford algorithm*. It is an example of a method called *dynamic programming*, which can be applied to problems that are decomposable into *overlapping subproblems* and have what is called *optimal substructure*, such that an overall solution can be constructed efficiently from solutions to the subproblems.

Note that $\lambda_i(|V|) < \lambda_i(|V|-1)$ for some $i \in V$ if and only if there exists a cycle of negative length, and that otherwise $\lambda_i = \lambda_i(|V|-1)$. In any case, $O(|V|)$ iterations of the Bellman-Ford algorithm suffice to determine λ_i. Each iteration requires $O(|E|)$ steps, for an overall running time of $O(|E| \cdot |V|)$. Given the values λ_i for all $i \in V$, a shortest path from i to t then leads along an edge $(i,j) \in E$ such that $\lambda_i = c_{ij} + \lambda_j$. Alternatively, one could store such a successor vertex for every vertex i while running the algorithm, and update it whenever $\lambda_i(k) < \lambda_i(k-1)$.

11.3 Dijkstra's Algorithm

The Bellman-Ford algorithm does not make any assumptions about edge lengths, and works in particular if some or all of them are negative. In the special case where all edges are known to have non-negative lengths, the running time can sometimes be decreased. The idea is to collect vertices in the order of increasing shortest path length to t. We assume from now on that $E = V \times V$, and set $c_{ij} = \infty$ if necessary. The following lemma will be useful.

Lemma 11.1. *Consider a graph with vertices* V *and edge lengths* $c_{ij} \geqslant 0$ *for all* $i,j \in V$. *Fix* $t \in V$ *and let* λ_i *denote the length of a shortest path from* $i \in V$ *to* t. *Let* $j \in V \setminus \{t\}$ *such that* $c_{jt} = \min_{i\in V\setminus\{t\}} c_{it}$. *Then,* $\lambda_j = c_{jt}$ *and* $\lambda_j = \min_{i\in V\setminus\{t\}} \lambda_i$.

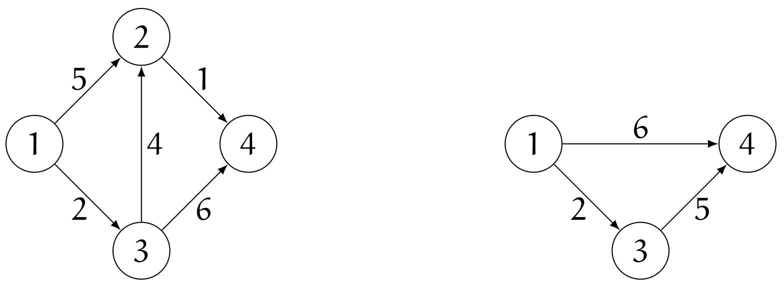

Figure 11.2: An iteration of Dijkstra's algorithm with $t = 4$. In the graph on the left, $c_{2t} = \min_{i \in V \setminus \{t\}} c_{it}$ and therefore, by Lemma 11.1, $\lambda_2 = c_{2t} = 1$. The graph on the right is then obtained by removing vertex 2 and updating c_{14} to $\min\{c_{14}, c_{12} + c_{24}\} = \min\{\infty, 5 + 1\} = 6$ and c_{34} to $\min\{c_{34}, c_{32} + c_{24}\} = \min\{6, 4 + 1\} = 5$.

Proof. Let $i \in V \setminus \{t\}$, consider a shortest path from i to t, and let (ℓ, t) be the last edge on this path. Then, $\lambda_i \geqslant \lambda_\ell \geqslant c_{\ell t} \geqslant c_{jt}$. This holds in particular for $i = j$, and on the other hand $\lambda_j \leqslant c_{jt}$. Thus $\lambda_j = c_{jt} \leqslant \lambda_i$. □

Dijkstra's algorithm uses this lemma to determine λ_j for a particular vertex j, removes j from the graph, and repeats the process for the new graph:

1. Find a vertex $j \in V \setminus \{t\}$ with $c_{jt} = \min_{i \in V \setminus \{t\}} c_{it}$. Set $\lambda_j = c_{jt}$.
2. For every vertex $i \in V \setminus \{j\}$, set $c_{it} = \min\{c_{it}, c_{ij} + c_{jt}\}$.
3. Remove vertex j from V. If $|V| > 1$, return to Step 1.

An example is shown in Figure 11.2.

The algorithm performs $|V| - 1$ iterations, each of which determines the new length of one edge for each of the remaining $O(|V|)$ vertices. The overall running time is thus $O(|V|^2)$. This improves on the Bellman-Ford algorithm in graphs with many edges, and is optimal in the sense that any algorithm for the single-destination shortest path problem has to inspect all of the edges, of which there are $\Omega(|V|^2)$ in the worst case.

One might wonder whether there exists a way to transform the edge lengths to make them non-negative without affecting the structure of the shortest paths, so that Dijkstra's algorithm could be used in the presence of negative lengths as well. Let λ_i be the length of a shortest path from vertex i to vertex t, and recall that $\lambda_i \leqslant c_{ij} + \lambda_j$ for all $(i, j) \in E$. Let $\bar{c}_{ij} = c_{ij} + \lambda_j - \lambda_i$. Then, $\bar{c}_{ij} \geqslant 0$ for every edge $(i, j) \in E$, and for an arbitrary path $v_1, v_2, \ldots, v_k$,

$$\sum_{i=1}^{k-1} \bar{c}_{v_i v_{i+1}} = \sum_{i=1}^{k-1} (c_{v_i v_{i+1}} + \lambda_{v_{i+1}} - \lambda_{v_i}) = \lambda_{v_k} - \lambda_{v_1} + \sum_{i=1}^{k-1} c_{v_i v_{i+1}}.$$

So indeed, changing edge lengths from c_{ij} to $\bar{c}_{ij}$ allows Dijkstra's algorithm to work correctly, and it does not affect the structure of the shortest paths.

This observation is not very useful in the context of single-pair or single-destination shortest path problems: we do not know the values λ_i, and computing them is at least as hard as the problem we are trying to solve. For the *all-pairs* shortest path problem, however, which requires us to find a shortest path between every pair of vertices $i, j \in V$, the situation is different. The straightforward solution to this problem is to run the Bellman-Ford algorithm $|V|$ times, once for every possible destination vertex. In a graph with $\Omega(|V|^2)$ edges, this leads to an overall running time of $|V| \cdot O(|V|^3) = O(|V|^4)$. Using the above observation, we can instead invoke the Bellman-Ford algorithm for one destination vertex t to obtain the shortest path lengths λ_i for all $i \in V$, and compute shortest paths for the remaining destination vertices by running Dijkstra's algorithm on the graph with edge lengths $\bar{c}_{ij}$. This improves the asymptotic running time to $O(|V|^3) + |V-1| \cdot O(|V|^2) = O(|V|^3)$.

11.4 Minimum Spanning Trees and Prim's Algorithm

The *minimum spanning tree problem* for a network (V, E) with associated costs c_{ij} for each edge $(i, j) \in E$ asks for a spanning tree of minimum cost, where the cost of a tree is the sum of costs of all its edges. This problem arises, for example, if one wishes to design a communication network that connects a given set of locations. The following property of minimum spanning trees will be useful.

THEOREM 11.2. *Let* (V, E) *be a graph with edge costs* c_{ij} *for all* $(i, j) \in E$. *Let* $U \subseteq V$ *and* $(u, v) \in U \times (V \setminus U)$ *such that* $c_{uv} = \min_{(i,j) \in U \times (V \setminus U)} c_{ij}$. *Then there exists a spanning tree of minimum cost that contains* (u, v).

Proof. Let $T \subseteq E$ be a spanning tree of minimum cost. If $(u, v) \in T$ we are done. Otherwise, $T \cup \{(u, v)\}$ contains a cycle, and there must be another edge $(u', v') \in T$ such that $(u', v') \in U \times (V \setminus U)$. Then, $(T \cup \{(u, v)\}) \setminus \{(u', v')\}$ is a spanning tree, and since (u, v) has minimum cost among the edges in $U \times (V \setminus U)$ its cost is no greater than that of T, and therefore minimum. $\square$

Prim's algorithm uses this property to inductively construct a minimum spanning tree. It proceeds as follows:

1. Set $U = \{1\}$ and $T = \emptyset$.
2. If $U = V$, return T. Otherwise find an edge $(u, v) \in U \times (V \setminus U)$ such that $c_{uv} = \min_{(i,j) \in U \times (V \setminus U)} c_{ij}$.
3. Add v to U and (u, v) to T, and return to Step 2.

Prim's algorithm is called a *greedy algorithm*, because it always chooses an edge of minimum cost.

Suppose that after each iteration, we compute and store for every vertex $j \in V \setminus U$ a minimum cost edge to any vertex in U, i.e., an edge $(i, j) \in U \times (V \setminus U)$ such that $c_{ij} = \min_{(i',j) \in U \times (V \setminus U)} c_{i'j}$. This only requires a comparison between the previously stored edge and the edge to the vertex added to U in the current iteration, and can be done in time $O(|V|)$ per iteration. It then suffices in Step 2 to minimize cost among vertices in $V \setminus U$, of which there are $O(|V|)$. Since the algorithm performs $|V| - 1$ iterations, it thus has an overall running time of $O(|V|^2)$.

12 Semidefinite Programming

Again consider the standard form (2.2) of a linear program,

$$\min\{\, c^T x : Ax = b, x \geqslant 0 \,\}.$$

The goal in linear programming is to optimize a linear objective over the intersection of the non-negative orthant $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x \geqslant 0\}$ with an affine space, described by the linear equation $Ax = b$. The non-negative orthant is a *convex cone*, i.e., a set $C \subseteq \mathbb{R}^n$ for which $\alpha x + \beta y \in C$ for all $\alpha, \beta \in \mathbb{R}$ with $\alpha, \beta \geqslant 0$ and all $x, y \in C$.

Semidefinite programming replaces the non-negative orthant with a different convex cone. Let $\mathbb{S}^n = \{X \in \mathbb{R}^{n\times n} : X^T = X\}$ be the set of all symmetric $n \times n$ matrices, and call $A \in \mathbb{S}^n$ *positive semidefinite*, denoted $A \succeq 0$, if $z^T Az \geqslant 0$ for all $z \in \mathbb{R}^n$. Let $\mathbb{S}^n_+ = \{A \in \mathbb{S}^n : A \succeq 0\}$. It is easy to see that $\mathbb{S}^n_+$ is a convex cone, henceforth called the convex cone of positive semidefinite matrices, or simply the positive semidefinite cone.

A linear function of $X \in \mathbb{S}^n$ can be expressed in terms of the inner product

$$\langle C, X\rangle = \mathrm{tr}(CX) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

for some $C \in \mathbb{S}^n$. A *semidefinite program* (SDP) therefore has the form

$$\begin{aligned} \text{minimize} \quad & \langle C, X\rangle \\ \text{subject to} \quad & \langle A_i, X\rangle = b_i \quad \text{for all } i = 1, \dots, m \\ & X \succeq 0, \end{aligned} \tag{12.1}$$

where $C, A_1, \dots, A_m \in \mathbb{S}^n$ and $b \in \mathbb{R}^m$.

An equivalent formulation, which is sometimes more convenient, is to

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & B_0 + x_1B_1 + \dots + x_kB_k \succeq 0, \end{aligned}$$

where $B_0, B_1, \dots, B_k \in \mathbb{S}^n$ and $c \in \mathbb{R}^k$. A problem of this type can be brought into the form of (12.1) by setting $X = B_0 + x_1B_1 + \dots + x_kB_k$. The entries of X then depend in a linear way on the variables $x_1, \dots, x_k$, which leads to linear relationships between the former when the latter are eliminated.

To see that linear programming is a special case of semidefinite programming, observe that $v \geqslant 0$ for a vector $v \in \mathbb{R}^n$ if and only if

$$\mathrm{diag}(v) = \begin{pmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & v_n \end{pmatrix}$$

is positive semidefinite. The linear program (2.2) can thus be written as

$$\begin{array}{ll} \text{minimize} & \langle \operatorname{diag}(c), \operatorname{diag}(x)\rangle \\ \text{subject to} & \langle \operatorname{diag}(a_i), \operatorname{diag}(x)\rangle = b_i \quad \text{for all } i = 1, \dots, m \\ & \operatorname{diag}(x) \succeq 0, \end{array}$$

where $a_i = (a_{i1}, \dots, a_{in})^T$, for $i = 1, \dots, m$, is a vector consisting of the elements of the ith row of A. This problem can be brought into the form of (12.1) by replacing the diagonal matrix $\operatorname{diag}(x)$ by a general symmetric matrix X, and adding linear constraints to ensure that the off-diagonal entries of X are zero.

SDPs can be viewed as having an infinite number of linear constraints on X, namely, $z^T X z \geqslant 0$ for all $z \in \mathbb{R}^n$. As a consequence, there are optimization problems that can be written as an SDP, but not as an LP.

There are good reasons to study semidefinite programming. It includes important classes of convex optimization problems as special cases, for example linear programming and quadratically constrained quadratic programming. In a later lecture we will see that it can be used to obtain approximate solutions to hard combinatorial and non-convex optimization problems. Moreover, SDPs can often be solved very efficiently, both in theory and in practice.

12.1 SDP Duality

The Lagrangian of (12.1) can be written as

$$L(X, \lambda, Z) = \langle C, X\rangle - \sum_{i=1}^{m} \lambda_i(\langle A_i, X\rangle - b_i) - \langle Z, X\rangle,$$

where the last term takes account of the constraint $X \succeq 0$. This works because for any $Y \in \mathbb{S}^n$, $\max_{Z \succeq 0} -\langle Z, Y\rangle$ is finite if and only if $Y \succeq 0$, so (12.1) is equivalent to the unconstrained problem $\min_{X \in \mathbb{S}^n} \max_{\lambda \in \mathbb{R}^m, Z \succeq 0} L(X, \lambda, Z)$. Then,

$$g(\lambda, Z) = \inf_{X \in \mathbb{S}^n} L(X, \lambda, Z) = \begin{cases} \lambda^T b & \text{if } C - \sum_{i=1}^{m} \lambda_i A_i - Z = 0, \\ -\infty & \text{otherwise.} \end{cases}$$

By eliminating Z, we obtain the following dual of (12.1), which is itself an SDP:

$$\begin{array}{ll} \text{maximize} & \lambda^T b \\ \text{subject to} & C - \sum_{i=1}^{m} \lambda_i A_i \succeq 0. \end{array}$$

Primal and dual SDP satisfy weak duality, because

$$\langle C, X\rangle - \lambda^T b = \langle C, X\rangle - \sum_{i=1}^{m} \lambda_i b_i = \langle C, X\rangle - \sum_{i=1}^{m} \lambda_i \langle A_i, X\rangle = \langle C - \sum_{i=1}^{m} \lambda_i A_i, X\rangle \geqslant 0,$$

where the last inequality holds because both $C - \sum_{i=1}^{m} \lambda_i A_i$ and X are positive semidefinite. If the *duality gap* $\langle C, X\rangle - \lambda^T b$ is zero, then X and λ are optimal solutions of the primal and dual, respectively. Unlike in the case of LPs, strong duality might not hold. Consider for example the SDP to

$$\begin{array}{ll} \text{minimize} & x_1 \\ \text{subject to} & \begin{pmatrix} x_1 & 1 \\ 1 & x_2 \end{pmatrix} \succeq 0. \end{array}$$

The positive semidefiniteness condition is equivalent to the constraints $x_1 \geqslant 0$, $x_2 \geqslant 0$, and $x_1 x_2 \geqslant 1$, which in turn are satisfied if and only if $x_1 > 0$ and $x_2 \geqslant 1/x_1$. The SDP thus has an optimum of 0, but the optimum is not attained. There exist SDPs whose minimum duality gap is strictly positive or even infinite. Strong duality does hold, on the other hand, if both primal and dual have a feasible solution that is *positive definite*, i.e., lies in the interior of the positive semidefinite cone.

While no algorithm is known for solving SDPs in a finite number of steps, they can be solved approximately in polynomial time, for example by a variant of the ellipsoid method. We will now briefly discuss a different class of methods that run in polynomial time in the worst case and are also very efficient in practice.

12.2 Primal-Dual Interior-Point Methods

We discuss the method for the primal and dual linear programs

$$\min\{ c^T x : Ax = b, x \geqslant 0\} \qquad \text{and} \qquad \max\{ b^T \lambda : A^T\lambda + z = c, z \geqslant 0\}.$$

The reason why these optimization problems cannot be solved using Newton's method are the inequality constraints $x \geqslant 0$ and $z \geqslant 0$. The idea behind *barrier methods* is to drop the inequality constraints and instead augment the objective by a so-called barrier function that penalizes solutions close to the boundary of, or outside, the feasible set. *Primal-dual interior-point methods* apply this idea to both the primal and the dual and try to solve them simultaneously. In the case of the above linear programs we add a *logarithmic barrier* and obtain the modified primal and dual problems

$$\min\{ c^T x - \mu \sum_{i=1}^{n} \log x_i : Ax = b\} \quad \text{and} \quad \max\{ b^T\lambda + \mu \sum_{j=1}^{m} \log z_j : A^T\lambda + z = c\},$$

for a parameter $\mu > 0$. The constraint $X \succeq 0$ in an SDP can be handled analogously using the barrier

$$-\mu \sum_{i=1}^{n} \log(\kappa_i(X)) = -\mu \log\left(\prod_{i=1}^{n} \kappa_i(X)\right) = -\mu \log(\det(X)),$$

where κ_i is the ith eigenvalue of X. This works because $X \succeq 0$ if and only if $\kappa_i \geqslant 0$ for all $i = 1, \dots, n$.

By considering the Lagrangian, it can be shown that (x, λ, z) is optimal for the modified primal and dual problems if

$$\begin{aligned} Ax = b, &\quad x \geqslant 0, \\ A^T\lambda + z = c, &\quad z \geqslant 0, \\ x_i z_i = \mu &\quad \text{for all } i = 1, \dots, n. \end{aligned} \tag{12.2}$$

Note that for $\mu = 0$, the last constraint is identical to the usual complementary slackness condition, which ensures optimality for the original problem.

A solution to the modified problems can be found using Newton's method, and provides a better and better approximation to a solution of the original problems as μ tends to zero. When μ is small, however, the modified objective is hard to optimize using Newton's method because its second-order partial derivatives vary rapidly near the boundary of the feasible set. Primal-dual interior-point methods circumvent this problem by solving a sequence of problems, decreasing μ in each iteration and starting each Newton minimization at the solution obtained in the previous round. The procedure terminates when $\mu < \epsilon$ for some desired accuracy $\epsilon > 0$. Suppose that we have found a solution (x, λ, z) that satisfies (12.2) for a given value of μ. If $\mu < \epsilon$, we stop. Otherwise we update μ, for example to $(x^T z)/(2n)$, and use Newton's method to compute a solution $(x, \lambda, z)_k + (\delta x, \delta\lambda, \delta z)$ that satisfied (12.2) for the new value of μ. Then we proceed with the next round. It can be shown that for an appropriate choice of the parameters, the method decreases the duality gap from ϵ_0 to ϵ in time $O(\sqrt{n}\log(\epsilon_0/\epsilon))$.

13 Branch and Bound

Lectures 13 through 15 will be concerned with three conceptually different approaches for optimization problems that are computationally hard: an exact method, which finds optimal solutions but has an exponential worst-case running time; heuristic methods, which need not offer guarantees regarding running time or solution quality, but often provide a good tradeoff between the two in practice; and approximation algorithms, which run in polynomial time and return solutions with a guaranteed bound on the degree of suboptimality.

