

# 1 Lagrangian Methods

## 1.1 Lagrangian methods

Let  $P(b)$  denote the optimization problem

$$\text{minimize } f(x), \quad \text{subject to } h(x) = b, \quad x \in X.$$

Let  $x \in X(b) = \{x \in X : h(x) = b\}$ . We say that  $x$  is **feasible** if  $x \in X(b)$ . Define the **Lagrangian** as

$$L(x, \lambda) = f(x) - \lambda^\top (h(x) - b).$$

Typically,  $X \subseteq \mathbb{R}^n$ ,  $h : \mathbb{R}^n \mapsto \mathbb{R}^m$ , with  $b, \lambda \in \mathbb{R}^m$ . Here  $\lambda$  is called a **Lagrangian multiplier**.

**Theorem 1.1 (Lagrangian Sufficiency Theorem)** *If  $\bar{x}$  is feasible for  $P(b)$  and there exists  $\bar{\lambda}$  such that*

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

*then  $\bar{x}$  is optimal for  $P(b)$ .*

**Proof of the LST.** For all  $x \in X(b)$  and  $\lambda$  we have

$$f(x) = f(x) - \lambda^\top (h(x) - b) = L(x, \lambda).$$

Now  $\bar{x} \in X(b) \subseteq X$  and so by assumption,

$$f(\bar{x}) = L(\bar{x}, \bar{\lambda}) \leq L(x, \bar{\lambda}) = f(x), \quad \text{for all } x \in X(b).$$

Thus  $\bar{x}$  is optimal for the optimization problem  $P(b)$ . ■

**Example 1.1** Minimize  $x_1^2 + x_2^2$  subject to  $a_1x_1 + a_2x_2 = b$ ,  $x_1, x_2 \geq 0$ .

Here  $L = x_1^2 + x_2^2 - \lambda(a_1x_1 + a_2x_2 - b)$ . We consider the problem

$$\text{minimize}_{x_1, x_2 \geq 0} [x_1^2 + x_2^2 - \lambda(a_1x_1 + a_2x_2 - b)].$$

This has a stationary point where  $(x_1, x_2) = (\lambda a_1/2, \lambda a_2/2)$ . Now we choose  $\lambda$  such that  $a_1x_1 + a_2x_2 = b$ . This happens for  $\lambda = 2b/(a_1^2 + a_2^2)$ . We have a minimum since  $\partial^2 L / \partial x_i^2 > 0$ ,  $\partial^2 L / \partial x_1 \partial x_2 = 0$ . Thus with this value of  $\lambda$  the conditions of the LST are satisfied and the optimal value is  $b^2/(a_1^2 + a_2^2)$  at  $(x_1, x_2) = (a_1b, a_2b)/(a_1^2 + a_2^2)$ .

## 1.2 The Dual Problem

Let us define

$$\phi(b) = \inf_{x \in X(b)} f(x) \quad \text{and} \quad g(\lambda) = \inf_{x \in X} L(x, \lambda).$$

Then for all  $\lambda$

$$\phi(b) = \inf_{x \in X(b)} L(x, \lambda) \geq \inf_{x \in X} L(x, \lambda) = g(\lambda). \quad (1.1)$$

Thus  $g(\lambda)$  is a lower bound on  $\phi(b)$ , i.e., a lower bound on the solution value of  $P(b)$ . As this holds for all  $\lambda$  it is interesting to make this lower bound as large as possible. Of course we can restrict ourselves to  $\lambda$  for which  $g(\lambda) > -\infty$ . This motivates the **dual problem**, defined as

$$\text{maximize } g(\lambda), \quad \text{subject to } \lambda \in Y,$$

where  $Y = \{\lambda : g(\lambda) > -\infty\}$ . In (1.1) we have a proof of the so-called **weak duality theorem** that

$$\inf_{x \in X(b)} f(x) \geq \max_{\lambda \in Y} g(\lambda). \quad (1.2)$$

The left hand side of (1.2) poses the **primal problem**.

## 1.3 Strong Lagrangian

We say that  $P(b)$  is **Strong Lagrangian** if there exists  $\lambda$  such that

$$\phi(b) = \inf_{x \in X} L(x, \lambda). \quad (1.3)$$

In other words,  $P(b)$  is Strong Lagrangian if it can be solved by the Lagrangian method. But when does this happen? Usually we just try the method and see. If we are lucky, as in Example 1.1, then we will be able to establish that there exists a  $\lambda$  that lets us solve  $P(b)$  this way. However, there are important classes of problems for which we can guarantee that Lagrangian methods always work.