Branch and bound is a general method for solving optimization problems, especially in the context of non-convex and combinatorial optimization. Suppose for concreteness that we want to

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && x \in X \end{aligned}$$

for some feasible region X. Branch and bound uses an algorithmic technique known as *divide and conquer*, which splits a problem into smaller and smaller subproblems until they become easy to solve. For the above minimization problem, it works by splitting X into $k \geqslant 2$ sets $X_1, \dots, X_k$ such that $\bigcup_{i=1,\dots,k} X_i = X$. This step is called *branching*, since its recursive application defines a tree structure, the so-called *search tree*, whose vertices are the subsets of X. Once optimal solutions have been found for the subsets $X_1, \dots, X_k$, it is easy to obtain a solution for X, because $\min_{x \in X} f(x) = \min_{i=1,\dots,k} \min_{x \in X_i} f(x)$.

Of course, branching as such doesn't make the problem any easier to solve, and for an NP-hard problem there might be no way around exploring an exponential number of vertices of the search tree. In practice we might hope, however, that we will be able to *prune* large parts of the tree that cannot contain an optimal solution. The procedure that allows us to do this is known as *bounding*. It tries to find lower and upper bounds on the optimal solution, i.e., functions ℓ and u such that for all $X' \subseteq X$, $\ell(X') \leqslant \min_{x \in X'} f(x) \leqslant u(X')$. Then, if $\ell(Y) \geqslant u(Z)$ for two sets $Y, Z \subseteq X$, Y can be discarded. A particular situations where this happens is when Y does not contain any feasible solutions, and we assume that $\ell(Y) = \infty$ by convention in this case.

For the upper bound, it suffices to store the value $U = f(x)$ of the best feasible solution $x \in X$ found so far. A good way to obtain a lower bound for a set $Y \subseteq X$ is by letting $\ell(Y) = \min_{x \in Y'} f(x)$ for some set $Y' \supseteq Y$ for which minimization of f is computationally tractable. It is easy to see that this indeed provides a lower bound. Moreover, if minimization over Y' yields a solution $x \in Y$, then this solution is optimal for Y. The branch and bound method stores U and a list L of active sets $Y \subseteq X$ for which no optimal solution has been found so far, corresponding to vertices in the search tree that still need to be explored, along with their associated lower bounds. It then proceeds as follows:

1. Initialize: set $U = \infty$, $L = \{X\}$.
2. Branch: pick a set $Y \in L$, remove it from L, and split it into $k \geqslant 2$ sets $Y_1, \dots, Y_k$.
3. Bound: for $i = 1, \dots, k$, compute $\ell(Y_i)$. If this yields $x \in X$ such that $\ell(Y_i) = f(x) < U$, then set U to $f(x)$. If $\ell(Y_i) < U$, but no $x \in X$ as above is found, then add Y_i to L.
4. If $L = \emptyset$, then stop. The optimum objective value is U. Otherwise go to Step 2.

To apply the method in a concrete setting, we need to specify how a set Y to branch on is chosen and how it is split into smaller sets, and how lower bounds are computed. These decisions are of course critical for the practical performance of the procedure.

13.1 Dakin's Method

Dakin's method applies the branch and bound idea to integer programs. An obvious way to obtain lower bounds in this case is by solving the LP relaxation, i.e., the linear program obtained by dropping the integrality constraints.

Assume that we are branching on a set $Y \in L$, and that the LP relaxation corresponding to Y has optimal solution y. If $y \in Y$, then y is optimal for Y. Otherwise, there is some i such that y_i is not integral, and we can split Y into two sets $Y^1 = \{x \in Y : x_i \leqslant \lfloor y_i \rfloor\}$ and $Y^2 = \{x \in Y : x_i \geqslant \lceil y_i \rceil\}$. Note that $Y^1 \cup Y^2 = Y$, as desired. Moreover, this branching rule forces the solution away from its current value $y \notin Y$. While this does not guarantee that y_i becomes integral in the next step, and may even force another variable away from its integral value, it works remarkably well in practice. It is worth noting that we do not have to start from scratch when solving the LP relaxation for Y_i: it was obtained by adding a constraint to an LP that is already solved, and the dual simplex method often finds a solution satisfying the additional constraint very quickly. In order to minimize the number of solved LPs that have to be stored to implement this approach, it makes sense to branch on a set obtained in the previous step whenever possible, i.e., to traverse the search tree in a depth-first manner.

Example 13.1. Assume that we want to

$$\begin{aligned} \text{minimize} \quad & x_1 - 2x_2 \\ \text{subject to} \quad & -4x_1 + 6x_2 \leqslant 9 \\ & x_1 + x_2 \leqslant 4 \\ & x_1 \geqslant 0, x_2 \geqslant 0 \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

An illustration is shown in Figure 13.1. Let $f(x) = x_1 - 2x_2$, and $X = \mathbb{Z}^2 \cap \tilde{X}$ where

$$\tilde{X} = \{\, x \in \mathbb{R}^2 : -4x_1 + 6x_2 \leqslant 9, x_1 + x_2 \leqslant 4, x_1 \geqslant 0, x_2 \geqslant 0 \,\}.$$

We start with $U = \infty$ and $L = \{X\}$. By solving the LP relaxation for X, we find that $\ell(X) = \min_{x \in \tilde{X}} f(x) = f(x^0) = -7/2$ for $x^0 = (3/2, 5/2)$. Set X is the only candidate for

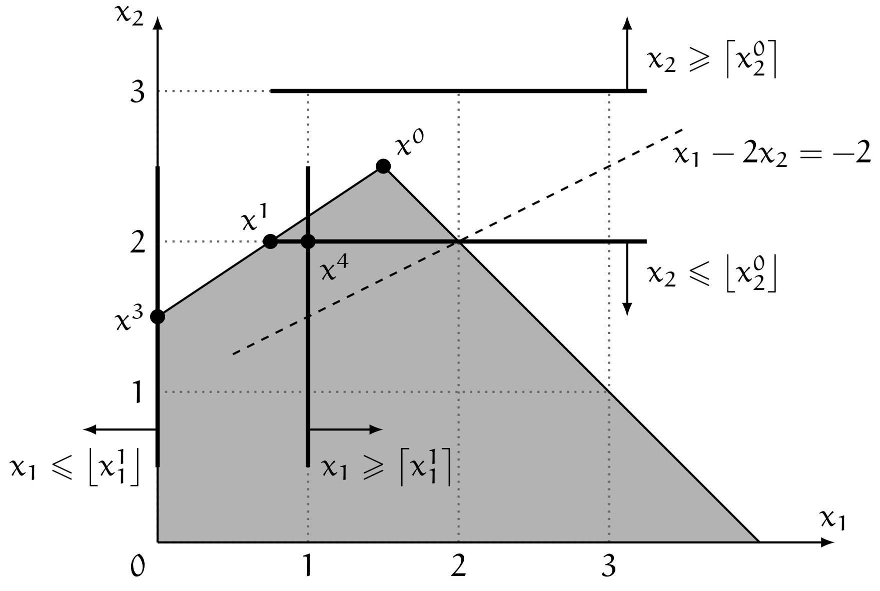
x_2
$x_2 \geqslant \lceil x_2^0 \rceil$
$x_1 - 2x_2 = -2$
$x_2 \leqslant \lfloor x_2^0 \rfloor$
x^0
x^1
x^4
x^3
$x_1 \leqslant \lfloor x_1^1 \rfloor$
$x_1 \geqslant \lceil x_1^1 \rceil$
x_1
0
1
2
3

Figure 13.1: Illustration of Dakin's method, applied to the IP of Example 13.1

branching and can for example be split into $X^1 = \mathbb{Z}^2 \cap \tilde{X}^1$ and $X^2 = \mathbb{Z}^2 \cap \tilde{X}^2$, where

$$\tilde{X}^1 = \{x \in \tilde{X} : x_2 \leqslant 2\} \quad \text{and} \quad \tilde{X}^2 = \{x \in \tilde{X} : x_2 \geqslant 3\}.$$

We then bound X^1 and X^2 by solving the corresponding LP relaxations, and obtain $\ell(X^1) = \min_{x \in \tilde{X}^1} f(x) = f(x^1) = -13/4$ for $x^1 = (3/4, 2)$, and $\tilde{X}^2 = \emptyset$. We thus set $L = \{X^1\}$. We now branch by splitting X^1 into $X^3 = \mathbb{Z}^2 \cap \tilde{X}^3$ and $X^4 = \mathbb{Z}^2 \cap \tilde{X}^4$, where

$$\tilde{X}^3 = \{x \in \tilde{X}^1 : x_1 \leqslant 0\} \quad \text{and} \quad \tilde{X}^4 = \{x \in \tilde{X}^1 : x_1 \geqslant 1\},$$

and bounding X^3 and X^4 to obtain $\ell(X^3) = \min_{x \in \tilde{X}^3} f(x) = f(x^3) = -3$ for $x^3 = (0, 3/2)$ and $\ell(X^4) = \min_{x \in \tilde{X}^4} f(x) = f(x^4) = -3$ for $x^4 = (1, 2)$. Since $x^4 \in X$, we can set $U = f(x^4) = -3$. Then, $\ell(X^3) \geqslant U$, so we can discard X^3 and are done.

13.2 The Traveling Salesman Problem

Recall that in the traveling salesman problem (TSP) we are given a matrix $A \in \mathbb{N}^{n \times n}$ and are looking for a permutation $\sigma \in S_n$ that minimizes $a_{\sigma(n)\sigma(1)} + \sum_{i=1}^{n-1} a_{\sigma(i)\sigma(i+1)}$. Matrix entry a_{ij} can be interpreted as a cost associated with edge $(i, j) \in E$ of a graph $G = (V, E)$, and we are then trying to find a *tour*, i.e., a cycle in G that visits every vertex exactly once, of minimum overall cost. We have seen that the TSP is NP-hard, but we could try to encode it as an integer program and solve it using branch and bound. Consider variables

$$x_{ij} \in \{0, 1\} \quad \text{for } i, j = 1, \ldots, n, \tag{13.1}$$

encoding whether the tour traverses edge (i, j). There are various ways to ensure that these variables indeed encode a tour, i.e., that $x_{ij} = 1$ if and only if $\sigma(n) = i$ and

$\sigma(1) = j$, or $\sigma(k) = i$ and $\sigma(k+1) = j$ for some $k \in \{1, \dots, n-1\}$. Of course, there has to be exactly one edge entering and one edge leaving every vertex, i.e.,

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1 \quad \text{for } j = 1, \dots, n, \\ \sum_{j=1}^{n} x_{ij} &= 1 \quad \text{for } i = 1, \dots, n. \end{aligned} \tag{13.2}$$

The so-called cut-set formulation additionally requires that there are at least two edges across every cut $S \subseteq V$, whereas the subtour elimination formulation makes sure that no set $S \subsetneq V$ contains more than $|S|-1$ edges. The problem with both of these formulations is of course that they require an exponential number of constraints, one for each set $S \subseteq V$.

A polynomial formulation can be obtained by introducing, for $i = 1, \dots, n$, an auxiliary variable $t_i \in \{0, \dots, n-1\}$ indicating the position of vertex i in the tour. If $x_{ij} = 1$, it holds that $t_j = t_i + 1$. If $x_{ij} = 0$, on the other hand, then $t_j \geqslant t_i - (n-1)$. This can be written more succinctly as

$$t_j \geqslant t_i + 1 - n(1 - x_{ij}) \quad \text{for all } i \geqslant 1,\, j \geqslant 2,\, i \neq j. \tag{13.3}$$

Since values satisfying (13.3) exist for every valid tour, adding this constraint does not affect solutions corresponding to valid tours. On the other hand, it suffices to rule out subtours, i.e., cycles of length less than $|V|$. To see this, consider a solution that satisfies (13.3), and assume for contradiction that it consists of two or more subtours. Summing the constraints over the edges in a subtour that does not contain vertex 1 leads to the condition that $0 \geqslant k$, where k is the number of edges in the subtour, a contradiction.

A minimum cost tour can thus be found by minimizing $\sum_{i,j} x_{ij} a_{ij}$ subject to (13.1), (13.2), and (13.3). This integer program has a polynomial number of variables and constraints and can be solved using Dakin's method, which bounds the optimum by relaxing the integrality constraints (13.1). There are, however, other relaxations that are specific to the TSP and can provide better bounds.

Observe, for example, that the integer program obtained by relaxing the subtour elimination constraints (13.3) is an instance of the assignment problem (9.1). It can be solved efficiently in practice using the network simplex method, which yields a solution consisting of one or more subtours. If there is more than one subtour, then taking the set $\{e_1, \dots, e_k\} \subseteq E$ of edges of one or more of the subtours and disallowing each of them in turn splits the feasible set Y into $Y_1, \dots, Y_k$, where $Y_i = \{x \in Y : x_{uv} = 0, e_i = (u, v)\}$. Clearly, the optimal TSP tour cannot contain all edges of a subtour, so it must be contained in one of the sets Y_i. Moreover, adding a constraint of the form $x_{ij} = 0$ is equivalent to setting the corresponding cost a_{ij} to a large enough value, so the new problem will still be an instance of the assignment problem. Note that none of the sets Y_i contains the optimal solution of the current relaxation, so $\ell(Y_i) \geqslant \ell(Y)$ for all i, and $\ell(Y_i) > \ell(Y)$ if the optimal solution of the current relaxation was unique.

14 Heuristic Algorithms

For all we know, a complete exploration of the search tree, either explicitly or implicitly, might be required to guarantee that an optimal solution is found. When this is impractical, heuristic methods can be used to find a satisfactory, but possibly suboptimal, solution. Heuristics sacrifice solution quality in order to gain computational performance or conceptual simplicity.

14.1 Basic Heuristics for the TSP

A straightforward way of constructing a TSP tour is by starting from an arbitrary vertex, traversing a minimum cost edge to an unvisited vertex until all vertices have been visited, and returning to the initial vertex to complete the tour. This greedy algorithm is known as the *nearest neighbor heuristic* and has an asymptotic complexity of $O(n^2)$, where n is the number of vertices. Intuitively it will work well most of the time, but will sometimes have to add an edge with very high cost because all vertices connected to the current one by an edge with low cost have already been visited.

Instead of adding a minimum cost edge among those adjacent to the current vertex, we could add an edge that has minimum cost overall, while ensuring that the resulting set of edges will form a tour. This corresponds to ordering the edges by increasing cost and adding them to the tour in that order, skipping edges that would lead to a vertex with degree more than two or a cycle of length less than n. This so-called *savings heuristic* has complexity $O(n^2 \log n)$, which is the complexity of sorting a set with n^2 elements.

Another intuitive approach is to start with a subtour, i.e., a tour on a subset of the set of vertices, and extending it with additional vertices. Heuristics following this general approach are known as *insertion heuristics*. A particular heuristics of this type has to specify a way of choosing (i) the initial subtour, (ii) the vertex to be inserted, and (iii) the way the new vertex is inserted into the subtour. Obvious choices for the initial subtour are cycles of length two or three. The *cheapest insertion heuristic* then chooses a vertex, and a place to insert the vertex into the subtour, in order to minimize the overall length of the resulting subtour. The *farthest insertion heuristic*, on the other hand, inserts a vertex whose minimum distance to any vertex in the current subtour is maximal. The idea behind the latter strategy is to fix the overall layout of the tour as soon as possible.

Of course, optimization does not need to end once we have constructed a tour. Rather, we could try to make a small modification to the tour in order to reduce its cost, and repeat this procedure as long as we can find such an improving modification. An algorithm that follows this general procedure is known as *local search algorithm*, because it makes local modifications to a solution to obtain a new solution with a better objective value. A tour created using the nearest neighbor heuristic, for example, will usually contain a few edges with very high cost, so we would be interested in local modifications that eliminate these edges from the tour.

14.2 Local Search

Assume that we want to

$$\begin{aligned} \text{minimize} \quad & c(x) \\ \text{subject to} \quad & x \in X, \end{aligned}$$

and that for any feasible solution $x \in X$, the cost $c(x)$ and a *neighborhood* $N(x) \subseteq X$ can be computed efficiently. Local search then proceeds as follows:

1. Find an initial feasible solution $x \in X$.
2. Find a solution $y \in N(x)$ such that $c(y) < c(x)$.
3. If there is no such solution, then stop and return x; otherwise set the current solution x to y and return to Step 2.

The solution returned by this procedure is a *local optimum*, in the sense that its cost is no larger than that of any solution in its neighborhood. It need not be globally optimal, as there might be a solution outside the neighborhood with strictly smaller cost.

Any of the basic tour construction heuristics can be used to find an initial feasible solution in Step 1, and the whole procedure can also be run several times with different initial solutions. Step 2 requires a choice if more than one neighboring solution provides a decrease in cost. Natural options include the first such solution to be found, or the solution providing the largest decrease.

Most importantly, however, any implementation of a local search method must specify the neighborhood function N. A natural neighborhood for the TSP is the k-OPT neighborhood. Here, the neighbors of a given tour are obtained by removing any set of k edges, for some $k \geqslant 2$, and reconnecting the k paths thus obtained to a tour by adding k edges. Viewing tours as permutations, k-OPT cuts a permutation into k segments and reverses and swaps these segments in a arbitrary way. An illustration for $k = 2$ and $k = 3$ is shown in Figure 14.1.

The choice of k provides a tradeoff between solution quality and speed: the k-OPT neighborhood of a solution contains its ℓ-OPT neighborhood if $k \geqslant \ell$, so the quality of the solution increases with k; the same is also true for the complexity of the method, because the k-OPT neighborhood of a tour of length n has size $O(n^k)$ and computing the change in cost between two neighboring tours requires $O(k)$ operations. Empirical evidence suggests that 3-OPT often performs better than 2-OPT, while there is little gain in taking $k > 3$.

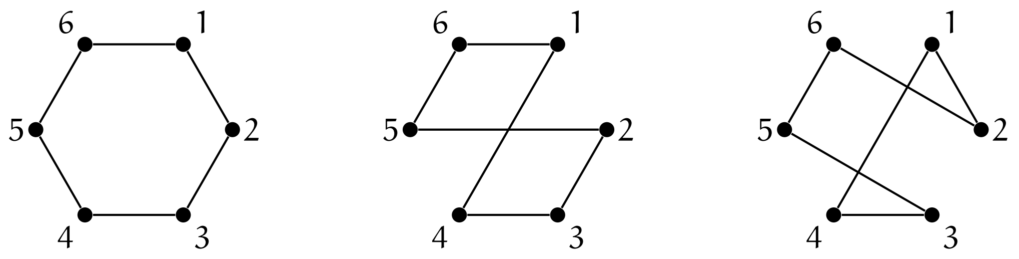

Figure 14.1: A TSP tour (left) and neighboring tours under the 2-OPT (middle) and 3-OPT neighborhoods (right). The tours respectively correspond to the permutations 123456, 143256, and 126534.

Note that the simplex method for linear programming can also be viewed as a local search algorithm, where two basic feasible solutions are neighbors if their bases differ by exactly one element. We have seen that in this case every local optimum is also a global optimum, so that the simplex method yields a globally optimal solution.

In general, however, local search might get stuck in a local optimum and fail to find a global one. Consider for example the TSP instance given by the cost matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & 4 & 4 \\ 4 & 0 & 1 & 0 & 4 \\ 4 & 4 & 0 & 1 & 0 \\ 0 & 4 & 4 & 0 & 1 \\ 1 & 0 & 4 & 4 & 0 \end{pmatrix}.$$

There are $4! = 24$ TSP tours, and

$$\begin{array}{llll} c(12345) = 5, & c(13245) = 6, & c(14235) = 10, & c(15234) = 6, \\ c(12354) = 6, & c(13254) = 12, & c(14253) = 20, & c(15243) = 12, \\ c(12435) = 6, & c(13425) = 10, & c(14325) = 17, & c(15324) = 12, \\ c(12453) = 10, & c(13452) = 6, & c(14352) = 9, & c(15342) = 17, \\ c(12534) = 10, & c(13524) = 0, & c(14523) = 10, & c(15423) = 17, \\ c(12543) = 17, & c(13542) = 12, & c(14532) = 17, & c(15432) = 20. \end{array}$$

It is easily verified that the tour 12345 is a local optimum under the 2-OPT neighborhood, while the global optimum is the tour 13524.