Note that (1.3) says that there is a  $\lambda$  such that  $\phi(b) = g(\lambda)$ . Combining this with (1.2), we see that if the problem is Strong Lagrangian then  $\min$  of primal =  $\max$  of dual.

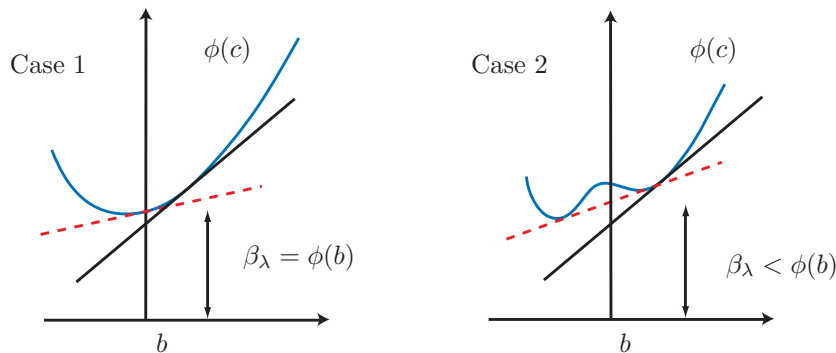
## 1.4 Hyperplanes

Let the hyperplane  $(c, \alpha)$  be given by

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

It has intercept at  $\beta$  on the vertical axis through  $b$ , and has slope(s)  $\lambda$ . Consider the following approach to finding  $\phi(b)$ :

1. For each  $\lambda$ , find  $\beta_\lambda \equiv \max \beta$  such that the hyperplane lies completely below the graph of  $\phi$ .
2. Now choose  $\lambda$  to maximize  $\beta_\lambda$ .



Lagrangian methods work in Case 1 because of the existence of a tangent to  $\phi$  at  $b$ . Define a **supporting hyperplane**  $(c, \alpha)$  at  $b$  as

$$\alpha = \phi(b) - \lambda^\top (b - c), \text{ where } \phi(c) \geq \phi(b) - \lambda^\top (b - c) \text{ for all } c \in \mathbb{R}^m.$$

In fact,  $\beta_\lambda = g(\lambda) = \min_{x \in X} L(x, \lambda)$ . To see this, we argue

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

Hence, the dual problem is  $\max \beta_\lambda$ . Again, we see the weak duality result of  $\max \beta_\lambda \leq \phi(b)$ , with equality if the problem is **Strong Lagrangian**.

**Theorem 1.2** *The following are equivalent:*

- (a) there exists a (non-vertical) supporting hyperplane to  $\phi$  at  $b$ ;
- (b) the problem is Strong Lagrangian.

This is important since a (non-vertical) supporting hyperplane exists if  $\phi(b)$  is a convex function of  $b$ . We can find conditions that make  $\phi$  convex.

**Proof.** Suppose there exists a (non-vertical) supporting hyperplane to  $\phi$  at  $b$ . This means that there exists  $\lambda$  such that

$$\phi(b) - \lambda^\top (b - c) \leq \phi(c) \text{ for all } c \in \mathbb{R}^m.$$

This implies

$$\begin{aligned} \phi(b) &\leq \inf_{c \in \mathbb{R}^m} [\phi(c) - \lambda^\top (c - b)] \\ &= \inf_{c \in \mathbb{R}^m} \inf_{x \in X(c)} [f(x) - \lambda^\top (h(x) - b)] \\ &= \inf_{x \in X} L(x, \lambda) \\ &= g(\lambda) \end{aligned}$$

However, we have the opposite inequality in (1.1). Hence  $\phi(b) = g(\lambda)$ . This means that  $P(b)$  is Strong Lagrangian, i.e., can be solved by minimizing  $L(x, \lambda)$  with respect to  $x$ .

Conversely, if the problem is Strong Lagrangian then there exists  $\lambda$  such that for all  $x \in X$

$$\phi(b) \leq f(x) - \lambda^\top (h(x) - b).$$

Imagine minimizing the right hand side over  $x \in X(c)$ , where  $h(x) = c$ . This gives

$$\phi(b) \leq \phi(c) - \lambda^\top (c - b).$$

This is true for all  $c$ , and hence

$$\phi(b) - \lambda^\top (b - c) \leq \phi(c) \text{ for all } c \in \mathbb{R}^m.$$

Hence,  $\phi$  has a (non-vertical) supporting hyperplane at  $b$ . ■

## 2 Linear Programming

### 2.1 Convexity and Lagrangian methods

1. A set  $S$  is a **convex set** if for all  $0 \leq \delta \leq 1$

$$x, y \in S \implies \delta x + (1 - \delta)y \in S.$$

2. A real-valued  $f$  is a **convex function** if for all  $x, y \in S$  and  $0 \leq \delta \leq 1$

$$\delta f(x) + (1 - \delta)f(y) \geq f(\delta x + (1 - \delta)y).$$

3. A point  $x$  is an **extreme point** of  $S$  if whenever

$$x = \delta y + (1 - \delta)z$$

for some  $y, z \in S$  and  $0 < \delta < 1$  then  $x = y = z$ .

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

So, we are interested in conditions on the problem that make  $\phi$  convex.

**Theorem 2.2** *Consider the problem  $P(b)$ , defined as*

$$\underset{x \in X}{\text{minimize}} f(x) \quad \text{subject to} \quad h(x) \leq b.$$

*If  $X$  is a convex set and  $f$  and  $h$  are convex then  $\phi$  is convex.*

**Proof.** Take  $b_1, b_2$  and  $b = \delta b_1 + (1 - \delta)b_2$  for  $0 < \delta < 1$  with  $b_1, b_2$  such that  $\phi$  is defined. Take  $x_i$  feasible for  $P(b_i)$  for  $i = 1, 2$  and consider  $x = \delta x_1 + (1 - \delta)x_2$ . Then  $X$  convex,  $x_1, x_2 \in X$  implies that  $x \in X$ . Also,  $h$  convex gives

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

So  $x$  is feasible for  $P(b)$ . So, if  $f$  is convex

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

This holds for all  $x_1 \in X(b_1)$  and  $x_2 \in X(b_2)$  so taking infimums gives

$$\phi(b) \leq \delta \phi(b_1) + (1 - \delta)\phi(b_2)$$

so that  $\phi$  is convex. ■

**Remark.** Consider the constraint  $h(x) = b$ . This is the same as  $h(x) \leq b$  and  $-h(x) \leq -b$ . So  $\phi$  is convex under these constraints if  $X$  is a convex set and  $f, h$  and  $-h$  are all convex. Thus  $h$  should be linear in  $x$ .

### 2.2 Linear programs

We will study problems of the form<sup>1</sup>

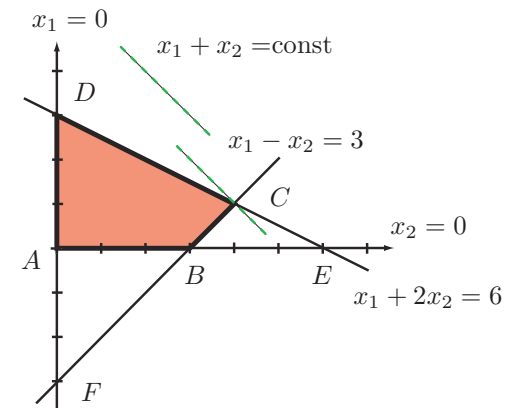
$$\text{minimize } \{c^\top x : Ax \leq b, x \geq 0\}$$

where  $x$  and  $c$  are  $n$ -vectors,  $b$  is a  $m$ -vector and  $A$  is a  $m \times n$  matrix. Such problems are also written out in the longer form

$$\text{minimize } c^\top x, \quad \text{subject to } Ax \leq b, x \geq 0.$$

**Example**

$$\begin{aligned} &\text{minimize } -(x_1 + x_2) \\ &\text{subject to} \\ &x_1 + 2x_2 \leq 6 \\ &x_1 - x_2 \leq 3 \\ &x_1, x_2 \geq 0. \end{aligned}$$



### 2.3 Duality of linear programs

The primal LP optimization problems

$$(LP =) : \quad \text{minimize} \{c^\top x : Ax = b, x \geq 0\}$$

$$(LP \geq) : \quad \text{minimize} \{c^\top x : Ax \geq b, x \geq 0\}$$

<sup>1</sup>For a thorough introduction to the topic of linear programming see Richard Weber's course on Optimization, available at: <http://www.statslab.cam.ac.uk/~rrw1/opt/>

have corresponding dual problems

$$\text{Dual of } (LP =) : \quad \text{maximize}\{b^\top \lambda : A^\top \lambda \leq c\}$$

$$\text{Dual of } (LP \geq) : \quad \text{maximize}\{b^\top \lambda : A^\top \lambda \leq c, \lambda \geq 0\}$$

## 2.4 Derivation of the dual LP problem

Consider  $(LP \geq)$ , and introduce slack variables  $z$  to form the problem

$$\text{minimize } c^\top x, \quad \text{subject to } Ax - z = b, \quad x \geq 0, \quad z \geq 0.$$

So the set  $X \subset \mathbb{R}^{m+n}$  is given by

$$X = \{(x, z) : x \geq 0, z \geq 0\}.$$

We use a Lagrangian approach. The **Lagrangian** is

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

with finite minimum over  $(x, z) \in X$  if and only if

$$\lambda \in Y = \{\lambda : \lambda \geq 0, c^\top - \lambda^\top A \geq 0\}.$$

The minimum of  $L((x, z); \lambda)$  for  $\lambda \in Y$  is attained where both  $(c^\top - \lambda^\top A)x = 0$  and  $\lambda^\top z = 0$ , so that

$$g(\lambda) \equiv \inf_{x \in X} L(x; \lambda) = \lambda^\top b.$$

Hence form of **dual problem**.