14.3 Simulated Annealing

To prevent local search methods from getting stuck in a local optimum, one could allow transitions to a neighbor even if it has higher cost, with the hope that solutions with lower cost will be reachable from there. *Simulated annealing* implements this idea using an analogy to the process of annealing in metallurgy, in which a metal is heated and then cooled gradually in order to bring it to a low-energy state that comes with better physical properties.

In each iteration, simulated annealing considers a neighbor y of the current solution x and moves to the new solution with probability

$$p_{xy} = \min\left(1, \exp\left(-\frac{c(y) - c(x)}{T}\right)\right),$$

where $T \geqslant 0$ is a parameter, the *temperature*, that can vary over time. With the remaining probability the solution stays the same. When T is large, the method allows transitions even when $c(y)$ exceeds $c(x)$ by a certain amount. As T approaches zero, so does the probability of moving to a solution with larger cost.

It can further be shown that with a suitable *cooling schedule* that decreases T sufficiently slowly from iteration to iteration, the probability of reaching an optimal solution after t iterations tends to 1 as t tends to infinity. To motivate this claim, consider the special case where every solution has k neighbors and a neighbor of the current solution is chosen uniformly at random. The behavior of the algorithm can then be modeled as a *Markov chain* with transition probabilities

$$P_{xy} = \begin{cases} p_{xy}/k & \text{if } y \in N(x), \\ 1 - \sum_{z \in N(x)} p_{xz}/k & \text{if } y = x, \\ 0 & \text{otherwise.} \end{cases}$$

This Markov chain has a unique *stationary distribution* π, i.e., a distribution over X such that for all $x \in X$, $\pi_x = \sum_{y \in X} \pi_y P_{yx}$. In addition it can be shown that π must satisfy the *detailed balance* condition that $\pi_x P_{xy} = \pi_y P_{yx}$ for every pair of solutions $x, y \in X$. In fact, detailed balance is not only necessary but also sufficient for stationary, because it implies that $\sum_{x \in X} \pi_x P_{xy} = \sum_{x \in X} \pi_y P_{yx} = \pi_y \sum_{x \in X} P_{yx} = \pi_y$. It is not hard to show that π with

$$\pi_x = \frac{e^{-c(x)/T}}{\sum_{z \in X} e^{-c(z)/T}}$$

for every $x \in X$ is a distribution and satisfies detailed balance, and must therefore be the stationary distribution. Letting $Y \subseteq X$ be the set of solutions with minimum cost and $\pi_Y = \sum_{x \in Y} \pi_x$, we conclude that $\pi_Y/(1 - \pi_Y) \to \infty$ as $T \to 0$.

The idea now is to decrease T slowly enough for the Markov chain to be able to reach its stationary distribution. A common cooling schedule is to set $T = c/\log t$ in iteration t, for some constant c.

15 Approximation Algorithms

An optimization problem can be represented by a function $o : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$, where $o(x,y)$ is the objective value of output $y \in \{0,1\}^*$ for input $x \in \{0,1\}^*$. Restricting our attention to minimization problems, a function $f : \{0,1\}^* \to \{0,1\}^*$ provides an optimal solution to optimization problem o if for every input $x \in \{0,1\}^*$, $o(x, f(x)) = \min\{o(x,y) : y \in \{0,1\}^*\}$. When f is hard to compute, one might instead try to find a solution that is as good as possible. The most common notion of quality in this context is a worst-case multiplicative one: a function $g : \{0,1\}^* \to \{0,1\}^*$ will be called an α-approximation for o, for some $\alpha \geqslant 1$, if for all $x \in P$, $o(x, g(x)) \leqslant \alpha o(x, f(x))$. In what follows we will be interested in algorithms that compute function g in polynomial time, and will refer to such an algorithm as a (polynomial-time) α-approximation algorithm for optimization problem o.

While in principle α could depend on the size of the input, we will only consider problems in the complexity class APX, which have an α-approximation algorithm for some constant α. The class PTAS $\subseteq$ APX, for polynomial-time approximation scheme, contains problems that possess an $(1+\epsilon)$-approximation algorithm for any $\epsilon > 0$, where the (polynomial) running time can depend on ϵ in an arbitrary way. APX-hardness is established by a reduction corresponding to the smaller class PTAS, and implies that a problem can be approximated in polynomial time up to some, but not every, constant factor.

15.1 The Max-Cut Problem

Given an undirected graph $G = (V, E)$, the max-cut problem asks for a cut of G that maximizes the number of edges crossing from one side to the other, i.e., a set $S \subseteq V$ such that $|E \cap (S \times (V \setminus S))|$ is as large as possible. The max-cut problem is NP-complete and thus cannot be solved exactly in polynomial time unless $P = NP$.

On the other hand, a simple greedy algorithm provides a $1/2$-approximation. First of all, observe that every graph has a cut of size at least $|E|/2$. For this, consider a random cut $S \subseteq V$ such that for each $v \in V$, $v \in S$ independently with probability $1/2$. Then, the number of edges across the cut is a random variable $Q = \sum_{\{i,j\} \in E} \mathbb{I}[Q_i \neq Q_j]$, where $\mathbb{I}$ denotes the indicator function on binary events and for each $i \in V$, Q_i is a Bernoulli random variable with parameter $1/2$. Thus,

$$\mathbb{E}[Q] = \mathbb{E}\Big[\sum_{\{i,j\} \in E} \mathbb{I}[Q_i \neq Q_j]\Big] = \sum_{\{i,j\} \in E} \mathbb{E}\big[\mathbb{I}[Q_i \neq Q_j]\big] = \sum_{\{i,j\} \in E} \mathbb{P}[Q_i \neq Q_j]\big] = |E|/2,$$

where the second equality holds by linearity of expectation, so there must exist a cut of size at least $|E|/2$. This probabilistic argument can be de-randomized efficiently using the *method of conditional probabilities*. To this end, each vertex is considered in turn, replacing the random decision whether the vertex is included in S by a deterministic one. The goal is to ensure that the conditional probability of obtaining a cut of size at least $|E|/2$, assuming that the remaining decisions are taken randomly, remains positive. While the conditional probability itself may not be easy to determine, it suffices in each step to maximize the conditional expectation of the random variable Q, because the maximum is guaranteed to be at least $|E|/2$. Let $U \subseteq V$ be the set of vertices for which a decision has already been made, and let $S \subseteq U$ be the resulting cut. The conditional expectation of Q given this choice is

$$\Big|E \cap \big(S \times (U \setminus S)\big)\Big| + \frac{1}{2}\Big|E \cap \big(V \times (V \setminus U)\big)\Big|,$$

and it can be maximized by maximizing the first term. Since the size of a cut can be at most $|E|$, a greedy algorithm that considers each vertex in turn and adds it to either S or $U \setminus S$ in order to maximize the first term provides a $1/2$-approximation.

A better approximation can be obtained using semidefinite programming.

Theorem 15.1 (Goemans and Williamson, 1995). *There exists a 0.87856-approximation algorithm for the max-cut problem.*

Proof sketch. The max-cut problem can be written as the following integer quadratic program:

$$\begin{aligned} &\text{maximize} && \sum_{\{i,j\}\in E} \frac{1 - x_i x_j}{2} \\ &\text{subject to} && x_i \in \{-1, 1\} \quad \text{for all } i \in V. \end{aligned} \tag{15.1}$$

The intuition behind the objective is that $x_i = 1$ if $i \in S$, and edge $\{i, j\} \in E$ contributes 1 to the sum if and only if $|\{i, j\} \cap S| = 1$.

Since the max-cut problem is NP-complete, an optimal solution of (15.1) cannot be found in polynomial time unless P = NP. Note, however, that

$$\sum_{\{i,j\}\in E} \frac{1 - x_i x_j}{2} = \frac{|E|}{2} - \frac{1}{4} x^T C x = \frac{|E|}{2} - \frac{1}{4}\langle C, xx^T\rangle,$$

where $C \in \{0, 1\}^{|V|\times|V|}$ with $C_{ij} = 1$ if $\{i, j\} \in E$ and $C_{ij} = 0$ otherwise. Moreover, xx^T is a positive semidefinite matrix. We can thus relax the constraints, and obtain an upper bound on the optimal solution of (15.1), by replacing xx^T by a general positive semidefinite matrix X with $X_{ii} = 1$ for all $i \in V$. We arrive at the following optimization problem, which is an SDP:

$$\begin{aligned} &\text{maximize} && \frac{|E|}{2} - \frac{1}{4}\langle C, X\rangle \\ &\text{subject to} && X_{ii} = 1 \quad \text{for all } i \in V \\ &&& X \succeq 0. \end{aligned}$$

Since the constraints have been relaxed, an optimal solution of this SDP need not be feasible for (15.1). Intuitively, a feasible solution of (15.1) corresponds to a set of $|V|$

unit vectors in $\mathbb{R}^1$, while a feasible solution of the relaxed problem corresponds to a set of $|V|$ unit vectors in $\mathbb{R}^n$. The latter can be "rounded" to the former, however, by randomly picking a hyperplane in $\mathbb{R}^n$ that passes through the origin and mapping each unit vector in $\mathbb{R}^n$ to -1 or 1 depending on its relative position to this hyperplane. Surprisingly, this changes the objective value by a factor of at most 0.87856, and thus yields a feasible solution of (15.1) that is within the same factor of an optimal one. All the necessary steps can be carried out in polynomial time, and the method can also be de-randomized without affecting the approximation factor. $\square$

15.2 Hardness of Approximation

An obvious question is whether the bound of Theorem 15.1 is optimal, or whether it can be improved further. NP-hardness of a problem establishes that it is hard to distinguish instances with a certain optimum, like the size of a maximum cut in the case of the max-cut problem, from instances whose optimum is smaller or larger. A possible approach for showing that a problem does not admit an α-approximation algorithm for some $\alpha < 1$ would be to create a gap between positive and negative instances, and show that it is hard to distinguish instances with a large optimum from instances with a small optimum. The problem with this approach is that our characterization of the class NP is very fragile. Cook's proof of Theorem 5.1 uses a class of Boolean formulae to encode the computations of a Turing machine, which in turn are very sensitive to small changes. And indeed, if one were to inspect the formulae more closely, one would see that it is very easy for each of them to find an assignment that satisfies every clause except one. What is needed to show hardness of approximation is a more robust model of NP. Such a model is provided by *probabilistically checkable proofs* (PCPs).

PCPs can be obtained using a probabilistic modification of the definition of NP. As before, we are given access to an input x and to a certificate y which acts as proof that x satisfies a certain property. Instead of a deterministic Turing machine as in the case of NP, however, we want to use a *probabilistic verifier* V to check the proof. The fact that V is probabilistic can be modeled by assuming that besides x and y it takes an additional input r, which is a string of random bits, and then performs a deterministic computation based on x, y, and r. For fixed x and y, we say that V accepts x and y with probability p if it accepts with this probability for a uniformly distributed random string r. Note that so far we have only made the verifier more powerful, by giving it access to a random string. To be able to say something interesting about its relationship to the class NP, we therefore have to restrict what it can do with its inputs. We call a verifier V $(r(n), q(n))$-restricted, for two functions $r : \mathbb{Z} \to \mathbb{Z}$ and $q : \mathbb{Z} \to \mathbb{Z}$, if for every input of length n and every certificate y, it queries at most $q(n)$ bits of y and uses at most $r(n)$ random bits. A problem L then is in the class $\mathrm{PCP}[r(n), q(n)]$ if there exists an $(r(n), q(n))$-restricted verifier with the following properties: if $x \in L$, then there exists a certificate y such that V accepts x and y with probability 1; if $x \notin L$, then for every certificate y, V accepts x and y with probability at most $1/2$.

It is easy to see, for example, that $\mathrm{PCP}[O(\log n), O(\log n)] \subseteq \mathrm{NP}$: a verifier that uses a logarithmic number of random bits can easily be de-randomized by considering all possible random strings of logarithmic length. The PCP theorem states a surprising converse: every problem in NP has a probabilistic verifier that uses a logarithmic number of random bits and examines a *constant* number of bits of the certificate.

THEOREM 15.2 (Arora et al., 1998). $\mathrm{NP} = \mathrm{PCP}[O(\log n), O(1)]$

The PCP theorem can be used to show that max-3SAT, the problem of computing the maximum number of simultaneously satisfiable clauses of an instance of SAT with three literals per clause, is APX-hard.

THEOREM 15.3 (Arora et al., 1998). *There exists* $\epsilon > 0$ *such that there is no* $(1-\epsilon)$-*approximation algorithm for max-3SAT unless* $\mathrm{P} = \mathrm{NP}$.

Proof sketch. By the PCP Theorem, any NP-complete problem has a $(c \log n, q)$-restricted verifier V for some constants c and q. For a particular random string r, V chooses q positions $i_1^r, \dots, i_q^r$ of the certificate y and a function $f_x^r : \{0,1\}^q \to \{0,1\}$, and accepts if and only if $f_x^r(y_{i_1^r}, \dots, y_{i_q^r}) = 1$.

The proof works by constructing, for each $x \in L$, a Boolean formula ϕ_x with variables $v_1, \dots, v_{|y|}$ and clauses representing the constraint $f_x^r(v_{i_1^r}, \dots, v_{i_q^r}) = 1$ for every possible random string r. Formula ϕ_x has a number m of clauses that is polynomial in $|x|$, and the following can be shown to hold for $\epsilon = \frac{1}{2}\frac{1}{q2^q}$:

if $x \in L$, then ϕ_x is satisfiable;

if $x \notin L$, then no assignment satisfies more than $(1-\epsilon)m$ clauses of ϕ_x.

By distinguishing between the case where the number of satisfiable clauses is m and the case where the number of satisfiable clauses is $(1-\epsilon)m$, we can thus distinguish between the cases $x \in L$ and $x \notin L$, thereby solving an NP-complete problem. The existence of a polynomial-time algorithm for the former problem would thus imply that $\mathrm{P} = \mathrm{NP}$. $\square$

Since there is a PTAS reduction from max-3SAT to max-cut, the latter is APX-hard as well. Improved PCP characterizations of NP have lead to better bounds for various problems, which in some cases are tight: max-3SAT, for example, is NP-hard to approximate to a factor of $7/8 + \epsilon$ for any $\epsilon > 0$, and a factor of $7/8$ can be achieved easily by choosing a value for each variable uniformly at random. For max-cut the same techniques yield an upper bound of $16/17 + \epsilon$, which does not match the lower bound of Theorem 15.1. The apparent difficulty in improving the upper bound seems to be related to the fact that constraints in the max-cut problem involve two variables, compared to three in the case of max-3SAT, and that the known PCP characterizations corresponding to the two-variable case are weaker. The *unique games conjecture* postulates the existence of a PCP construction for the two-variable case that would imply an upper bound for max-cut that matches the bound of Theorem 15.1.

16 Non-Cooperative Games

The second part of the course will be concerned with situations in which multiple self-interested entities, or *agents*, operate in the same environment. *Game theory* provides mathematical models, so-called games, for studying these types of situations. We focus for now on *non-cooperative games*, where agents independently optimize different objectives and outcomes must be self-enforcing. In a later lecture we also consider *cooperative games*, which focus on conditions under which cooperation among subsets of the agents can be sustained.

16.1 Games and Solutions

The central object of study in non-cooperative game theory are normal-form games. A *normal-form game* is a tuple $\Gamma = (N, (A_i)_{i \in N}, (p_i)_{i \in N})$ where N is a finite set of *players*, and for each player $i \in N$, A_i is a non-empty and finite set of *actions* available to i and $p_i : (\times_{i \in N} A_i) \to \mathbb{R}$ is a function mapping each action profile, i.e., each combination of actions, to a real-valued *payoff* for i. Unless noted otherwise, the results we consider are invariant under positive affine transformations, and payoffs and their relative intensities will not be comparable across players.

More complicated games where players move sequentially and base their decisions on their and others' earlier moves can also be represented as normal-form games, by encoding every possible course of action in the former by an action of the latter. It should be noted, however, that this generally leads to a large increase in the number of actions.

We henceforth restrict our attention to two-player games, but note that most concepts and results extend in a straightforward way to games with more than two players. A two-player game with m actions for player 1 and n actions for player 2 can be represented by a pair of matrices $P, Q \in \mathbb{R}^{m \times n}$, where p_{ij} and q_{ij} are the payoffs of players 1 and 2 when player 1 plays action i and player 2 plays action j. Two-player games are therefore sometimes referred to as bimatrix games, and players 1 and 2 as the row and column player, respectively.

Assume that players can choose their actions randomly and denote the set of possible *strategies* of the two players by X and Y, respectively, i.e., $X = \{x \in \mathbb{R}^m_{\geqslant 0} : \sum_{i=1}^m x_i = 1\}$ and $Y = \{y \in \mathbb{R}^n_{\geqslant 0} : \sum_{i=1}^n y_i = 1\}$. A *pure strategy* is a strategy that chooses some action with probability one, and we make no distinction between pure strategies and the corresponding actions. A profile $(x, y) \in X \times Y$ of strategies induces a lottery over outcomes, and we write $p(x, y) = x^T P y$ and $q(x, y) = x^T Q y$ for the expected payoff of the two players in this lottery.

Consider for example the well-known prisoner's dilemma, involving two suspects

$$\begin{array}{c|cc|} & S & T \\ \hline S & (2,2) & (0,3) \\ T & (3,0) & (1,1) \\ \hline \end{array}$$

Figure 16.1: Representation of the prisoner's dilemma as a normal-form game. The matrices P and Q are displayed as a single matrix with entries (p_{ij}, q_{ij}), and players 1 and 2 respectively choose a row and a column of this matrix. Action S corresponds to remaining silent, action T to testifying.

$$\begin{array}{c|cc|} & C & D \\ \hline C & (2,2) & (1,3) \\ D & (3,1) & (0,0) \\ \hline \end{array}$$

Figure 16.2: The game of chicken, where players can chicken out or dare

accused of a crime who are being interrogated separately. If both remain silent, they walk free after spending a few weeks in pretrial detention. If one of them testifies against the other and the other remains silent, the former is released immediately while the latter is sentenced to ten years in jail. If both suspects testify, each of them receives a five-year sentence. A representation of this situation as a two-player normal-form game is shown in Figure 16.1.

It is easy to see what the players in this game should do, because action T yields a strictly larger payoff than action S for *every* action of the respective other player. More generally, for two strategies $x, x' \in X$ of the row player, x is said to *(strictly) dominate* x' if for every strategy $y \in Y$ of the column player, $p(x, y) > p(x', y)$. Dominance for the column player is defined analogously. Strategy profile (T, T) in the prisoner's dilemma is what is called a *dominant strategy equilibrium*, a profile of strategies that dominate every other strategy of the respective player. The source of the dilemma is that outcome resulting from (T, T) is strictly worse for both players than the outcome resulting from (S, S). More generally, an outcome that is weakly preferred to another outcome by all players, and strictly preferred by at least one player is said to *Pareto dominate* that outcome. An outcome that is Pareto dominated is clearly undesirable.

In the absence of dominant strategies, it is less obvious how players should proceed. Consider for example the game of chicken illustrated in Figure 16.2. This game has its origins in a situation where two cars drive towards each other on a collision course. Unless one of the drivers yields, both may die in a crash. If one of them yields while the other goes straight, however, the former will be called a "chicken," or coward. It is easily verified that this game does not have any dominant strategies.