## 2.5 Shadow prices

The Lagrange multipliers play the role of **prices** since we have that

$$\frac{\partial \phi}{\partial b_i} = \frac{\partial g(\lambda)}{\partial b_i} = \lambda_i.$$

The variables  $\lambda_i$  are also known as **shadow prices**.

## 2.6 Conditions for optimality

For the  $(LP \geq)$  problem,  $x$  and  $\lambda$  are primal and dual optimal respectively if and only if  $x$  is primal feasible,  $\lambda$  is dual feasible and, in addition, for any  $i = 1, \dots, n$  and  $j = 1, \dots, m$

$$(c^\top - \lambda^\top A)_i x_i = 0 = \lambda_j (Ax - b)_j.$$

These are known as the **complementary slackness conditions**.

## 2.7 Basic insight

*If an LP has a finite optimum then it has an optimum at an extreme point of the feasible set.*

There are a finite number of extreme points so an algorithm for solving the LP is

- Find all the vertices of the feasible set.
- Pick the best one.

Our example has an optimum at  $C$ . However, there are  $\binom{n+m}{m}$  vertices, so this algorithm could take a long time!

## 2.8 Basic solutions

A **basic solution** to  $Ax = b$  is a solution with at least  $n - m$  zero variables. The solution is **non-degenerate** if exactly  $n - m$  variables are zero. The choice of the  $m$  non-zero variables is called the **basis**. Variables in the basis are called **basic**; the others are called **non-basic**.

If a basic solution satisfies  $x \geq 0$  then it is called a **basic feasible solution** (bfs). The following is a theorem.

**The basic feasible solutions are the extreme points of the feasible set.**

In our example, the vertices  $A-F$  are basic solutions (and non-degenerate) and  $A-D$  are basic feasible solutions.

### 3 The Simplex Algorithm

1. Start with a bfs.
2. Test whether this bfs is optimal.
3. If YES then stop.
4. If NO then move to an ‘adjacent’ bfs which is better. Return to step 2.

#### 3.1 Algebraic viewpoint

A **basis**,  $B$ , is a choice of  $m$  non-zero variables. For any  $x$  satisfying the constraints  $Ax = b$ , we can write

$$A_B x_B + A_N x_N = b$$

where  $A_B$  is a  $m \times m$  matrix,  $A_N$  is a  $m \times (n - m)$  matrix,  $x_B$  and  $b$  are  $m$ -vectors and  $x_N$  is a  $(n - m)$ -vector.

A **basic solution** has  $x_N = 0$  and  $A_B x_B = b$  and a **basic feasible solution** has  $x_N = 0$ ,  $A_B x_B = b$  and  $x_B \geq 0$ .

As we have seen, if there exists a finite optimum then there exists a bfs that is optimal.

#### Nondegeneracy assumptions

We will assume that the following assumptions hold. (If they do not, then they will do so for a small perturbation of the data).

1. The matrix  $A$  has linearly independent rows, i.e.,  $\text{rank}(A) = m$ .
2. Any  $m \times m$  matrix formed from  $m$  columns of  $A$  is non-singular.
3. All basic solutions  $A_B x_B = b$ ,  $x_N = 0$  have exactly  $m$  non-zero variables, i.e.,  $x_i \neq 0$  for  $i \in B$ .

#### 3.2 Simplex tableau

Now for any  $x$  with  $Ax = b$ , we have  $x_B = A_B^{-1}(b - A_N x_N)$ . Hence,

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

We can assemble this information in a **tableau**.

basic	non-basic	
$I$	$A_B^{-1} A_N$	$A_B^{-1} b$
0	$c_N^\top - c_B^\top A_B^{-1} A_N$	$-c_B^\top A_B^{-1} b$

#### 3.3 Test for optimality

Suppose we want to maximize  $c^\top x$  and we find

$$(c_N^\top - c_B^\top A_B^{-1} A_N) \leq 0 \quad \text{and} \quad A_B^{-1} b \geq 0.$$

Then for all feasible  $x$  (since  $x \geq 0 \implies x_N \geq 0$ )

$$f(x) = c_B^\top A_B^{-1} b + (c_N^\top - c_B^\top A_B^{-1} A_N) x_N \leq c_B^\top A_B^{-1} b.$$

But for bfs  $\hat{x}$  with  $\hat{x}_B = A_B^{-1} b$  and  $\hat{x}_N = 0$  we have  $f(\hat{x}) = c_B^\top A_B^{-1} b$ . So,  $\hat{x}$  is optimal.