The most cautious choice in a situation like this would be to ignore that the other player is self-interested and choose a strategy that maximizes the payoff in the worst case, taken over all of the other player's strategies. A strategy of this type is known as a *maximin strategy*, and the payoff thus obtained as the player's *security level.* It is easy to see that it suffices to maximize the minimum payoff over all *pure* strategies of the other player, i.e., to choose x such that $\min_{j\in\{1,\ldots,n\}} \sum_{i=1}^{m} x_i p_{ij}$ is as large as possible. Maximization of this minimum can be achieved by maximizing a lower bound that holds for all $j = 1, \ldots, n$, so a maximin strategy and the security level for the row player can be found as a solution of the following linear program with variables $v \in \mathbb{R}$ and $x \in \mathbb{R}^m$:

$$\begin{aligned} \text{maximize} \quad & v \\ \text{subject to} \quad & \sum_{i=1}^{m} x_i p_{ij} \geqslant v \quad \text{for } j = 1, \ldots, n \\ & \sum_{i=1}^{m} x_i = 1 \\ & x \geqslant 0. \end{aligned} \tag{16.1}$$

The unique maximin strategy in the game of chicken is to yield, for a security level of 1. This also illustrates that a maximin strategy need not be optimal: assuming that the other player yields, the best response is in fact to go straight. Formally, strategy $x \in X$ of the row player is a *best response* to strategy $y \in Y$ of the column player if for all $x' \in X$, $p(x, y) \geqslant p(x', y)$. The concept of a best response for the column player is defined analogously.

16.2 The Minimax Theorem

In the special case when the interests of the two players are diametrically opposed, maximin strategies turn out to be optimal in a very strong sense. A two-player game is called *zero-sum game* if $q_{ij} = -p_{ij}$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$. A game of this type is sometimes called a matrix game, because it can be represented just by the matrix P containing the payoffs of the row player. Assuming invariance of utilities under positive affine transformations, results for zero-sum games in fact apply to the larger class of *constant-sum* games, in which the payoffs of the two players always sum up to the same constant. For games with more than two players, these properties are far less interesting, as one can always add an extra player who "absorbs" the payoffs of the others.

Theorem 16.1 (von Neumann, 1928). *Let* $P \in \mathbb{R}^{m\times n}$, $X = \{x \in \mathbb{R}^m_{\geqslant 0} : \sum_{i=1}^{m} x_i = 1\}$, $Y = \{y \in \mathbb{R}^n_{\geqslant 0} : \sum_{i=1}^{n} y_i = 1\}$. *Then,*

$$\max_{x\in X} \min_{y\in Y} p(x, y) = \min_{y\in Y} \max_{x\in X} p(x, y).$$

Proof. Again consider the linear program (16.1), and recall that the optimal solution of this linear program is equal to $\max_{x\in X} \min_{y\in Y} p(x, y)$. By adding a slack variable

$z \in \mathbb{R}^n$ with $z \geqslant 0$ we obtain the Lagrangian

$$\begin{aligned} L(v, x, z, w, y) &= v + \sum_{j=1}^{n} y_j \Big(\sum_{i=1}^{m} x_i p_{ij} - z_j - v \Big) - w \Big(\sum_{i=1}^{m} x_i - 1 \Big) \\ &= \Big(1 - \sum_{j=1}^{n} y_j \Big) v + \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} p_{ij} y_j - w \Big) x_i - \sum_{j=1}^{n} y_j z_j + w, \end{aligned}$$

where $w \in \mathbb{R}$ and $y \in \mathbb{R}^n$. The Lagrangian has a finite maximum for $v \in \mathbb{R}$ and $x \in \mathbb{R}^m$ with $x \geqslant 0$ if and only if $\sum_{j=1}^{n} y_j = 1$, $\sum_{j=1}^{n} p_{ij} y_j \leqslant w$ for $i = 1, \dots, m$, and $y \geqslant 0$. The dual of (16.1) is therefore

$$\begin{aligned} \text{minimize} \quad & w \\ \text{subject to} \quad & \sum_{j=1}^{n} p_{ij} y_j \leqslant w \quad \text{for } i = 1, \dots, m \\ & \sum_{j=1}^{n} y_j = 1 \\ & y \geqslant 0. \end{aligned}$$

It is easy to see that the optimal solution of the dual is $\min_{y \in Y} \max_{x \in X} p(x, y)$, and the theorem follows from strong duality. □

The number $\max_{x \in X} \min_{y \in Y} p(x, y) = \min_{y \in Y} \max_{x \in X} p(x, y)$ is also called the *value* of the matrix game with payoff matrix P.

17 Strategic Equilibrium

A pair of strategies $(x, y) \in X \times Y$ such that x is a best response to y and y is a best response to x is called an *equilibrium*. It is easily verified that both (C, D) and (D, C) are equilibria of the game of chicken, and we will see later that there is one more equilibrium, in which both players randomize between their two actions.

17.1 Equilibria of Matrix Games

The minimax theorem implies that every matrix game has an equilibrium, and in fact characterizes the set of equilibria of these games.

THEOREM 17.1. *A pair of strategies $(x, y) \in X \times Y$ is an equilibrium of the matrix game with payoff matrix P if and only if*

$$\begin{aligned} \min_{y' \in Y} p(x, y') &= \max_{x' \in X} \min_{y' \in Y} p(x', y') \quad \textit{and} \\ \max_{x' \in X} p(x', y) &= \min_{y' \in Y} \max_{x' \in X} p(x', y'). \end{aligned} \tag{17.1}$$

Proof. For all $(x, y) \in X \times Y$,

$$\min_{y' \in Y} \max_{x' \in X} p(x', y') \leqslant \max_{x' \in X} p(x', y) \geqslant p(x, y) \geqslant \min_{y' \in Y} p(x, y') \leqslant \max_{x' \in X} \min_{y' \in Y} p(x', y'),$$

and the first and last term are equal by Theorem 16.1.

If (x, y) is an equilibrium, the second and third inequality hold with equality. This means that the first and last inequality have to hold with equality as well, and (17.1) follows.

On the other hand, if (17.1) is satisfied, then the first and last inequality hold with equality. This means that the second and third inequality have to hold with equality as well, so (x, y) is an equilibrium. □

Other properties specific to matrix games are that all equilibria yield the same payoffs and that any pair of strategies of the two players, such that each of them is played in some equilibrium, is itself an equilibrium.

THEOREM 17.2. *Let $(x, y), (x', y') \in X \times Y$ be equilibria of the matrix game with payoff matrix P. Then $p(x, y) = p(x', y')$, and (x, y') and (x', y) are equilibria as well.*

Proof. Since equilibrium strategies are best responses to each other, we have that

$$p(x, y) \leqslant p(x, y') \leqslant p(x', y') \leqslant p(x', y) \leqslant p(x, y).$$

Since the first and last term are the same, the inequalities have to hold with equality and the first claim follows. Then,

$$\begin{aligned}
&p(x,y') = p(x',y') \geqslant p(z,y') && \text{for all } z \in X,\\
&p(x,y') = p(x,y) \leqslant p(x,z) && \text{for all } z \in Y,\\
&p(x',y) = p(x,y) \geqslant p(z,y) && \text{for all } z \in X, \text{ and}\\
&p(x',y) = p(x',y') \geqslant p(x',z) && \text{for all } z \in X,
\end{aligned}$$

where the inequalities hold because (x,y) and (x',y') are equilibria. Thus (x,y') and (x',y) are pairs of strategies that are best responses to each other, and the second claim follows as well. □

Theorems 16.1, 17.1, and 17.2 together also imply that the set of equilibria of a matrix game is convex.

17.2 Nash's Theorem

Many of the results concerning equilibria of matrix games do *not* carry over to bimatrix games, with the exception of existence.

THEOREM 17.3 (Nash, 1951). *Every bimatrix game has an equilibrium.*

We use the following result.

THEOREM 17.4 (Brouwer Fixed Point Theorem). *Let $f : S \to S$ be a continuous function, where $S \subseteq \mathbb{R}^n$ is closed, bounded, and convex. Then f has a fixed point.*

Proof of Theorem 17.3. Define X and Y as before, and observe that $X \times Y$ is closed, bounded, and convex. For $x \in X$ and $y \in Y$ define $s_i(x,y)$ and $t_j(x,y)$ as the additional payoff the two players could obtain by playing their ith or jth pure strategy instead of x or y, i.e.,

$$\begin{aligned}
&s_i(x,y) = \max\{0, p(e_i^m, y) - p(x,y)\} && \text{for } i = 1,\dots,m \text{ and}\\
&t_j(x,y) = \max\{0, q(x, e_j^n) - q(x,y)\} && \text{for } j = 1,\dots,n,
\end{aligned}$$

where e_ℓ^k denotes the ℓth unit vector in $\mathbb{R}^k$. Further define $f : (X \times Y) \to (X \times Y)$ by letting $f(x,y) = (x',y')$ with

$$x_i' = \frac{x_i + s_i(x,y)}{1 + \sum_{k=1}^m s_k(x,y)} \qquad \text{and} \qquad y_j' = \frac{y_j + t_j(x,y)}{1 + \sum_{k=1}^n t_k(x,y)}$$

for $i = 1,\dots,m$ and $j = 1,\dots,n$. Function f is continuous, so by Theorem 17.4 is must have a fixed point, i.e., a pair of strategies $(x,y) \in X \times Y$ such that $f(x,y) = (x,y)$.

Further observe that there has to exist $i \in \{1, \dots, m\}$ such that $x_i > 0$ and $s_i(x, y) = 0$, since otherwise

$$p(x,y) = \sum_{k=1}^{m} x_k p(e_k^m, y) > \sum_{k=1}^{m} x_k p(x,y) = p(x,y).$$

Therefore, and since (x, y) is a fixed point,

$$x_i = \frac{x_i + s_i(x,y)}{1 + \sum_{k=1}^{m} s_k(x,y)}$$

and thus

$$\sum_{k=1}^{m} s_k(x,y) = 0.$$

This means that for $k = 1, \dots, m$, $s_k(x, y) = 0$, and therefore

$$p(x,y) \geqslant p(e_k^m, y).$$

It follows that

$$p(x,y) \geqslant p(x', y) \quad \text{for all } x' \in X.$$

An analogous argument shows that $q(x, y) \geqslant q(x, y')$ for all $y' \in Y$, so (x, y) must be an equilibrium. □

Our requirement that a bimatrix game has a finite number of actions is crucial for this result. This can be seen very easily by considering a game where the set of actions of each player is the set of natural numbers, and players get a payoff of 1 if they choose a number that is greater than the one chosen by the other player, and zero otherwise.

17.3 The Complexity of Finding an Equilibrium

The proof of Theorem 17.3 relies on fixed points of a continuous function and does not give rise to a finite method for finding an equilibrium. Quite surprisingly, equilibrium computation turns out to be more or less a combinatorial problem.

Define the *support* of strategy $x \in X$ as $S(x) = \{i \in \{1, \dots, m\} : x_i > 0\}$, and that of strategy $y \in Y$ as $S(y) = \{j \in \{1, \dots, m\} : y_j > 0\}$. It is easy to see that a mixed strategy is a best response if and only if all pure strategies in its support are best responses: if one of them was not a best response, then the payoff could be increased by reducing the probability of that strategy, and increasing the probabilities of the other strategies in the support appropriately. In other words, randomization over the support of an equilibrium does not happen for the player's own sake, but to allow the other player to respond in a way that sustains the equilibrium.

It also follows from these considerations that finding an equilibrium boils down to finding its supports. Once the supports are known, the precise strategies can be computed by solving a set of equations, which in the two-player case are linear. For supports of sizes k and ℓ, there is one equation for each player stating that the probabilities sum up to one, and $k-1$ or $\ell-1$ equations, respectively, stating that the expected payoff is the same for every pure strategy in the support. Solving these $k+\ell$ equations in $k+\ell$ variables yields k values for player 1 and ℓ values for player 2. If the solution corresponds to a strategy profile and expected payoffs are maximized by the pure strategies in the support, then an equilibrium has been found. An equilibrium with supports of size two in the game of chicken would have to satisfy $x_1 + x_2 = 1$, $y_1 + y_2 = 1$, $2x_1 + 1x_2 = 3x_1 + 0x_2$, and $2y_1 + 1y_2 = 3y_1 + 0y_2$. The unique solution, $x_1 = x_2 = y_1 = y_2 = 1/2$, also satisfies the additional requirements and therefore is an equilibrium. No equilibrium with full supports exists in the prisoner's dilemma, because the corresponding system of equalities does not have a solution.

A basic procedure for finding an equilibrium, and in fact all equilibria, is to iterate over all possible supports and check for each of them whether there is an equilibrium with that support. The running time of this method is finite, but clearly exponential in general. It is natural to ask whether there is a hardness result that stands in the way of a polynomial-time algorithm. While the equilibrium condition can easily be verified for a given pair of strategies, which implies membership in NP, the notion of NP-hardness seems inappropriate: equilibria always exist and the decision problem is therefore trivial. On the other hand, NP-hardness follows immediately if the problem is modified slightly to obtain a non-trivial decision problem.

THEOREM 17.5. *Given a bimatrix game, it is NP-complete to decide whether it has at least two equilibria; an equilibrium in which the expected payoff of the row player is at least a given amount; an equilibrium in which the expected sum of payoff of the two players is a least a given amount; an equilibrium with supports of a given minimum size; an equilibrium whose support includes a given pure strategy; or an equilibrium whose support does not include a given pure strategy.*

In the next lecture we will consider an algorithm that searches the possible supports in a more organized way. This algorithm also provides an alternative, combinatorial, proof of existence, and will lead us to the appropriate complexity class for the problem of finding an equilibrium.

18 Equilibrium Computation

Consider a bimatrix game with payoff matrices $P, Q \in \mathbb{R}^{m \times n}$, and assume without loss of generality that $P, Q > 0$. It will be convenient to index the actions of the row and column player by $M = \{1, \dots, m\}$ and $N = \{m+1, \dots, m+n\}$, respectively, and define the sets X and Y of strategies accordingly.

A pair $(x, y) \in X \times Y$ is an equilibrium if and only if all pure strategies in $S(x)$ are best responses to y and all pure strategies in $S(y)$ are best responses to x, i.e., if

$$\begin{aligned} &\text{for all } i \in M, \quad x_i > 0 \text{ implies } (Py)_i = \max_{k \in M} (Py)_k \text{ and} \\ &\text{for all } j \in N, \quad y_j > 0 \text{ implies } (Q^T x)_j = \max_{k \in N} (Q^T x)_k. \end{aligned} \tag{18.1}$$

18.1 Labeled Polytopes

Not every subset of actions can be the support of an equilibrium, and it makes sense to search for equilibrium supports in a more organized way. In the following we restrict our attention to non-degenerate games, but note that degeneracies can be handled by a slight perturbation of the payoff matrices. A bimatrix game is *non-degenerate* if for every $(x, y) \in X \times Y$, $|S(x)| \geqslant |S(y)|$ if y is a best response to x and $|S(y)| \geqslant |S(x)|$ if x is a best response to y.

Let

$$\begin{aligned} \hat{X} &= \{(x, u) \in \mathbb{R}^m \times \mathbb{R} : x \geqslant 0, x^T \mathbf{1} = 1, Q^T x \leqslant u\mathbf{1}\} \quad \text{and} \\ \hat{Y} &= \{(y, v) \in \mathbb{R}^n \times \mathbb{R} : y \geqslant 0, y^T \mathbf{1} = 1, Py \leqslant v\mathbf{1}\}, \end{aligned}$$

where $\mathbf{1}$ denotes an all-ones vector of appropriate dimension. The first two inequalities for $\hat{X}$ ensure that x is indeed a strategy of the row player, while the third inequality ensures that u is an upper bound on the expected payoff for every pure strategy of the column player. We will be interested in the extreme points of $\hat{X}$ and $\hat{Y}$, because they satisfy some of the constraints with equality and therefore correspond to strategies with specific properties: if $x_i = 0$, then the pure strategy $i \in M$ is not played in strategy x; if $(Q^T x)_j = u$, then the pure strategy $j \in N$ is a best response to x. In the case of $\hat{X}$ there are $m+1$ variables, so in addition to the equality constraint at least m of the inequality constraints have to hold with equality at every extreme point. An extreme point (x, u) of $\hat{X}$ thus corresponds to a situation where $|S(x)| = k$, for some $k \leqslant m$, and at least k pure strategies of the column player are a best response to x. By non-degeneracy, there in fact have to be exactly k such strategies. Analogously, at every extreme point (y, v) of $\hat{Y}$, $|S(y)| = k$ for some $k \leqslant n$ and k pure strategies of the row player are a best response to y. It is easy to see an equilibrium (x, y) of a non-degenerate game must satisfy $|S(x)| = |S(y)|$, so that every equilibrium of such a game can be described as a pair of extreme points of $\hat{X}$ and $\hat{Y}$.

Since $P, Q > 0$ implies that $u, v > 0$, we can simplify notation and consider

$$\begin{aligned}\overline{X} &= \{x \in \mathbb{R}^m : x \geqslant 0, Q^T x \leqslant \mathbf{1}\} \quad \text{and}\\ \overline{Y} &= \{y \in \mathbb{R}^n : y \geqslant 0, Py \leqslant \mathbf{1}\}.\end{aligned}$$

The vectors in $\overline{X}$ are no longer normalized, but there is a direct correspondence between the extreme points of $\hat{X}$ and $\overline{X}$, except for the zero vector: for each extreme point (x, u) of $\hat{X}$, x/u is an extreme point of $\overline{X}$, and for each extreme point x of $\overline{X}$, apart from the zero vector, $(x/\sum_{i=1}^m x_i, 1/\sum_{i=1}^m x_i)$ is an extreme point of $\hat{X}$. Analogous conditions hold for $\hat{Y}$ and $\overline{Y}$.

We will now give a characterization of those extreme points of $\overline{X}$ and $\overline{Y}$ that correspond to an equilibrium of the underlying game. For given extreme points x and y of $\overline{X}$ and $\overline{Y}$, let $L(x)$ and $L(y)$ be the index sets of those constraints that hold with equality, i.e.,

$$\begin{aligned}L(x) &= \{i \in M : x_i = 0\} \cup \{j \in N : (Q^T x)_j = 1\} \quad \text{and}\\ L(y) &= \{j \in N : y_j = 0\} \cup \{i \in M : (Py)_i = 1\}.\end{aligned}$$

We refer to the sets $L(x)$ and $L(y)$ as the *labels* of x and y and call a pair (x, y) *fully labeled* if $L(x) \cup L(y) = M \cup N$.

Theorem 18.1. *A pair of extreme points* $(x, y) \in \overline{X} \times \overline{Y}$ *with* $(x, y) \neq (0, 0)$ *corresponds to an equilibrium if and only if it is fully labeled.*

Proof. Suppose that $L(x) \cup L(y) = M \cup N$, and let

$$\begin{aligned}M_1 &= \{i \in M : x_i = 0\}, & N_1 &= \{j \in N : y_j = 0\},\\ M_2 &= \{i \in M : (Py)_i = 1\}, & N_2 &= \{j \in N : (Q^T x)_j = 1\}.\end{aligned}$$

Then, $M_1 \cup M_2 = M$ and $N_1 \cup N_2 = N$, and it is easy to see that (18.1) is satisfied.

Conversely assume that (x, y) corresponds to an equilibrium. Then, by (18.1), $(M \setminus S(x)) \cup S(y) \subseteq L(x)$ and $(N \setminus S(y)) \cup S(x) \subseteq L(y)$, and thus $L(x) \cup L(y) = M \cup N$. □

18.2 The Lemke-Howson Algorithm

The relationship between completely labeled pairs of extreme points and equilibria of the underlying game can be used to obtain a combinatorial algorithm for finding an equilibrium. The idea is to start from $(0, 0)$ and pivot alternatingly in $\overline{X}$ and $\overline{Y}$ until a completely labeled pair is found. To make this idea precise, let V_X and V_Y be the sets of extreme points of $\overline{X}$ and $\overline{Y}$, and let E_X and E_Y be the set of edges between adjacent extreme points, i.e.,

$$\begin{aligned}E_X &= \{(x, x') \in V_X \times V_X : |L(x) \cap L(x')| = m - 1\}\\ E_Y &= \{(y, y') \in V_Y \times V_Y : |L(y) \cap L(y')| = n - 1\}.\end{aligned}$$

Further let $V = V_X \times V_Y$ and $E = \{((x,y),(x',y')) \in V \times V : (x,x') \in E_X$ or $(y,y') \in E_Y\}$.