This gives us an easy way to check if a given bfs is optimal.

#### 3.4 Choice of new bfs

Alternatively, if some  $(c_N^\top - c_B^\top A_B^{-1} A_N)_i$  is positive we can increase the value of the objective function by increasing from zero the value of  $(x_N)_i$ .

We would like to increase  $(x_N)_i$  by as much as possible. However, we need to keep the constraints satisfied. So as we alter  $(x_N)_i$  the other variables alter and we must stop increasing  $(x_N)_i$  if one becomes zero.

The net effect is that we interchange one basic variable with one non-basic variable.

#### 3.5 Simplex algorithm

1. Find an initial bfs with basis  $B$ .
2. Check the sign of  $(c_N^\top - c_B^\top A_B^{-1} A_N)_i$  for  $i \in N$ . Are all components non-positive?
3. If YES then we are at an optimum. Stop.
4. If NO, so that  $(c_N^\top - c_B^\top A_B^{-1} A_N)_i > 0$ , say with  $i \in N$ , increase  $(x_N)_i$  as much as possible.

**Either** we can do this indefinitely, which means the maximum is unbounded. Stop.

**or** one of  $x_B$  variables becomes zero, giving a new new bfs. Repeat from step 2.

### 3.6 Simplex algorithm: tableau form

#### 0. Find an initial basic feasible solution.

The tableau takes the form

$(a_{ij})$	$a_{i0}$
$a_{0j}$	$a_{00}$

This is easy when the constraints are of the form

$$Ax \leq b, \quad b \geq 0.$$

We can write this as

$$Ax + z = b, \quad z \geq 0$$

and take an initial basic feasible solution of

$$x = 0, \quad z = b \geq 0.$$

It is best to think of this as extending  $x$  to  $(x, z)$  and then setting

$$(x_B, x_N) = (z, x) = (b, 0).$$

#### 1. Choose a variable to enter the basis

Look for a  $j$  such that  $a_{0j} > 0$ . Column  $j$  is called the **pivot column** and the variable corresponding to column  $j$  will enter the basis. If  $a_{0j} \leq 0$  for all  $j \geq 1$  then the current solution is optimal. If there is more than one  $j$  such that  $a_{0j} > 0$  choose any one. A common rule-of-thumb is to choose the  $j$  for which  $a_{0j}$  is most positive. Alternatively, we could choose the least  $j \geq 1$  for which  $a_{0j} > 0$ .

#### 2. Find the variable to leave the basis

Choose  $i$  to minimize  $a_{i0}/a_{ij}$  from the set  $\{i : a_{ij} > 0\}$ . Row  $i$  is called the **pivot row** and  $a_{ij}$  is called the **pivot**. If  $a_{ij} \leq 0$  for all  $i$  then the problem is unbounded and the objective function can be increased without limit.

If there is more than one  $i$  minimizing  $a_{i0}/a_{ij}$  the problem has a degenerate basic feasible solution.

In our example we have at this point

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

#### 3. Pivot on the element $a_{ij}$

The purpose of this step is to get the equations into the appropriate form for the new basic feasible solution.

- Multiply row  $i$  by  $1/a_{ij}$ .
- Add  $-(a_{kj}/a_{ij}) \times (\text{row } i)$  to each row  $k \neq i$ , including the objective function row.

The new tableau form: (after re-arranging rows and columns), is as at the end of Section 3.6. In our example we reach

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

Now return to Step 1.

In our example, one further iteration brings us to the optimum.

	$x_1$	$x_2$	$z_1$	$z_2$	$a_{i0}$
$x_2$ basic	0	1	$\frac{1}{3}$	$-\frac{1}{3}$	1
$x_1$ basic	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  $x_1 = 4, x_2 = 1, z_1 = z_2 = 0$ , i.e., vertex  $C$ .

## 4 Advanced Simplex Procedures

### 4.1 Two phase simplex method

Suppose we do not have the obvious basic feasible solution. Consider

$$\begin{array}{ll}
 \text{maximize} & -6x_1 - 3x_2 \\
 \text{subject to} & \\
 x_1 + x_2 & \geq 1 \\
 2x_1 - x_2 & \geq 1 \\
 3x_2 & \leq 2 \\
 x_1, x_2 & \geq 0
 \end{array}
 \quad \equiv \quad
 \begin{array}{ll}
 \text{maximize} & -6x_1 - 3x_2 \\
 \text{subject to} & \\
 x_1 + x_2 - z_1 & = 1 \\
 2x_1 - x_2 - z_2 & = 1 \\
 3x_2 + z_3 & = 2 \\
 x_1, x_2, z_1, z_2, z_3 & \geq 0
 \end{array}$$

Unfortunately, the basic solution

$$x_1 = 0 \quad x_2 = 0 \quad z_1 = -1 \quad z_2 = -1 \quad z_3 = 2$$

is not feasible. The trick is to add **artificial variables**,  $y_1, y_2$  to the constraints and then minimize  $y_1 + y_2$  subject to

$$\begin{array}{l}
 x_1 + x_2 - z_1 + y_1 = 1 \\
 2x_1 - x_2 - z_2 + y_2 = 1 \\
 3x_2 + z_3 = 2 \\
 x_1, x_2, z_1, z_2, z_3, y_1, y_2 \geq 0
 \end{array}$$

We can take the ‘easy’ initial bfs of  $y_1 = 1, y_2 = 1, z_3 = 2, x_1 = 0, x_2 = 0$ .

In **Phase I** we minimize  $y_1 + y_2$ , starting with  $y_1 = 1, y_2 = 1$  and  $z_3 = 2$ . (Notice we did not need an artificial variable in the third equation.) Provided the original problem is feasible we should be able to obtain a minimum of 0 with  $y_1 = y_2 = 0$  (since  $y_1$  and  $y_2$  are not needed to satisfy the constraints if the original problem is feasible). At the end of Phase I the simplex algorithm will have found a bfs for the original problem. **Phase II** proceeds with the solution of the original problem, starting from this bfs.

Note: the original objective function doesn’t enter into Phase I, but it is useful to carry it along as an extra row in the tableau since the algorithm will then arrange for it to be in the appropriate form to start Phase II.

We start with

	$x_1$	$x_2$	$z_1$	$z_2$	$z_3$	$y_1$	$y_2$	
$y_1$	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
Phase II	-6	-3	0	0	0	0	0	0
Phase I	0	0	0	0	0	-1	-1	0

**Preliminary step.** The Phase I objective must be written in terms of the non-basic variables. This is accomplished by adding rows 1 and 2 to the bottom row, to give

	$x_1$	$x_2$	$z_1$	$z_2$	$z_3$	$y_1$	$y_2$	
$y_1$	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
Phase II	-6	-3	0	0	0	0	0	0
Phase I	3	0	-1	-1	0	0	0	2

**Begin Phase I.** Pivot on  $a_{21}$  to get

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

Pivot on  $a_{14}$  to get

	$x_1$	$x_2$	$z_1$	$z_2$	$z_3$	$y_1$	$y_2$	
$z_2$	0	3	-2	1	0	2	-1	1
$x_1$	1	1	-1	0	0	1	0	1
$z_3$	0	3	0	0	1	0	0	2
	0	3	-6	0	0	6	0	6
	0	0	0	0	0	-1	-1	0

**End of Phase I.**  $y_1 = y_2 = 0$  and we no longer need these variables, and so drop the last two columns and Phase I objective row. We have a bfs with which to start Phase II, with  $x_1 = 1, z_2 = 1, z_3 = 2$ . The rest of the tableau is already in appropriate form. So we rewrite the preceding tableau without the  $y_1, y_2$  columns.

**Begin Phase II.**

	$x_1$	$x_2$	$z_1$	$z_2$	$z_3$	
$z_2$	0	3	-2	1	0	1
$x_1$	1	1	-1	0	0	1
$z_3$	0	3	0	0	1	2
	0	3	-6	0	0	6

In one more step we reach the optimum, by pivoting on  $a_{12}$ .

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

In general, artificial variables are needed when there are constraints like

$$\leq -1, \text{ or } \geq 1, \text{ or } = 1,$$

unless they happen to be of a special form for which it is easy to spot a bfs. If the Phase I objective cannot be minimized to zero then the original problem is infeasible.

The problem we have solved is the dual of the problem P that we considered in Chapters 2–3, augmented by the constraint  $3x_2 \leq 2$ . It is interesting to compare the final tableau above with the tableau obtained in solving the primal. They are essentially transposes of one another.