The key observation is that if we restrict our attention to extreme points that are almost fully labeled, with the possible exception of a particular label ℓ, then there is always a unique way in which we can proceed. For $\ell \in M \cup N$, let $V_\ell = \{(x,y) \in V : L(x) \cup L(y) \supseteq M \cup N \setminus \{\ell\}\}$ and $E_\ell = E \cap (V_\ell \times V_\ell)$.

THEOREM 18.2. *Let* $\ell \in M \cup N$. *Then,* V_ℓ *contains* $(0,0)$ *as well as every pair* $(x,y) \in V$ *that corresponds to an equilibrium of the underlying game. If the underlying game is non-degenerate, then* $(0,0)$ *and the elements of* V_ℓ *corresponding to an equilibrium have degree one in the graph* (V_ℓ, E_ℓ)*, and all other elements of* V_ℓ *have degree two.*

Proof. Both $(0,0)$ and all pairs corresponding to equilibria are fully labeled, so the first property is obvious.

For the second property, consider $(x,y) \in V_\ell$ and let $\bar{x}$ and $\bar{y}$ be the strategies in the underlying game corresponding to x and y. Since the game is non-degenerate, at most $|S(\bar{x})|$ pure strategies can be a best response to $\bar{x}$, and at most $|S(\bar{y})|$ pure strategies can be a best response to $\bar{y}$, so $L(x) \leqslant |M \setminus S(x)| + S(x) = m$ and $L(y) \leqslant |N \setminus S(y)| + S(y) = n$. Since x and y are extreme points of $\overline{X}$ and $\overline{Y}$, respectively, we have that $|L(x)| \geqslant m$ and $|L(y)| \geqslant n$, and thus $L|(x)| = m$ and $|L(y)| = n$.

If $(x,y) = (0,0)$, or if $(\bar{x},\bar{y})$ is an equilibrium, then (x,y) is fully labeled and therefore $L(x) \cap L(y) = \emptyset$. The neighbors of (x,y) are those elements of V_ℓ that replace ℓ with some other label, i.e., those where some other constraint holds with equality instead of the one corresponding to ℓ. Since the constraints are linear, and by non-degeneracy, there is exactly one such element.

Otherwise $L(x) \cap L(y) = \{k\}$ for some $k \in M \cup N$. The neighbors of (x,y) can again be obtained by replacing k with another label. This replacement can be done either in $\overline{X}$ or in $\overline{Y}$, and for each of the two there is one neighbor by the same reasoning as before. □

This means that the graph (V_ℓ, E_ℓ) for each $\ell \in M \cup N$ consists of paths and cycles that are pairwise disjoint, and the ends of the paths correspond to the pair $(0,0)$ and to the equilibria of the underlying game. The Lemke-Howson algorithm now takes the obvious steps: it starts from $(0,0)$ and follows the path until it reaches the pair of extreme points at the other end, which must correspond to an equilibrium. Moving from a vertex $(x,y) \in V_\ell$ to a neighboring vertex $(x',y') \in V_\ell$ corresponds to dropping a label from either x or y and picking up another label with the same element. In the first round the label that is dropped is ℓ. In subsequent rounds it is the duplicate label that has been picked up in the previous round, but it is dropped from the respective other element. Eventually ℓ will be picked up, leading to a fully labeled pair. Formally the algorithm proceeds as follows:

1. Pick $\ell \in M \cup N$. Let (x,y) be the pair obtained from $(0,0)$ by dropping label ℓ.

2. Let $k \in L(x') \cap L(y')$ be the label that was just picked up. If it was picked up by x, then drop it from y; otherwise drop it from x. Let (x, y) be the resulting pair.
3. If (x, y) is fully labeled, then stop. Otherwise go to Step 2.

Theorem 18.2 also shows that every non-degenerate game has at least one equilibrium, and in fact that the number of equilibria in such games is odd.

COROLLARY 18.3. *Every non-degenerate bimatrix game has an odd number of equilibria.*

18.3 Complementary Pivoting

Adding slack variables $r \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$ turns the best response conditions into $Q^T x + s = 1$ and $Py + r = 1$, where $x, y, r, s \geqslant 0$. The pair (x, y) then is completely labeled if and only if $x^T r = 0$ and $y^T s = 0$. Dropping a label corresponds to increasing or decreasing one of the variables such that one of the constraints, the one corresponding to the label being dropped, no longer holds with equality. When the variable is increased or decreased as much as possible, a different constraint starts to hold with equality, and the label corresponding to this constraint is picked up. This procedure can be carried out through pivoting in a tableau, in a very similar way to pivoting in the simplex method.

Consider for example the bimatrix game given by

$$P = \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix} \qquad \text{and} \qquad Q = \begin{pmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{pmatrix},$$

Indexing r and s by $M = \{1, \dots, m\}$ and $N = \{m+1, \dots, m+n\}$, respectively, the constraint $Q^T x + s = 1$ can be written in tableau form as follows:

x_1	x_2	x_3	s_4	s_5	
3	2	3	1	0	1
2	6	1	0	1	1

Assume that label $\ell = 2$ is dropped. In that case, want to add variable x_2 to the basis, by pivoting on the column that corresponds to this variable. The corresponding pivot row is the one that minimizes the ratio between the entry in the last column and the entry in the pivot column among those that have a positive entry in the latter. Pivoting then works by dividing the pivot row by the entry in the pivot column in order to make that entry equal to one, and adding a multiple of the pivot row to the other rows in order to make all other entries in the pivot column equal to zero. We obtain the following tableau:

$\frac{7}{3}$	0	$\frac{8}{3}$	1	$-\frac{1}{3}$	$\frac{2}{3}$
$\frac{1}{3}$	1	$\frac{1}{6}$	0	$\frac{1}{6}$	$\frac{1}{6}$

The second row now corresponds to variable x_2 that has entered the basis. On the other hand, variable s_5 has left the basis, and $5 \in L(x) \cap L(y)$. We thus want to turn to the constraint $Py + r = 1$ and drop the duplicate label 5. The initial tableau for this constraint looks as follows:

y_4	y_5	r_1	r_2	r_3	
3	3	1	0	0	1
2	5	0	1	0	1
0	6	0	0	1	1

By pivoting on the second column, corresponding to y_5, and on the third row, we pick up label 3 and obtain the following tableau:

3	0	1	0	$-\frac{1}{2}$	$\frac{1}{2}$
2	0	0	1	$-\frac{5}{6}$	$\frac{1}{6}$
0	1	0	0	$\frac{1}{6}$	$\frac{1}{6}$

Pivoting one more time in each of the two polytopes, we drop label 3 to pick up label 4, and drop label 4 to pick up label $\ell = 2$ and obtain a fully labeled pair. The final tableaus look as follows:

$\frac{7}{8}$	0	1	$\frac{3}{8}$	$-\frac{1}{8}$	$\frac{1}{4}$
$\frac{3}{16}$	1	0	$-\frac{3}{48}$	$\frac{3}{16}$	$\frac{1}{8}$

0	0	1	$-\frac{3}{2}$	$\frac{3}{4}$	$\frac{1}{4}$
1	0	0	$\frac{1}{2}$	$-\frac{5}{12}$	$\frac{1}{12}$
0	1	0	0	$\frac{1}{6}$	$\frac{1}{6}$

Reading off the values of x and y from the last column of each tableau and scaling them appropriately yields the equilibrium $((0, 1/3, 2/3), (1/3, 2/3))$.

18.4 The Complexity of Finding an Equilibrium Revisited

The Lemke-Howson algorithm creates a graph consisting of disjoint paths and cycles, such that one endpoint of a path can be found very easily and any other endpoint corresponds to an equilibrium. Clearly, there must be at least one other endpoint, and this endpoint can be found by following the path from the initial one. The problem is that the size of the graph, and thus potentially the length of the path in question, is exponential. Indeed, there exist games for which the Lemke-Howson algorithm takes an exponential number of steps until it terminates.

It turns out that there is a variety of other computational problems for which the existence of a solution is guaranteed by the same type of argument. One of them is the problem of finding a fixed point of a function satisfying the conditions of Brouwer's theorem. These problems comprise the complexity class PPAD (for polynomial parity argument, directed case), and the equilibrium problem is complete for this class under an appropriate type of reduction.

THEOREM 18.4 (Chen and Deng, 2006). *Finding an equilibrium of a bimatrix game is PPAD-complete.*

The question whether equilibria can be found in polynomial time is thus equivalent to the question whether PPAD $=$ P, or whether there exists a general method that avoids following paths in the graph and instead takes a shortcut to a solution. While it could in principle be the case that PPAD $=$ P even when P $\neq$ NP, this is still considered rather unlikely, one of the reasons being that many of the problems in PPAD are notoriously hard.

19 Cooperative Games

So far we have considered non-cooperative games, in which each players acts on its own. The theory of cooperative games, on the other hand, studies which coalitions of players are likely to form in a given situation. It takes the payoff achievable by any coalition as given, but requires that coalitions can distribute these payoffs among their members in such a way that the members are satisfied. We consider games with transferable payoff, where the payoff obtained by a coalition can be distributed in an arbitrary way among its members, and restrict our attention to the grand coalition N consisting of all players.

Formally, a *coalitional game* is given by a set $N = \{1, \dots, n\}$ of *players*, and a *characteristic function* $v : 2^N \to \mathbb{R}$ that maps each coalition of players to its *value*, the joint payoff the coalition can obtain by working together. For a given game (N, v), a vector $x \in \mathbb{R}^n$ of payoffs is said to satisfy *(economic) efficiency* if $\sum_{i \in N} x_i = v(N)$ and *individual rationality* if $x_i \geqslant v(\{i\})$ for $i = 1, \dots, n$. The first condition intuitively ensures that no payoff is wasted, while the second condition ensures that each player obtains at least the same payoff it would be able to obtain on its own. A payoff vector that is both efficient and individually rational is also called an *imputation*.

19.1 The Core

Efficiency and individual rationality may not be enough to guarantee a stable outcome. For any two imputations x and y, $\sum_{i \in N} x_i = \sum_{i \in N} y_i = v(N)$, so $y_i > x_i$ for some $i \in N$ implies that $y_j < x_j$ for some other $j \in N$. However, there could be some coalition $S \subseteq N$ such that $y_i > x_i$ for all $i \in S$. If in addition $\sum_{i \in S} y_i \leqslant v(S)$, the members of S could increase their respective payoffs by deviating from the grand coalition, forming the coalition S, and distributing the payoff thus obtained according to y. The core is the set of imputations that are stable against this kind of deviation. Formally, imputation x is in the core of game (N, v) if $\sum_{i \in S} x_i \geqslant v(S)$ for all $S \subseteq N$.

Consider a situation where $n \geqslant 2$ members of an expedition have discovered a treasure, and any pair of them can carry one piece of the treasure back home. This situation can be modeled by a coalitional game (N, v) where $N = \{1, \dots, n\}$ and $v(S) = |S|/2$ if $|S|$ is even and $v(S) = (|S| - 1)/2$ if $|S|$ is odd. The core then contains all imputations if $n = 2$, the single imputation $(1/2, \dots, 1/2)$ if $n \geqslant 4$ is even, and is empty if n is odd. The latter can for example be shown using a characterization of games with a non-empty core, which we discuss next.

Call a function $\lambda : 2^N \to [0, 1]$ *balanced* if for every player the weights of all coalitions containing that player sum to 1, i.e., if for all $i \in N$, $\sum_{S \subseteq N \setminus \{i\}} \lambda(S \cup \{i\}) = 1$. A game (N, v) is called *balanced* if for every balanced function λ, $\sum_{S \subseteq N} \lambda(S) v(S) \leqslant v(N)$. The intuition behind this definition is that each player allocates one unit of time among the coalitions it is a member of, and each coalition earns a fraction of its value proportional to the minimum amount of time devoted to it by any of its members. Balancedness of a collection of weights imposes a feasibility condition on players' allocations of time, and a game is balanced if there is no feasible allocation that yields more than $v(N)$.

THEOREM 19.1 (Bondareva 1963, Shapley 1967). *A game has a non-empty core if and only if it is balanced.*

Proof. The core of a game (N, v) is non-empty if and only if the linear program to

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in N} x_i \\ \text{subject to} \quad & \sum_{i \in S} x_i \geqslant v(S) \quad \text{for all } S \subseteq N \end{aligned}$$

has an optimal solution with value $v(N)$. This linear program has the following dual:

$$\begin{aligned} \text{maximize} \quad & \sum_{S \subseteq N} \lambda(S) v(S) \\ \text{subject to} \quad & \sum_{S \subseteq N, i \in S} \lambda(S) = 1 \quad \text{for all } i \in N \\ & \lambda(S) \geqslant 0 \quad \text{for all } S \subseteq N, \end{aligned}$$

where $\lambda : 2^N \to [0,1]$. Note that λ is feasible for the dual if and only if it is a balanced function. Both primal and dual are feasible, so by strong duality their optimal objective values are the same. This means that the core is non-empty if and only if $\sum_{S \subseteq N} \lambda(S) v(S) \leqslant v(N)$ for every balanced function λ. □

To see that the core of our example game is empty if n is odd, define $\lambda : 2^N \to [0,1]$ such that $\lambda(S) = 1/(n-1)$ if $|S| = 2$ and $\lambda(S) = 0$ otherwise. Then, for all $i \in N$, $\sum_{S \subseteq N \setminus \{i\}} \lambda(S \cup \{i\}) = 1$, because each player is contained in exactly $(n-1)$ sets of size 2. Moreover, $\sum_{S \subseteq N} \lambda(S) v(S) = n(n-1)/2 \cdot 1/(n-1) = n/2$, which is greater than $v(N)$ if n is odd.

19.2 The Nucleolus

In cases where the core is empty, one might consider weakening the requirement that no coalition should be able to gain, and instead look for an efficient payoff vector that minimizes the possible gain over all coalitions. This can intuitively be interpreted as minimizing players' incentive to deviate from the solution by forming another coalition, or as a natural notion of fairness when distributing the joint payoff $v(N)$ among the players.

To this end, define the *excess* $e(S, x)$ of coalition $S \subseteq N$ for payoff vector x as its gain from leaving the grand coalition, i.e., $e(S, x) = v(S) - \sum_{i \in S} x_i$. For a given vector

x, let $S^x_1, \dots, S^x_{2^n-1}$ be an ordering of the coalitions such that $e(S^x_k, x) \geqslant e(S^x_{k+1}, x)$ for $k = 1, \dots, 2^n - 2$, and let $E(x) \in \mathbb{R}^{2^n-1}$ be the vector given by $E_k(x) = e(S^x_k, x)$. We say that $E(x)$ is lexicographically smaller than $E(y)$ if there exists $i \in \{1, \dots, 2^n - 1\}$ such that $E_k(x) = E_k(y)$ for $k = 1, \dots, i-1$ and $E_i(x) < E_i(y)$. The *nucleolus* is then defined as the set of efficient payoff vectors x for which $E(x)$ is lexicographically minimal.

Observe that for each $k = 1, \dots, 2^n - 1$, E_k is a continuous function because

$$E_k(x) = \min_{\substack{\mathcal{T} \subseteq 2^N \\ |\mathcal{T}| = k-1}} \max_{S \in 2^N \setminus \mathcal{T}} e(S, x).$$

Since E_1 is continuous, $X_1 = \arg\min_{x \in X_0} E_1(x)$, where X_0 is the set of efficient payoff vectors, is non-empty and compact. By induction, the same holds for $X_k = \arg\min_{x \in X_{k-1}} E_k(x)$, $k \geqslant 2$. Since X_{2^n-1} is the nucleolus, the nucleolus is non-empty. It can in fact be shown that the nucleolus always contains exactly one element.

Theorem 19.2. *The nucleolus of any coalitional game is a singleton.*

Proof. Consider two vectors x and y in the nucleolus, and assume for contradiction that $x \neq y$. Observe that $E(x) = E(y)$, and let $S_1, \dots, S_{2^n-1}$ be an ordering of the coalitions such that $e(S_k, x) \geqslant e(S_{k+1}, x)$ for $k = 1, \dots, 2^n - 2$. Since $x \neq y$, there has to exist $\ell \in \{1, \dots, 2^n - 1\}$ such that $e(S_k, x) = e(S_k, y)$ for $k = 1, \dots, \ell - 1$ and $e(S_\ell, x) \neq e(S_\ell, y)$. In fact $e(S_\ell, x) > e(S_\ell, y)$, because $e(S_\ell, x) < e(S_\ell, y)$ would imply that $E(x)$ is lexicographically smaller than $E(y)$. Moreover, for $k = \ell + 1, \dots, 2^n - 1$, $E_k(x) \leqslant E_\ell(x)$ and $e(S_k, y) \leqslant E_\ell(x)$. The latter inequality holds because $e(S_k, y) = E_k(y)$ for $k = 1, \dots, \ell - 1$, so it must be the case that $e(S_k, y) \leqslant E_\ell(y) = E_\ell(x)$ for $k = \ell + 1, \dots, 2^n - 1$.

Now consider the vector $z = (x + y)/2$, and observe that $e(S_k, z) = (e(S_k, x) + e(S_k, y))/2$ for $k = 1, \dots, 2^n - 1$. Thus $E_k(z) = E_k(x)$ for $k = 1, \dots, \ell - 1$, $E_\ell(z) < E_\ell(x)$, and $E_k(z) < E_\ell(x)$ for $k = \ell + 1, \dots, 2^n - 1$. This means that $E(z)$ is lexicographically smaller than $E(x)$, contradicting the assumption that x is in the nucleolus. □

The set of efficient payoff vectors that minimize maximum excess is the set of optimal solutions of the following linear program:

$$\begin{aligned} &\text{minimize} && \epsilon \\ &\text{subject to} && \textstyle\sum_{i \in S} x_i \geqslant v(S) - \epsilon \quad \text{for all } S \subset N \\ &&& \textstyle\sum_{i \in N} x_i = v(N). \end{aligned} \tag{P_1}$$

The set of feasible solutions of (P_1) for a fixed value of ϵ is also called the ϵ-*core*, and the set of optimal solutions of (P_1) the *least core*. The core is non-empty if and only if the optimal objective value of (P_1) is non-positive.

Now let $\mathcal{S}_1 \subseteq 2^N \setminus \{N\}$ be the set of coalitions whose constraint holds with equality in *every* optimal solution of (P_1). Clearly, if x is in the nucleolus, then $e(S, x) = E_1(x)$ for all $S \in \mathcal{S}_1$. Given that $\mathcal{S}_1$ is non-empty, we can thus write down a new linear program that fixes the excess of the coalitions in $\mathcal{S}_1$ to $E_1(x)$ and minimizes the next smaller excess. By repeating this procedure we obtain a sequence of linear programs $P_2, P_3, \dots$ defined recursively as follows:

$$\begin{array}{lll} \text{minimize} & \epsilon & \\ \text{subject to} & \sum_{i \in S} x_i = v(S) - \epsilon_1 & \text{for all } S \in \mathcal{S}_1 \\ & \vdots & \\ & \sum_{i \in S} x_i = v(S) - \epsilon_{i-1} & \text{for all } S \in \mathcal{S}_{i-1} \\ & \sum_{i \in S} x_i \geqslant v(S) - \epsilon & \text{for all } S \in \bar{\mathcal{S}}_{i-1} \\ & \sum_{i \in N} x_i = v(N), & \end{array} \tag{P_i}$$

where for $j = 1, \dots, i-1$, ϵ_j is the optimal objective value of (P_j), $\mathcal{S}_j \subseteq \bar{\mathcal{S}}_{j-1}$ is the set of coalitions with an inequality constraint in (P_j) that holds with equality in every optimal solution, and $\bar{\mathcal{S}}_j = (2^N \setminus \{N\}) \setminus \cup_{\ell=1}^{j} \mathcal{S}_\ell$.