## 4.2 Primal and dual algorithms

Consider the problem ( $LP =$ ), defined as  $\text{minimize}\{c^\top x : Ax = b, x \geq 0\}$ . This has dual  $\text{maximize}\{\lambda^\top b : c^\top - \lambda^\top A \geq 0\}$ . At each stage of the primal simplex algorithm, we have a tableau,

basic, $x_B \geq 0$	non-basic, $x_N = 0$	
$I$	$A_B^{-1}A_N$	$A_B^{-1}b \geq 0$
$c_B^\top - c_B^\top A_B^{-1}A_B = 0$	$c_N^\top - c_B^\top A_B^{-1}A_N$ , free	$-c_B^\top A_B^{-1}b$

Here we have a basic feasible solution for the primal,  $x_B = A_B^{-1}b$ , and a basic (though not necessarily feasible) solution for the dual,  $\lambda_B^\top = c_B^\top A_B^{-1}$ . We always have primal feasibility and complementary slackness. Recall

primal feasibility $Ax = b$ and $x \geq 0$	+	dual feasibility $(c^\top - \lambda^\top A) \geq 0$	+	complementary slackness $(c^\top - \lambda^\top A)x = 0$
--	---	---	---	--

$\implies$  optimality.

**Primal algorithms** maintain primal feasibility and complementary slackness and seek dual feasibility. **Dual algorithms** maintain dual feasibility and complementary slackness and seek primal feasibility.

## 4.3 Dual simplex algorithm

The **dual simplex algorithm** starts with and maintains a primal/dual basic solution that is dual feasible and satisfies complementary slackness while seeking primal feasibility. This can be useful.

**It may be easier to spot a dual feasible solution**

$$\begin{aligned} \text{minimize } 2x_1 + 3x_2 + 4x_3 \quad \text{s.t.} \quad & x_1 + 2x_2 + x_3 \geq 3 \\ & 2x_1 - x_2 - 3x_3 \geq 4 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Note  $c_i \geq 0$  for all  $i$ . Let us add slack variables,  $z_i \geq 0$  to obtain

$$\begin{aligned} x_1 + 2x_2 + x_3 - z_1 &= 3 \\ 2x_1 - x_2 - 3x_3 - z_2 &= 4 \end{aligned}$$

The primal algorithm must use two-phases since  $z_1 = -3, z_2 = -4$  is not primal feasible. However, the tableau contains a dual feasible solution,  $\lambda_1 = \lambda_2 = 0$ , and  $c^\top - \lambda^\top A = (2, 3, 4, 0, 0) \geq 0$ .

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

**Rule:** for rows,  $i$ , with  $a_{i0} < 0$  pick column  $j$  with  $a_{ij} < 0$  to minimize  $a_{0j}/-a_{ij}$ . Pivoting on  $a_{21}$  gives

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  $a_{12}$  gives

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



So the optimum is  $\frac{28}{5}$ , with  $x_1 = \frac{11}{5}$ ,  $x_2 = \frac{2}{5}$ ,  $x_3 = 0$ .

Notice that for problems of the form  $Ax \geq b$  we can write

$$Ax - z = b \quad z \geq 0$$

$$A \begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot & -1 & 0 \\ \cdot & \cdot & \cdot & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} = b$$

Hence

$$(c^T - \lambda^T A) = \begin{pmatrix} 2 & 3 & 4 & 0 & 0 \end{pmatrix} - (\lambda_1 \quad \lambda_2) \begin{pmatrix} \cdot & \cdot & \cdot & -1 & 0 \\ \cdot & \cdot & \cdot & 0 & -1 \end{pmatrix}$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \lambda_1 & \lambda_2 \end{pmatrix}$$

So the dual variables,  $\lambda$ , can be found in the objective function row under the slack variables in the optimal tableau. E.g.,  $\lambda = (\frac{8}{5}, \frac{1}{5})$ .

### We may wish to add constraints to optimal solutions

Suppose we have solved an LP and have the final tableau

non-basic	basic	
	$I$	+ve
+ve	0	

Now we wish to add a new constraint

$$a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b.$$

If the optimal solution satisfies this constraint the solution remains optimal for the new problem. Otherwise, we add it to the tableau to give

	$I$	0	+ve
		$\vdots$	
		0	
$a_N$	$a_B$	1	$b$
+ve	0	0	

 $\longrightarrow$ 

	$I$	0	+ve
		$\vdots$	
		0	
	0	1	-ve?
+ve	0	0	

Notice that we still have a dual feasible solution. The problem solution may not be primal feasible. However, we can apply the dual simplex algorithm to find the new optimum under this additional constraint.