The following result establishes that there always exists an inequality constraint that can be tightened to an equality, which guarantees that we obtain the nucleolus after a finite number of iterations. In fact, n iterations can be shown to suffice if in addition to the constraints that are tight in every optimal solution we also fix those that are linearly dependent on constraints that have already been fixed.

LEMMA 19.3. *If $\bar{\mathcal{S}}_{i-1} \neq \emptyset$, then $\mathcal{S}_i \neq \emptyset$.*

Proof. Let $\bar{\mathcal{S}}_{i-1} = \{S_1, \dots, S_m\}$. If $m = 0$ or if the constraint for some $S \in \bar{\mathcal{S}}_{i-1}$ holds with equality in every optimal solution of (P_i) we are done. Otherwise, for $j = 1, \dots, m$, there exists an optimal solution x^j of (P_i) such that $e(S, x^j) < \epsilon_i$. By convexity of the set of optimal solutions, $\tilde{x} = 1/m \sum_{i=1}^{m} x^j$ is optimal for (P_i), which means that there has to be a coalition $\tilde{S} \in \bar{\mathcal{S}}_{i-1}$ such that $e(\tilde{S}, \tilde{x}) = \epsilon_i$. Thus,

$$e(\tilde{S}, \tilde{x}) = \frac{1}{m} \sum_{j=1}^{m} e(\tilde{S}, x^j) \leqslant \epsilon_i,$$

where the equality holds by convexity of e and the inequality because for $j = 1, \dots, m$, x^j is optimal for (P_i) and thus $e(\tilde{S}, x^j) \leqslant \epsilon_i$. If the constraint for $\tilde{S}$ had slack for some optimal solution of (P_i), i.e., if $\tilde{S} = S_j$ for some $j = 1, \dots, m$, then this inequality would be strict. Since it is not, the constraint for $\tilde{S}$ must hold with equality in every optimal solution of (P_i). □

Consider for example a game with $N = \{1, 2, 3\}$ and characteristic function v given by

$$\begin{array}{c} v(\{1\}) = 1 \quad v(\{2\}) = 2 \quad v(\{3\}) = 1 \\ v(\{1,2\}) = 2 \quad v(\{1,3\}) = 3 \quad v(\{2,3\}) = 5 \quad v(\{1,2,3\}) = 4. \end{array} \tag{19.1}$$

To find the nucleolus of this game, we first

$$\begin{aligned}
\text{minimize}\quad & \epsilon \\
\text{subject to}\quad & x_1 \geqslant 1-\epsilon, \quad x_2 \geqslant 2-\epsilon, \quad x_3 \geqslant 1-\epsilon \\
& x_1+x_2 \geqslant 2-\epsilon, \quad x_1+x_3 \geqslant 3-\epsilon \\
& x_2+x_3 \geqslant 5-\epsilon, \quad x_1+x_2+x_3 = 4.
\end{aligned}$$

It is easily verified that $\epsilon = 1$ and $x = (0,1,3)$ is a feasible solution. By adding the constraints for $\{1\}$ and $\{2,3\}$ and subtracting the constraint for $\{1,2,3\}$ we see that $\epsilon \geqslant 1$, so the solution must be optimal. For $\epsilon = 1$, the constraints for $\{1\}$ and $\{2,3\}$ have to hold with equality in every optimal solution, and the constraints for $\{3\}$, $\{1,2\}$, and $\{1,2,3\}$ become redundant. Thus $x_1 = 0$, $x_2 \geqslant 1$, $x_3 \geqslant 2$, and $x_2 + x_3 = 4$, and the least core is $\{(0, x_2, 4-x_2) : x_2 \in [1,2]\}$. We now

$$\begin{aligned}
\text{minimize}\quad & \epsilon \\
\text{subject to}\quad & x_1 = 0, \quad x_2 \geqslant 2-\epsilon \\
& x_3 \geqslant 3-\epsilon, \quad x_2+x_3 = 4
\end{aligned}$$

and obtain a unique optimal solution where $\epsilon = 1/2$ and $x = (0, 3/2, 5/2)$, which gives us the unique element in the nucleolus.

19.3 The Shapley Value

A different notion of fairness in distributing the joint payoff of a coalition among its members was proposed by Shapley, starting from a set of axioms. Call player $i \in N$ a *dummy* if its contribution to every coalition is exactly its value, i.e., if $v(S \cup \{i\}) = v(S) + v(\{i\})$ for all $S \subseteq N \setminus \{i\}$. Call two players $i, j \in N$ *interchangeable* if they contribute the same to every coalition, i.e., if $v(S \cup \{i\}) = v(S \cup \{j\})$ for all $S \subseteq N \setminus \{i,j\}$. Let a *solution* be a function $\phi : \mathbb{R}^{2^n} \to \mathbb{R}^n$ that maps every characteristic function v to an efficient payoff vector $\phi(v)$. Solution ϕ is said to satisfy

- *dummies* if $\phi_i(v) = v(\{i\})$ whenever i is a dummy;
- *symmetry* if $\phi_i(v) = \phi_j(v)$ whenever i and j are interchangeable; and
- additivity if $\phi(v+w) = \phi(v) + \phi(w)$.

It turns out that there is a unique solution satisfying these axioms.

THEOREM 19.4. *The* Shapley value*, given by*

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!} \big(v(S \cup \{i\}) - v(S)\big),$$

is the unique solution that satisfies dummies, symmetry, and additivity.

The Shapley value of player i can be interpreted as its average contribution over all possible sequences in which the players can join the grand coalition. For the three-player game with characteristic function given in (19.1), we for example have

$$\begin{aligned}\phi_1(v) &= \frac{0!2!}{3!}(v(\{1\}) - v(\emptyset)) + \frac{1!1!}{3!}(v(\{1,2\}) - v(\{2\})) + \\ &\quad \frac{1!1!}{3!}(v(\{1,3\}) - v(\{3\})) + \frac{2!0!}{3!}(v(\{1,2,3\}) - v(\{2,3\})) \\ &= \frac{1}{3}\,1 + \frac{1}{6}\,0 + \frac{1}{6}\,2 + \frac{1}{3}\,(-1) = \frac{1}{3}.\end{aligned}$$

20 Bargaining

20.1 Bargaining Problems

Bargaining theory investigates how agents should cooperate when non-cooperation may result in outcomes that are Pareto dominated. Formally, a (two-player) *bargaining problem* is a pair (F, d) where $F \subseteq \mathbb{R}^2$ is a convex set of *feasible outcomes* and $d \in F$ is a *disagreement point* that results if players fail to agree on an outcome. Here, convexity corresponds to the assumption that any lottery over feasible outcomes is again feasible. A *bargaining solution* then is a function that assigns to every bargaining problem (F, d) a unique element of F.

The most basic example of a bargaining problem is the so-called ultimatum game given by $F = \{(v_1, v_2) \in \mathbb{R}^2 : v_1 + v_2 \leqslant 1\}$ and $d = (0, 0)$, in which two players receive a fixed amount of payoff if they can agree on a way to divide this amount among themselves. This game has many equilibria when viewed as a normal-form game, since disagreement results in a payoff of zero to both players. Players' preferences regarding these equilibria differ, and bargaining theory tries to answer the question which equilibrium should be chosen. More generally, a two-player normal-form game with payoff matrices $P, Q \in \mathbb{R}^{m \times n}$ can be interpreted as a bargaining problem where $F = \mathrm{conv}(\{(p_{ij}, q_{ij}) : i = 1, \ldots, m,\ j = 1, \ldots, n\})$, $d_1 = \max_{x \in X} \min_{y \in Y} p(x, y)$, and $d_2 = \max_{y \in Y} \min_{x \in X} q(x, y)$, given that $(d_1, d_2) \in F$. Here, $\mathrm{conv}(S)$ denotes the convex hull of set S.

Two kinds of approaches to bargaining exist in the literature: a strategic one that considers iterative procedures resulting in an outcome in F, and an axiomatic one that tries to identify bargaining solutions that possess certain desirable properties. We will focus on the axiomatic approach in this lecture.

20.2 Nash's Bargaining Solution

For a given bargaining problem (F, d), Nash proposed to

$$\begin{aligned} \text{maximize} \quad & (v_1 - d_1)(v_2 - d_2) \\ \text{subject to} \quad & v \in F \\ & v \geqslant d. \end{aligned} \tag{20.1}$$

The objective function of this optimization problem is strictly quasi-concave and therefore has a unique maximum. Formally, a function $f : S \to \mathbb{R}$ defined on a convex set S is *strictly quasi-concave* if for all $x, y \in S$ with $x \neq y$ and every $\delta \in (0, 1)$, $f(x) \geqslant f(y)$ implies $f((1 - \delta)x + \delta y) > f(y)$. In other words, strict quasi-concavity means that the interior of any line segment joining points on two level sets of f lies strictly above

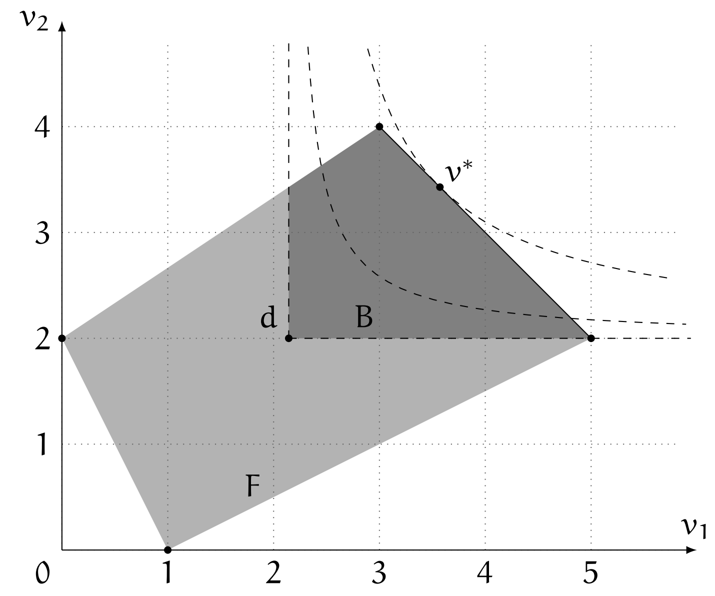

Figure 20.1: Illustration of the Nash bargaining solution

the level set corresponding to the lower value of the function. The objective function of (20.1) satisfies this criterion because its level sets are rectangular hyperbolae with horizontal and vertical asymptotes. Optimization problem (20.1) thus defines a bargaining solution, the so-called *Nash bargaining solution.*

Consider for example the two-player game with payoff matrices

$$P = \begin{pmatrix} 0 & 5 \\ 3 & 1 \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} 2 & 2 \\ 4 & 0 \end{pmatrix}.$$

In this game, the row player can guarantee a payoff of 15/7 by playing the two rows with probabilities 2/7 and 5/7, respectively. The column player can guarantee a payoff of 2 by playing the left column. The bargaining problem corresponding to this game is shown in Figure 20.1. The set F is the convex hull of the four payoff vectors $(0, 2)$, $(5, 2)$, $(3, 4)$, and $(1, 0)$, and it contains the feasible set $B = \{v \in F : v \geqslant d\}$ of (20.1). The disagreement point is $d = (15/7, 2)$. Level sets of the objective function corresponding to values 0 and 1 and to the optimal value are drawn as dashed curves. The Nash bargaining solution v^* is the unique point in the intersection of F with the optimal level set.

To compute v^*, we first observe that $v^* \in \{(v_1, v_2) : v_2 = 7 - v_1, 3 \leqslant v_1 \leqslant 5\}$. The objective function becomes

$$(v_1 - d_1)(v_2 - d_2) = (v_1 - \frac{15}{7})(5 - v_1) = \frac{50}{7}v_1 - v_1^2 - \frac{75}{7},$$

and has a stationary point if $50/7 - 2v_1 = 0$. We obtain $v^* = (25/7, 24/7)$, which is indeed a maximum.

While it is not obvious that maximizing the product of the excess of the two players is a good idea, it turns out that the Nash bargaining solution can be characterized using a set of simple axioms. Bargaining solution f is

- *Pareto efficient* if $f(F, d)$ is not Pareto dominated in F for any bargaining problem (F, d);
- *symmetric* if $(f(F, d))_1 = (f(F, d))_2$ for every bargaining problem (F, d) such that $(y, x) \in F$ whenever $(x, y) \in F$ and $d_1 = d_2$;
- *invariant under positive affine transformations* if $f(F', d') = \alpha \circ f(F, d) + \beta$ for any $\alpha, \beta \in \mathbb{R}^2$ with $\alpha > 0$ and any two bargaining problems (F, d) and (F', d') such that $F' = \{\alpha \circ x + \beta : x \in F\}$ and $d' = \alpha \circ d + \beta$; and
- *independent of irrelevant alternatives* if $f(F, d) = f(F', d)$ for any two bargaining problems (F, d) and (F', d) such that $F' \subseteq F$ with $d \in F'$ and $f(F, d) \in F'$.

Here, $\circ$ denotes component-wise multiplication of vectors, i.e., $(s \circ t)^T = (s_1 t_1, s_2 t_2)$ for all $s, t \in \mathbb{R}^2$.

In the context of bargaining, Pareto efficiency means that no payoff is wasted, and symmetry is an obvious fairness property. Invariance under positive affine transformations should hold because payoffs are just a representation of the underlying ordinal preferences. The intuition behind independence of irrelevant alternatives is that an outcome only becomes easier to justify as a solution when other outcomes are removed from the set of feasible outcomes.

THEOREM 20.1. *Nash's bargaining solution is the unique bargaining solution that is Pareto efficient, symmetric, invariant under positive affine transformations, and independent of irrelevant alternatives.*

Proof. We denote the Nash bargaining solution by f^N and begin by showing that it satisfies the axioms. For Pareto efficiency, this follows directly from the fact that the objective function is increasing in v_1 and v_2. For symmetry, assume that $d_1 = d_2$ and let $v^* = (v_1^*, v_2^*) = f^N(F, d)$. Clearly (v_2^*, v_1^*) maximizes the objective function, and by uniqueness of the optimal solution $(v_2^*, v_1^*) = (v_1^*, v_2^*)$ and thus $f_1^N(F, d) = f_2^N(F, d)$. For invariance under positive affine transformations, define F' and d' as above, and observe that $f^N(F', d')$ is an optimal solution of the problem to maximize $(v_1 - \alpha_1 d_1 - \beta_1)(v_2 - \alpha_2 d_2 - \beta_2)$ subject to $v \in F'$, $v_1 \geqslant d_1$, and $v_2 \geqslant d_2$. By setting $v' = \alpha \circ v + \beta$, it follows that $f^N(F', d') = \alpha \circ f^N(F, d) + \beta$. For independence of irrelevant alternatives, let $v^* = f^N(F, d)$ and $F' \subseteq F$. If $v^* \in F'$, it remains optimal and thus $v^* = f^N(F', d)$.

Now consider a bargaining solution f that satisfies the axioms, and fix F and d. Let $z = f^N(F, d)$, and let F' be the image of F under an affine transformation that maps z to $(1/2, 1/2)$ and d to the origin, i.e.,

$$F' = \{\alpha \circ v + \beta : v \in F, \alpha \circ z + \beta = (1/2, 1/2)^T, \alpha \circ d + \beta = 0\}.$$

Since both f and f^N are invariant under positive affine transformations, $f(F, d) = f^N(F, d)$ if and only if $f(F', 0) = f^N(F', 0)$. It thus suffices to show that $f(F', 0) = (1/2, 1/2)$.

We begin by showing that for all $v \in F'$, $v_1 + v_2 \leqslant 1$. Assume for contradiction that there exists $v \in F$ with $v_1 + v_2 > 1$, and let $t^\delta = (1-\delta)(1/2, 1/2)^T + \delta v$. By convexity of F', $t^\delta \in F'$ for $\delta \in (0, 1)$. Moreover, since the objective function has a unique maximum, we can choose δ sufficiently small such that $t_1^\delta t_2^\delta > 1/4 = f^N(F', 0)$, contradicting optimality of $f^N(F', 0)$.

Now let F'' be the closure of F' under symmetry, and observe that for all $v \in F''$, $v_1 + v_2 \leqslant 1$. Therefore, by Pareto optimality and symmetry of f, $f(F'', 0) = (1/2, 1/2)^T$. Since f is independent of irrelevant alternatives, $f(F', 0) = (1/2, 1/2)^T$ as required. $\square$

21 Stable Matchings

Assume that there are two disjoint sets of agents, each having preferences over agents in the other set, and the goal is to find an assignment between agents in one set and agents in the other. Real-world examples for a setting like this include the assignment of students to schools or of medical residents to hospitals. We restrict our attention to the special case in which each agent is assigned to at most one agent in the other set. This case has originally been described in terms of marriages and is therefore sometimes referred to as the stable marriage problem.

We describe the problem in terms of a set S of *students* and a set A of potential *advisors*. Each student $i \in S$ has a strict linear order $\succ_i \subseteq A \times A$, each advisor $j \in A$ a strict linear order $\succ_j \subseteq S \times S$. A *matching* is a function $\mu : (S \cup A) \to (S \cup A)$ such that $\mu(\mu(i)) = i$ for all $i \in S \cup A$, $\mu(i) \in A$ for all $i \in S$, and $\mu(j) \in S$ for all $j \in A$. We henceforth assume that $|S| = |A|$, but note that the results can be extended to the case where some agents remain unmatched or prefer remaining unmatched to being matched to certain other agents.

A pair $(i, j) \in S \times A$ is a *blocking pair* for matching μ if i and j would rather be matched to each other than to their respective partners in μ, i.e., if $j \succ_i \mu(i)$ and $i \succ_j \mu(j)$. A matching is called *stable* if it does not have any blocking pairs. Stability is desirable in practice because matchings that are not stable tend to unravel: if two agents find out that they form a blocking pair, they have an incentive to leave the matching; this can introduce additional blocking pairs and eventually cause the whole matching to break down.

It is not obvious, but nevertheless true, that there always exists a stable matching. This can be shown by the so-called *deferred acceptance* procedure. There are two symmetric variants of this procedure, one in which students propose and one in which advisors propose. We concentrate on the former, but note that all results hold symmetrically for the latter. Deferred acceptance proceeds as follows:

1. Let each student $i \in S$ propose to the advisor it ranks highest.
2. Match advisor $j \in A$ tentatively to the highest-ranked among the students that have proposed to it, if any, and let it reject the others for good.
3. Let each student that has just been rejected but has not been rejected by all advisors propose to its next preferred advisor. If there are no such students, then stop and return the tentatively matched pairs. Otherwise go to Step 2.

THEOREM 21.1 (Gale and Shapley, 1962). *There always exists a stable matching.*

Proof. The deferred acceptance procedure clearly terminates after a finite number of rounds, and yields a matching when it does. Let μ be this matching, and assume for contradiction that it is not stable. Then there must exist a blocking pair $(i, j) \in S \times A$,

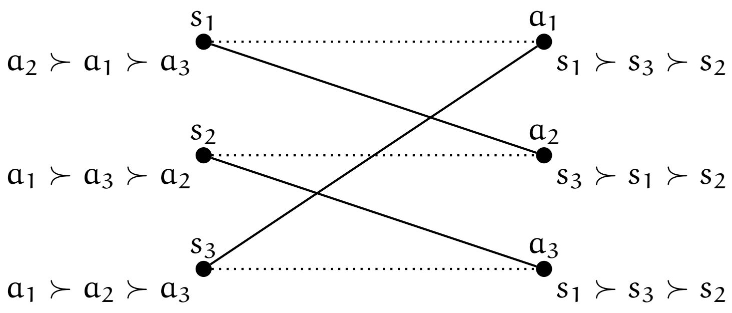
s_1
a_1
$a_2 \succ a_1 \succ a_3$
$s_1 \succ s_3 \succ s_2$
s_2
a_2
$a_1 \succ a_3 \succ a_2$
$s_3 \succ s_1 \succ s_2$
s_3
a_3
$a_1 \succ a_2 \succ a_3$
$s_1 \succ s_3 \succ s_2$

Figure 21.1: Matching problem and two matchings. In the first round of students-propose deferred acceptance, s_1 proposes to a_2 and both s_2 and s_3 propose to a_1, who in turn rejects s_2. Since s_2 was rejected it proposes to a_3 in the second round, and the procedure terminates in the stable matching indicated by solid lines. The matching indicated by dotted lines is not stable, because (s_1, a_2) is a blocking pair.

and in particular $j \succ_i \mu(i)$. In the deferred acceptance procedure i must therefore have proposed to j before proposing to $\mu(i)$, and since the two were not matched, j must have rejected i. This means that j must have received a proposal from a student it ranks higher than i but not higher than the student $\mu(j)$ it was eventually matched with. Thus $\mu(j) \succ_j i$, contradicting the fact that (i, j) is a blocking pair. □

21.1 Optimality

There generally exist multiple stable matchings, so one might ask how they compare to each other in quality. Call $j \in A$ *achievable* for $i \in S$ if there exists a stable matching μ such that $\mu(i) = j$. A stable matching μ is then called *student-optimal* if for all $i \in S$, $\mu(i)$ is most preferred among the advisors achievable for i. It is not clear that there should be a stable matching that is student-optimal, i.e., simultaneously gives every student the most preferred advisor it is assigned to in any stable matching. Intriguingly, students-propose deferred acceptance yields such a matching.

THEOREM 21.2. *Students-propose deferred acceptance yields a student-optimal stable matching.*

Proof. Assume for contradiction that the statement of the theorem is false. Then there has to exist a student $i \in S$ and an advisor $j \in A$ such that j is achievable for i but rejects i in students-propose deferred acceptance. Let k be the round in which this happens, and assume without loss of generality that no student is rejected by an achievable advisor in any earlier round. Let $i' \in S$ be the student tentatively matched to j at the end of round k, so that $i' \succ_j i$. Since j is achievable for i, there must exist a stable matching μ with $\mu(i) = j$. Observe that also $\mu(j) = i$, and thus $i' \succ_j \mu(j)$.

Now let $j' = \mu(i')$, which means in particular that j' is achievable for i'. We claim that $j \succ_{i'} j'$. If this was not the case, i.e., if instead $j' \succ_{i'} j$, then i' would have had to propose to, and be rejected by, j' before being tentatively matched to j. This is impossible because the latter happened in round k, and we assumed that no student was rejected by an achievable advisor before round k. Therefore, $j \succ_{i'} \mu(i')$.

We conclude that (i', j) is a blocking pair for μ, contradicting its stability. $\square$

Curiously, any student-optimal stable matching μ is necessarily *advisor-pessimal*, meaning that for all $j \in A$, $\mu(j)$ is least preferred among the students achievable for j.

THEOREM 21.3. *Every student-optimal stable matching is advisor-pessimal.*

Proof. Consider a student-optimal stable matching μ and a stable matching ν, and assume that $\mu(j) \succ_j \nu(j)$ for some $j \in A$. Let $i = \mu(j)$ and observe that $\nu(j) \neq i$. Since μ is student-optimal, $\mu(i) \succ_i \nu(i)$. Thus (i, j) is a blocking pair for ν, a contradiction. $\square$

21.2 The Stable Matching Polytope

The deferred acceptance procedure necessarily leads to a stable matching that is as good for one side and bad for the other. One might wonder whether it is possible to find a stable matching that is fair for both sides, e.g., one that minimizes $\sum_{i \in S} \sum_{j \in A} r(i,j) + r(j,i)$ or $\max_{(i,j) \in (S \times A) \cup (A \times S)} r(i,j)$, where $r(i,j)$ denotes the rank of agent j in the preference order of agent i. It turns out that this can be done for any linear objective, using a characterization of stable matchings as the extreme points of a convex polytope. Fix a stable matching μ and let $x_{ij} = 1$ for $i \in S$ and $j \in A$ if $\mu(i) = j$, and $x_{ij} = 0$ otherwise. Then,

$$\begin{aligned} \sum_{j \in A} x_{ij} &= 1 \quad \text{for all } i \in S, \\ \sum_{i \in S} x_{ij} &= 1 \quad \text{for all } j \in A, \\ x_{ij} + \sum_{k : j \succ_i k} x_{ik} + \sum_{k : i \succ_j k} x_{kj} &\leqslant 1 \quad \text{for all } i \in S,\ j \in A, \\ x_{ij} &\geqslant 0 \quad \text{for all } i \in S,\ j \in A. \end{aligned}$$

The first two constraints are satisfied since every agent is matched with exactly one agent of the respective other type. If the third constraint was not satisfied, then $\sum_{k : j \succ_i k} x_{ik} = 1$ and $\sum_{k : i \succ_j k} x_{kj} = 1$, which would mean that (i, j) is a blocking pair. The last constraint obviously holds as well.

Let P be the polytope described by the above constraints, and observe that by Theorem 21.1, $P \neq \emptyset$. We moreover have the following.

THEOREM 21.4. *A vector is an extreme point of* P *if and only if it is a stable matching.*

22 Social Choice

Social choice theory asks how the possibly conflicting preferences of a set of agents can be aggregated into a collective decision, and in particular which properties the aggregate choice should satisfy and which properties can be satisfied simultaneously. Examples of settings that can be studied in the framework of social choice theory include voting, resource allocation, coalition formation, and matching.

22.1 Social Welfare Functions

Let $N = \{1, \dots, n\}$ be a set of agents, or *voters*, and $A = \{1, \dots, m\}$ a set of alternatives. Assume that $n, m \geqslant 2$ and finite. Assume that each voter $i \in N$ has a strict linear order $\succ_i \in L(A)$ over A, and the goal is to map the profile $(\succ_i)_{i \in N}$ of individual preference orders to a social preference order. This is achieved by means of a *social welfare function* (SWF) $f : L(A)^n \to L(A)$.

When $m = 2$, selecting the social preference order that is preferred by a majority of the voters is optimal in a rather strong sense. An SWF $f : L(A)^n \to L(A)$ is

- *anonymous* if for every permutation $\pi \in S_n$ of the voters and all preference profiles $\succ, \succ' \in L(A)^n$ such that $a \succ_i b$ if and only if $a \succ'_{\pi(i)} b$ for all $a, b \in A$, it holds that $f(\succ) = f(\succ')$;
- *neutral* if for every permutation $\pi \in S_m$ of the alternatives and all preference profiles $\succ, \succ' \in L(A)^n$ such that $a \succ_i b$ if and only if $\pi(a) \succ'_i \pi(b)$ for all $a, b \in A$, it holds that $a\ f(\succ)\ b$ if and only if $\pi(a)\ f(\succ')\ \pi(b)$ for all $a, b \in A$; and
- *monotone* if for all $\succ, \succ' \in L(A)^n$ and $a, b \in A$, $a\ f(\succ)\ b$ and $\{i \in N : a \succ_i b\} \subseteq \{i \in N : a \succ'_i b\}$ imply that $a\ f(\succ')\ b$.

Anonymity requires that voters are treated equally, symmetry that alternatives are treated equally, and monotonicity that an alternative cannot become less preferred socially when it becomes more preferred by individuals. When the number of voters is odd, these intuitive fairness and welfare properties precisely characterize the majority rule.

THEOREM 22.1. *Consider an SWF $f : L(A)^n \to L(A)$, where $|A| = 2$ and n is odd. Then f is the majority rule if and only if it is anonymous, neutral, and monotone.*

Proof sketch. Let $A = \{a, b\}$. By anonymity, the social preference only depends on the number of voters that prefer a to b. By neutrality, the social preference has to change between a preference profile where $\lfloor n/2 \rfloor$ voters prefer a to b and one where $\lceil n/2 \rceil$ voters prefer a to b. By monotonicity, the socially preferred alternative can never change from a to b when the number of voters who prefer a to b increases, so this is actually the unique change, and it follows that f is the majority rule. □

a	b	c
b	c	a
c	a	b

Figure 22.1: An instance of the Condorcet paradox. Each column lists the preferences of a particular voter.

The result also holds for weak preference orders if monotonicity is replaced by positive responsiveness, which requires that a weak social preference for b over a changes to a strict preference for a when some voter changes from a strict preference for b to a weak preference for a or from a weak preference for b to a strict preference for a.

In light of this result, it seems promising to base the decision on pairwise comparisons of alternatives even when $m > 2$. As the Marquis de Condorcet pointed out already in 1785, this is somewhat problematic, since the pairwise majority relation may contain cycles. To see this, consider a situation with three alternatives a, b, and c, and three voters with preferences as shown in Figure 22.1. It is easily verified that a majority of the voters prefers a over b, a majority prefers b over c, and a majority prefers c over a.

Unfortunately, this kind problem is not specific to the majority rule, but applies to every SWF satisfying a set of desirable criteria. An SWF $f : L(A)^{|N|} \to L(A)$ is

- *Pareto optimal* if for all $a, b \in A$ and every $\succ \in L(A)^n$ such that $a \succ_i b$ for all $i \in N$, it holds that $a \succ' b$, where $\succ' = f(\succ)$;
- *independent of irrelevant alternatives* (IIA) if for all $a, b \in A$ and all $\succ, \succ' \in L(A)^n$ such that $\succ_i \cap (\{a,b\} \times \{a,b\}) = \succ'_i \cap (\{a,b\} \times \{a,b\})$ for all $i \in N$, it holds that $f(\succ) \cap (\{a,b\} \times \{a,b\}) = f(\succ') \cap (\{a,b\} \times \{a,b\})$; and
- *dictatorial* if there exists $i \in N$ such that for all $\succ \in L(A)^n$, $f(\succ) = \succ_i$.

Pareto optimality requires that alternative a is socially preferred over alternative b when every voter prefers a over b. Independence of irrelevant alternatives requires that the social preference with respect to a and b only depends on individual preferences with respect to a and b, but not on those with respect to other alternatives. Finally, an SWF is dictatorial if the social preference order is determined by a single voter. It turns out that dictatorships are the only SWFs for three or more alternatives that are Pareto optimal and IIA.

Theorem 22.2 (Arrow, 1951). *Consider an SWF $f : L(A)^n \to L(A)$, where $|A| \geqslant 3$. If f is Pareto optimal and IIA, then f is dictatorial.*

Requiring non-dictatorship and Pareto optimality is rather uncontroversial. Relaxing IIA for example enables *Kemeny's rule*, which chooses a social preference order $\succ'$ that maximizes the number of agreements with the individual preferences, such that

$$\sum_{i \in N} |\succ' \cap \succ_i| = \max_{\succ'' \in L(A)} \sum_{i \in N} |\succ'' \cap \succ_i|. \tag{22.1}$$

4	3	2
a	c	b
b	b	c
c	a	a

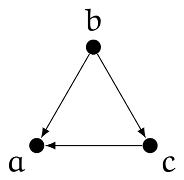

Figure 22.2: Preferences of three types of voters over a set of three alternatives (left), and the graph of the corresponding majority relation (right). Each column of the table on the left lists the preferences of a particular type of voter, the number of voters of that type is given in the top row. In the graph on the right, a directed edge from alternative x to alternative y indicates that a majority of the voters prefer x to y.

This maximization problem is NP-hard, but can be written as an integer program. An interesting alternative characterization of Kemeny's rule is as the *maximum likelihood estimator* for a simple probabilistic model in which votes are generated by an underlying "true" preference order. Fix a preference profile $\succ \in L(A)^n$ and define a vector $w \in \mathbb{N}^{m \times m}$ by letting w_{xy} be the number of voters who prefer x to y, i.e., $w_{xy} = \{i \in N : x \succ_i y\}$. Now assume that this vector was instead generated by picking a single preference order $\succ' \in L(A)$ and a probability $p \in (1/2, 1]$, and for each voter $i \in N$ and each pair of alternatives $x, y \in A$, letting i choose between x and y according to $\succ'$ with probability p and opposite to $\succ'$ with probability $1-p$. The probability of obtaining vector w from preference order $\succ'$ would then be

$$\mathbb{P}(w \mid \succ') = p^{\sum_{i \in N} |\succ_i \cap \succ'|}(1-p)^{nm(m-1)/2 - \sum_{i \in N} |\succ_i \cap \succ'|},$$

and it is easy to see that this probability is maximized by a preference order $\succ'$ that satisfies (22.1).

22.2 Social Choice Functions

Instead of the explicit assumptions of Theorem 22.2, one could also relax an implicit one. In particular, it might often suffice to identify a single best alternative rather than giving a complete ranking. This is achieved by a *social choice function* (SCF) $f : L(A)^n \to A$. Two of the most familiar SCFs are *plurality*, which chooses an alternatives ranked first by the largest number of voters, and *single transferable vote* (STV), which successively eliminates alternatives ranked first by the fewest voters until only one alternative remains.

Consider for example a situation with three alternatives a, b, and c, and nine voters with preferences as shown on the left of Figure 22.2. In this situation, plurality selects alternative a because it is ranked first by 4 voters, compared to 3 for c and 2 for b. STV first eliminates alternative b, which is ranked first by only 2 voters. Restricting attention to the remaining alternatives, a is ranked first by 4 voters and c by 5 voters. Alternative a is thus eliminated next, while alternative c remains and is selected. The graph of the majority relation shown on the right of Figure 22.2 illustrates that alternative b is a so-called *Condorcet winner*, i.e., it is preferred to any other alternative by a majority of the voters, while alternative a is a *Condorcet loser*, i.e., a majority of voters prefer any other alternative to a. The example of Figure 22.1 shows that a Condorcet winner or loser need not exist, but it is certainly reasonable to require that a Condorcet winner is selected when it exists, and that a Condorcet loser is never selected. An SCF satisfying the former property is called Condorcet consistent, and the example of Figure 22.2 shows that that neither plurality nor STV are Condorcet consistent.

23 Mechanism Design

Our discussion of social choice has so far ignored strategic considerations. Mechanism design augments social choice with game-theoretic reasoning, and effectively tries to construct games which yield a certain desirable outcome in equilibrium.

23.1 Strategic Manipulation

Again consider the situation of Figure 22.2, where we predicted that the use of plurality would result in the selection of alternative a. This prediction ignores, however, that voters of the third type have an incentive to misrepresent their preferences and claim that they prefer c over b: assuming that ties are broken in favor of c, only a single voter of the third type would have to change its reported preferences in this way to ensure that c is selected instead of a, an outcome this voter prefers. A similar problem exists with STV, where voters of the first type could benefit by pretending that their most preferred alternative is b, with the goal of having this alternative selected instead of their least preferred alternative, c. More generally, we say that SCF f is *manipulable* if there exist $i \in N$, $\succ \in L(A)^n$, and $\succ'_i \in L(A)$ such that $f((\succ_{-i}, \succ'_i)) \succ_i f(\succ)$, where $(\succ_{-i}, \succ'_i) = (\succ_1, \dots, \succ_{i-1}, \succ'_i, \succ_{i+1}, \succ_n)$ is the preference profile obtained by replacing voter i's preference order in $\succ$ by $\succ'_i$. SCF f is called *strategyproof* if it is not manipulable.

There are two obvious way to achieve strategyproofness: choosing an alternative based on the preferences of a single voter, or ignoring all but two alternatives and using majority rule to choose between these two. The first case corresponds to a dictatorship, the second to an SCF that is not surjective in the sense that some alternatives never get chosen. It turns out that these trivial cases are in fact the only SCFs that are strategyproof. Formally, SCF f is dictatorial if there exists $i \in N$ such that for all $\succ \in L(A)^n$ and $a \in A \setminus \{f(\succ)\}$, $f(\succ) \succ_i a$. SCF f is surjective if for all $a \in A$, there exists $\succ \in L(A)^n$ such that $f(\succ) = a$.

Theorem 23.1 (Gibbard, 1973; Satterthwaite, 1975). *Consider an SCF* $f : L(A)^n \to A$*, where* $|A| \geqslant 3$*. If* f *is surjective and strategyproof, then it is dictatorial.*

We need two lemmas. The first lemma states that a strategyproof SCF is monotone in the sense that the selected alternative does not change as long as all alternatives ranked below it are still ranked below it for all voters.

Lemma 23.2. *Let* f *be a strategyproof SCF,* $\succ \in L(A)^n$ *with* $f(\succ) = a$*. Then,* $f(\succ') = a$ *for every* $\succ' \in L(A)^n$ *such that for all* $i \in N$ *and* $b \in A \setminus \{a\}$*,* $a \succ'_i b$ *if* $a \succ_i b$*.*

Proof. We start from $\succ$ and change the preferences of one voter at a time until we get to $\succ'$, showing that the chosen alternative remains the same in every step. Let

$b = f(\succ'_1, \succ_{-1})$. By strategyproofness, $a \succeq_1 b$, and thus $a \succeq'_1 b$ by assumption. Also by strategyproofness, $b \succeq'_1 a$, and thus $a = b$. The claim now follows by repeating the same argument for the remaining voters. □

The second lemma states that the alternative selected by a surjective and strategyproof SCF must be Pareto optimal.

LEMMA 23.3. *Let* f *be a surjective and strategyproof SCF,* $a, b \in A$*, and* $\succ \in L(A)^n$ *such that* $a \succ_i b$ *for all* $i \in N$. *Then,* $f(\succ) \neq b$.

Proof. Assume for contradiction that $f(\succ) = b$. By surjectivity, there exists $\succ' \in L(A)^n$ such that $f(\succ') = a$. Let $\succ'' \in L(A)^n$ be a preference profile such that for all $i \in N$

$$a \succ''_i b \succ''_i x$$

for all $x \in A \setminus \{a, b\}$. Then, $x \succ_i b$ whenever $x \succ''_i b$ for some $i \in N$ and $x \in A \setminus \{b\}$, and $x \succ'_i a$ whenever $x \succ''_i a$ for some $i \in N$ and $x \in A \setminus \{a\}$. Thus, by Lemma 23.2, $f(\succ'') = f(\succ) = b$ and $f(\succ'') = f(\succ') = a$, a contradiction. □

Proof of Theorem 23.1. We first prove the theorem for $n = 2$ and then perform an induction on n.

Let $a, b \in A$ with $a \neq b$ and consider $\succ \in L(A)^2$ such that

$$a \succ_1 b \succ_1 x \qquad \text{and} \qquad b \succ_2 a \succ_2 x$$

for all $x \in A \setminus \{a, b\}$. Then, by Lemma 23.3, $f(\succ) \in \{a, b\}$.

Suppose that $f(\succ) = a$, and let $\succ' \in L(A)^2$ such that

$$a \succ'_1 b \succ'_1 x \qquad \text{and} \qquad b \succ'_2 x \succ'_2 a$$

for all $x \in A \setminus \{a, b\}$. Then, $f(\succ') = a$, since $f(\succ') \in \{a, b\}$ by Lemma 23.3 and $f(\succ') \neq b$ by strategyproofness. Lemma 23.2 now implies that f selects alternative a for any preference profile in which voter 1 ranks alternative a first.

By repeating the above analysis for every pair of distinct alternatives in A, we obtain two sets $A_1, A_2 \subseteq A$ such that A_i is the set of alternatives that are selected for every preference profile in which voter $i \in \{1, 2\}$ ranks them first. Let $A_3 = A \setminus (A_1 \cup A_2)$, and observe that $|A_3| \leqslant 1$: otherwise we would have performed the above analysis for two elements in A_3, which would place one of these elements in A_1 or A_2 and thus not in A_3.

Now observe that $|A| \geqslant 3$ and $|A_3| \leqslant 1$, so $|A_1 \cup A_2| \geqslant 2$. Moreover, for $x, y \in A$ with $x \neq y$, it cannot be the case that $x \in A_1$ and $y \in A_2$, because this would lead to a contradiction when voter 1 ranks x first and voter 2 ranks y first. Since $a \in A_1$, it follows that $A_1 \cap A_2 = \emptyset$ and thus that $A_2 = \emptyset$. It finally follows that $A_3 = \emptyset$: otherwise we could repeat the above analysis for $c \in A_3$ and $\succ'' \in L(A)^2$ with

$$c \succ''_1 a \succ''_1 x \qquad \text{and} \qquad a \succ''_2 c \succ''_2 x$$

for all $x \in A \setminus \{a, c\}$, and conclude that $c \in A_1$ or $a \in A_2$, a contradiction. It follows that $A_1 = A$, so voter 1 is a dictator.

Now we assume that the statement of the theorem holds for n voters and prove that it also holds for $n+1$ voters. Consider a surjective and strategyproof SCF $f : L(A)^{n+1} \to A$, and define $g : L(A)^2 \to A$ by letting

$$g(\succ_1, \succ_2) = f(\succ_1, \succ_2, \dots, \succ_2)$$

for all $\succ_1, \succ_2 \in L(A)$.

Since f is surjective and strategyproof, and by Lemma 23.3, g is surjective as well. Assume for contradiction that g is not strategyproof. By strategyproofness of f, the manipulator must be voter 2, so there must exist $\succ_1, \succ_2, \succ_2' \in L(A)$ and $a, b \in A$ such that $g(\succ_1, \succ_2) = a$, $g(\succ_1, \succ_2') = b$, and $b \succ_2 a$. For $k = 0, \dots, n$, let $\succ^k = (\succ_1, \succ_2', \dots, \succ_2', \succ_2, \dots, \succ_2) \in L(A)^{n+1}$ be the preference profile where k voters have preference order $\succ_2'$ and $n-k$ voters have preference order $\succ_2$, and let $a^k = f(\succ^k)$. Since $a^n = b \succ_2 a = a^0$, it must be the case that $a^{k+1} \succ_2 a^k$ for some k with $0 \leqslant k < n$, which means that f is manipulable, a contradiction. It follows that g is strategyproof, and therefore dictatorial.

If the dictator for g is voter 1, then by Lemma 23.2 voter 1 must also be a dictator for f. Assume instead that the dictator for g is voter 2, and let $h : L(A)^n \to A$ be given by

$$h(\succ_2, \dots, \succ_{n+1}) = f(\succ_1^*, \succ_2, \dots, \succ_{n+1})$$

for an arbitrary $\succ_1^* \in L(A)$. Then, h is strategyproof by strategyproofness of f, and surjective because voter 2 is a dictator for g. Therefore, by the induction hypothesis, h is dictatorial.

Assume without loss of generality that the dictator for h is voter 2, and let $e : L(A)^2 \to A$ be given by

$$e(\succ_1, \succ_2) = f(\succ_1, \succ_2, \succ_3^*, \dots, \succ_{n+1}^*)$$

for arbitrary $\succ_3^*, \dots, \succ_{n+1}^* \in L(A)$. Then e is strategyproof and surjective, and hence dictatorial. In fact, the dictator for e must be voter 2, because voter 1 is not a dictator for g and thus cannot be a dictator for e. Since $\succ_i^*$ for $i = 1, 3, \dots, n+1$ was chosen arbitrarily, it follows that voter 2 is a dictator for f. $\square$

23.2 Implementation of Social Choice Functions

A *mechanism design problem*, or *game form*, is given by a set A of *alternatives* and a set $N = \{1, \dots, n\}$ of *agents*, each with a set Θ_i of possible *types* and a *utility function* $u_i : A \times \Theta_i \to \mathbb{R}$. Note that a game form and a *type profile* $\theta \in \Theta = \times_{i \in N} \Theta_i$ together induce a normal-form game. A *mechanism* is given by a message space Σ_i for agent i and an outcome function $g : \times_{i \in N} \Sigma_i \to A$. A mechanism is called *direct* if the agents directly report their type to the mechanism, i.e., if $\Sigma_i = \Theta_i$ for all $i \in N$. The idea is that the agents send messages to the mechanism, providing information about their types, and depending on these messages the mechanism selects an alternative that optimizes some objective. The objective can be encoded by a social choice function.

Mechanism $M = ((\Sigma_i)_{i \in N}, g)$ is said to *implement* SCF $f : \times_{i \in N} \Theta_i \to A$ (in weakly dominant strategies) if there exist functions $s_i : \Theta_i \to \Sigma_i$ for $i \in N$ such that for every $\theta \in \Theta$, $g(s_1(\theta_1), \dots, s_n(\theta_n)) = f(\theta)$, and for all $i \in N$, $\theta_i \in \Theta_i$ and $\sigma \in \Sigma$, $u_i(g(s_i(\theta_i), \sigma_{-i}), \theta_i) \geqslant u_i(g(\sigma), \theta_i)$. An SCF is called *implementable* if it is implemented by some mechanism. A direct mechanism $M = ((\Theta_i)_{i \in N}, g)$ is called dominant strategy incentive compatible, or *strategyproof*, if for all $i \in N$, $\theta \in \Theta$, and $\theta_i' \in \Theta_i$, $u_i(g(\theta), \theta_i) \geqslant u_i(g(\theta_i', \theta_{-i}), \theta_i)$. The profile θ of true types is then also referred to as the truthful equilibrium of the mechanism.

It seems that in principle arbitrarily complicated mechanisms might be required to implement certain social choice functions. The following result implies that we can restrict our attention to strategyproof direct mechanisms.

Theorem 23.4 (Revelation Principle). *A social choice function is implementable if and only if it is implemented in the truthful equilibrium of a strategyproof direct mechanism.*

Proof. The theorem follows by observing that the direct mechanism can simulate the equilibrium strategies of the agents. Let f be an implementable SCF. Then there exists a mechanism $((\Sigma_i)_{i \in N}, g)$ and functions $s_i : \Sigma_i \to \Theta_i$ for $i \in N$ such that for every $\theta \in \Theta$, $g(s_1(\theta_1), \dots, s_n(\theta_n)) = f(\theta)$, and for every $i \in N$, $\theta_i \in \Theta_i$ and $\sigma \in \Sigma$, $u_i(g(s_i(\theta_i), \sigma_{-i}), \theta_i) \geqslant u_i(g(\sigma), \theta_i)$. Define $h : \Theta \to A$ by letting $h(\theta) = g(s_1(\theta_1), \dots, s_n(\theta_n))$ for all $\theta \in \Theta$. Then, for every $\theta \in \Theta$, $h(\theta) = f(\theta)$, and for all $i \in N$, $\theta \in \Theta$, and $\theta_i' \in \Theta_i$,

$$\begin{aligned} u_i(h(\theta), \theta_i) &= u_i(g(s_1(\theta_1), \dots, s_n(\theta_n)), \theta_i) \\ &\geqslant u_i(g(s_1(\theta_1), \dots, s_{i-1}(\theta_{i-1}), s_i(\theta_i'), s_{i+1}(\theta_{i+1}), \dots, s_n(\theta_n)), \theta_i) \\ &= u_i(h(\theta_i', \theta_{-i}), \theta_i). \end{aligned}$$

This means that (Θ, h) is a strategyproof direct mechanism that implements f, and the claim follows. □

It should be noted that indirect mechanisms can still be useful in certain settings, for example to reduce the amount of information agents have to send to the mechanism, or the amount of computation the mechanism has to carry out.

Theorems 23.1 and 23.4 imply that only dictatorial social choice function are implementable when there are more than two alternatives and utility functions u_i can be arbitrary. In the next lecture we will look at an interesting special case where this impossibility result can be circumvented.

24 Mechanisms with Payments

The Gibbard-Satterthwaite Theorem assumes that agents can have arbitrary preferences over the set of alternatives, and in particular does not apply in settings where the outcome selected by the mechanism includes monetary payments and the utility of each agent is *quasilinear*, i.e., a linear combination of a valuation for the alternative selected by the social choice function and the amount of money transfered to or from the agent. It is worth noting that this assumption makes utilities comparable across agents.

In cases where the outcome includes monetary payments, it will be instructive to separate these payments from the social choice and write a mechanism as a pair (f, p) of a social choice function $f : \Theta \to A$ and a payment function $p : \Theta \to \mathbb{R}^n$. The utility of agent i can then be written as $u_i(\theta', \theta_i) = v_i(f(\theta'), \theta_i) - p_i(\theta')$, where θ' is a profile of types revealed to the mechanism, θ_i is the true type of agent i, $v_i : A \times \Theta_i \to \mathbb{R}$ is a valuation function over alternatives, and $p_i(\theta') = (p(\theta'))_i$. The main result for the quasilinear setting is positive and provides a way to optimize the most natural social choice function, the one that maximizes social welfare. The *social welfare* of an alternative $a \in A$ is $\sum_{i \in N} v_i(a, \theta_i)$, i.e., the sum of all agents' valuations for this alternative.

24.1 Vickrey-Clark-Groves Mechanisms

The mechanisms implementing this social choice function are the so-called Vickrey-Clark-Groves (VCG) mechanisms. A mechanisms (f, p) is a *Vickrey-Clark-Groves mechanism* if

$$f(\theta) \in \arg\max_{a \in A} \sum_{i \in N} v_i(a, \theta_i) \qquad \text{and}$$
$$p_i(\theta) = h_i(\theta_{-i}) - \sum_{j \in N \setminus \{i\}} v_j(f(\theta), \theta_j) \qquad \text{for all } i \in N,$$

where $h_i : \Theta_{-i} \to \mathbb{R}$ is some function that depends on the types of all agents but i. The crucial component is the term $\sum_{j \in N \setminus \{i\}} v_j(f(\theta), \theta_j)$, which is equal to the social welfare for all agents but i. The utility of agent i adds its own valuation $v_i(f(\theta), \theta_i)$ and thus becomes equal to the social welfare of alternative $f(\theta)$ minus the term $h_i(\theta_{-i})$. The latter does not depend on θ_i and therefore has no strategic implications.

THEOREM 24.1. *VCG mechanisms are strategyproof.*

Proof. Let $i \in N$, $\theta \in \Theta$, and $\theta'_i \in \Theta_i$. Then,

$$u_i(\theta, \theta_i) = v_i(f(\theta), \theta_i) - p_i(\theta)$$

$$\begin{aligned}
&= \sum_{j \in N} v_j(f(\theta), \theta_j) - h_i(\theta_{-i}) \\
&\geqslant \sum_{j \in N} v_j(f(\theta'_i, \theta_{-i}), \theta_j) - h_i(\theta_{-i}) \\
&= u_i((\theta'_i, \theta_{-i}), \theta_i),
\end{aligned}$$

where the inequality holds because $f(\theta)$ maximizes social welfare with respect to θ. □

Strategyproofness holds for any choice of the functions h_i, so it is natural to ask for a good way to define these functions. In many cases it makes sense that agents are charged rather than paid, but not more than their gain from participating in the mechanism. Formally, mechanism (f, p) makes *no positive transfers* if $p_i(\theta) \geqslant 0$ for all $i \in N$ and $\theta \in \Theta$, and is *ex-post individually rational* if it always yields non-negative utility for all agents, i.e., if $v_i(f(\theta)) - p_i(\theta) \geqslant 0$ for all $i \in N$ and $\theta \in \Theta$. It turns out that these two properties can indeed be achieved simultaneously. The so-called *Clark pivot rule* sets $h_i(\theta_{-i}) = \max_{a \in A} \sum_{j \in N \setminus \{i\}} v_j(a, \theta_j)$, such that the payment of agent i becomes $p_i(\theta) = \max_{a \in A} \sum_{j \in N \setminus \{i\}} v_j(a, \theta_j) - \sum_{j \in N \setminus \{i\}} v_j(f(\theta))$. Intuitively, this latter amount is equal to the externality agent i imposes on the other agents, i.e., the difference between their social welfare with and without i's participation. The payment makes the agent internalize this externality.

THEOREM 24.2. *A VCG mechanism with Clarke pivot rule makes no positive transfers. If $v_i(a, \theta_i) \geqslant 0$ for all $i \in N$, $\theta_i \in \Theta_i$, and $a \in A$, it also is individually rational.*

Proof. Fix $\theta \in \Theta$ and $i \in N$, and let $a = f(\theta)$ and $b \in \arg\max_{a' \in A} \sum_{j \in N \setminus \{i\}} v_j(a', \theta_j)$. Then, by choice of b, $p_i(\theta) = \sum_{j \in N \setminus \{i\}} v_j(b, \theta_j) - \sum_{j \in N \setminus \{i\}} v_j(a, \theta_j) \geqslant 0$, so the mechanism makes no positive transfers. Moreover,

$$\begin{aligned}
u_i(\theta, \theta_i) &= v_i(a, \theta_i) + \sum_{j \in N \setminus \{i\}} v_j(a, \theta_j) - \sum_{j \in N \setminus \{i\}} v_j(b, \theta_j) \\
&\geqslant \sum_{j \in N} v_j(a, \theta_j) - \sum_{j \in N} v_j(b, \theta_j) \geqslant 0,
\end{aligned}$$

where the two inequalities respectively hold because $v_i(b, \theta_i) \geqslant 0$ and by choice of a. □

Consider for example the application of the VCG mechanism with Clarke pivot rule to an auction of a single good. In this case $A = N$, and the valuation function can be written as $v_i : A \to \mathbb{R}$, such that $v_i(a)$ is equal to agent i's valuation for the good if $a = i$ and zero otherwise. Since only a single agent can receive the good, $\max_{a \in A} \sum_{i \in N} v_i(a) = \max_{i \in N} v_i(i)$, and thus $f(\theta) \in \arg\max_{i \in N} v_i(i)$. Moreover, $p_i(\theta) = \max_{a \in A} \sum_{j \in N \setminus \{i\}} v_j(a) - \sum_{j \in N \setminus \{i\}} v_j(f(\theta))$. The first term is equal to $\max_{j \in N \setminus \{i\}} v_j(j)$ if $a = i$, the second term is zero if $a = i$ and equal to the first term otherwise, and thus $p_i(\theta) = \max_{j \in N \setminus \{i\}} v_j(j)$ if $f(\theta) = i$ and $p_i(\theta) = 0$ otherwise. We thus obtain the well-know Vickrey (or second-price) auction, which assigns the good to the agent with the highest bid and charges this agent a payment equal to the second-highest bid.

24.2 Characterizations of Strategyproof Mechanisms

One might wonder whether other objectives can be implemented in the quasilinear setting besides maximization of social welfare. Two characterizations of strategyproof mechanisms (f, p) exist in the literature. The first characterization states that a mechanism is strategyproof if and only if the payment of an agent is independent of its reported type and the chosen outcome simultaneously maximizes the utility of all agents.

THEOREM 24.3. *A mechanism* (f, p) *is strategyproof if and only if for every* $i \in N$ *and* $\theta \in \Theta$,

$$\begin{aligned} p_i(\theta) &= t_i(\theta_{-i}, f(\theta)) \qquad \textit{and} \\ f(\theta) &\in \arg\max_{a \in A(\theta_{-i})} (v_i(\theta_i, a) - t_i(\theta_{-i}, a)), \end{aligned}$$

where $t_i : \Theta_{-i} \times A \to \mathbb{R}$ *is a* price function *and* $A(\theta_{-i}) = \{f(\theta_i, \theta_{-i}) : \theta_i \in \Theta_i\}$ *is the range of* f *given that the reported types of all agents but* i *are fixed to* θ_{-i}.

Alternatively, strategyproof mechanisms can be characterized purely in terms of their social choice function. SCF f satisfies *weak monotonicity* if for all $\theta \in \Theta$, $i \in N$, and $\theta_i' \in \Theta_i$, $f(\theta) = a \neq b = f(\theta_i, \theta_{-i})$ implies that $v_i(a, \theta_i) - v_i(b, \theta_i) \geqslant v_i(a, \theta_i') - v_i(b, \theta_i')$. Intuitively, an SCF is weakly monotone if a change in the social choice due to a change of type of a single agent means that the agent's value for the new choice must have increased relative to its value for the old choice.

THEOREM 24.4. *If mechanism* (f, p) *is strategyproof, then* f *satisfies weak monotonicity. If SCF* f *satisfies weak monotonicity and for each* $i \in N$, $\{(v_i(a, \theta_i))_{a \in A} : \theta_i \in \Theta_i\} \subseteq \mathbb{R}^{|A|}$ *is a convex set, then there exists a payment function* $p : \Theta \to \mathbb{R}^n$ *such that* (f, p) *is strategyproof.*

This result reduces the characterization of strategyproof mechanisms to one of weakly monotone social choice function. The problem with the latter is that weak monotonicity is a local condition that is hard to check, and existence of a global condition depends on the domain of possible preferences. Good global conditions are known to exist for two extreme cases: domains that are unrestricted in the sense that the utilities an agent assigns to the alternatives in A can be arbitrary vectors in $\mathbb{R}^{|A|}$, and domains that are essentially one-dimensional.

A closer look at the unrestricted case reveals that the only strategyproof mechanisms are simple variations of VCG mechanisms, which allow for the assignment of weights to agents and alternatives and for restrictions of the range. SCF f is called an *affine* *maximizer* if there exist $A' \subseteq A$, $w_i \in \mathbb{R}_{>0}$ for $i \in N$, and $c_a \in \mathbb{R}$ for $a \in A'$ such that for every $\theta \in \Theta$, $f(\theta) \in \arg\max_{a \in A'}(c_a + \sum_{i \in N} w_i v_i(a, \theta_i))$. It is easy to see that VCG mechanisms can be generalized to affine maximizers.

THEOREM 24.5. *Let* f *be an affine maximizer, and for each* $i \in N$ *and* $\theta \in \Theta$*, let* $p_i(\theta) = h_i(\theta_{-i}) - \sum_{j \in N \setminus \{i\}}(w_j/w_i)v_j(f(\theta), \theta_j) - c_{f(\theta)}/w_i$*, where* $h_i : \Theta_{-i} \to \mathbb{R}$*. Then* (f, p) *is strategyproof.*

Proof. The utility of agent $i \in N$ is

$$u_i(\theta, \theta_i') = v_i(f(\theta), \theta_i') - h_i(\theta_{-i}) + \sum_{j \in N \setminus \{i\}} (w_j/w_i)v_j(f(\theta), \theta_j) + c_{f(\theta)}/w_i.$$

By adding $h_i(\theta_{-i})$, which does not depend on θ_i, and multiplying by w_i, we see that $u_i(\theta, \theta_i')$ can be maximized by maximizing $c_{f(\theta)} + \sum_{j \in N} w_j v_j(f(\theta), \theta_j')$. This happens when $\theta_i = \theta_i'$. $\square$

When there are at least three alternatives and preferences are unrestricted, affine maximizers are the only strategyproof mechanisms.

THEOREM 24.6 (Roberts, 1979). *Let* $|A| \geqslant 3$ *and* $\{(v_i(a, \theta_i))_{a \in A} : \theta \in \Theta\} = \mathbb{R}^{|A|}$ *for every* $i \in N$*. Let* $f : \theta \to A$ *be a surjective SCF,* $p : \Theta \to \mathbb{R}^n$ *a payment function. If* (f, p) *is strategyproof, then* f *is an affine maximizer.*