### Gomory's cutting plane method

The addition of new constraints is useful in **Gomory's cutting plane method** for integer linear programming. Consider the final tableau on page 16. Pick a row in which the far right hand element is not an integer, say row 1. This says

$$x_2 + x_3 - \frac{2}{5}z_1 + \frac{1}{5}z_2 = \frac{2}{5}.$$

Suppose  $x_1, x_2, x_3, z_1, z_2$  are restricted to be non-negative integers. Then we must have

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

This is because  $z_1, z_2 \geq 0$  and we have replaced  $-\frac{2}{5}$  and  $\frac{1}{5}$  by the integers that lie just below them.

Since the left hand side is an integer, it can be no more than the integer just below  $\frac{2}{5}$ . So a solution in integers must satisfy

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

However, the present solution does not satisfy this, since  $x_2 = \frac{2}{5}$ ,  $x_3 = z_1 = z_2 = 0$ . Thus we can add this new constraint (or cutting plane) to give

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

which is written into standard form

0	1	1	$-\frac{2}{5}$	$\frac{1}{5}$	0	$\frac{2}{5}$
1	0	-2	$-\frac{1}{5}$	$-\frac{2}{5}$	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	$-\frac{28}{5}$

Applying the dual simplex algorithm to this, and repeating the process, we will eventually arrive at the optimal integer solution. In this example we reach the optimum of  $x_1 = 3$ ,  $x_2 = x_3 = 0$  in just one more iteration.

0	1	1	-1	0	1	0
1	0	-2	1	0	-2	3
0	0	0	3	1	-5	2
0	0	3	1	0	1	-6

## 5 Complexity of Algorithms

### 5.1 Theory of algorithmic complexity

An **instance** of an optimization problem is defined by its input data. E.g., an instance of linear programming with  $n$  variables and  $m$  constraints is described by the inputs  $c$ ,  $A$  and  $b$ . There are  $mn + m + n$  numbers and if all of them can be expressed in no more than  $k$  bits, the instance can be described in a string of  $(mn + m + n)k$  bits. This is the **instance size**.

An optimization problem is solved by a computational **algorithm** whose **running time** depends on how it is programmed and the speed of the hardware. A large instance can be easy to solve, such as LP with  $A = I$ . However, in general, we expect an algorithm's running time to increase with size of the instance. Ignoring details of the implementation, the running time depends on the number of arithmetic operations involved. For example, the linear system  $Ax = b$ , with  $A$  being  $n \times n$ , can be solved by the algorithm of Gaussian elimination, using  $O(n^3)$  operations of addition, subtraction, multiplication and division. We define

- $f(n) = O(g(n))$  if there exists a  $c$  such that  $f(n) \leq cg(n)$  for all  $n$ .
- $f(n) = \Omega(g(n))$  if there exists a  $c$  such that  $f(n) \geq cg(n)$  for all  $n$ .
- $f(n) = \Theta(g(n))$  if  $f(n)$  is both  $O(g(n))$  and  $\Omega(g(n))$ .

Of course multiplication is more difficult than addition and so in computing running time we might count operations according more elementary computer instructions. In what follows we make use of Turing's famous proof that the class of things that can be computed is the class things that can be computed by a Deterministic Turing Machine (DTM). A DTM is essentially a finite-state machine that can read and write to an external storage medium.

When a DTM is given an input  $x$  it runs for some number of steps and then outputs an answer,  $f(x)$ . This number of steps is its running time. There are many Turing machines. Let  $T_M(n)$  be the **worst-case** running time of some Turing machine, say  $M$ , over inputs  $x$  of size  $|x| = n$ . We say that a function  $f(x)$  is **computable in polynomial time** if there exists some Turing machine that can compute  $f(x)$  within  $|x|^k$  steps (for some fixed  $k$ ). The definition is robust, since different Turing machines can simulate one another and more efficient models of computation, by at most squaring or cubing the the computation time. In contrast, if  $T_M(n) = \Omega(2^{cn})$  for all  $M$ , then  $f(x)$  is said to be computable in **exponential time**.

### 5.2 The Travelling Salesman Problem

Given a finite set of points  $S$  in the plane, the TSP asks for the shortest tour of  $S$ . More formally, given  $S = \{s_1, s_2, \dots, s_n\}$  a shortest tour that visits all points of  $S$  is specified by a permutation  $\sigma$  that minimizes the sum of distances

$$d(s_{\sigma(1)}, s_{\sigma(2)}) + d(s_{\sigma(2)}, s_{\sigma(3)}) + \dots + d(s_{\sigma(n-1)}, s_{\sigma(n)}) + d(s_{\sigma(n)}, s_{\sigma(1)})$$

where  $d(s_i, s_j)$  is the distance between points  $i$  and  $j$ .

In general, it is difficult to prove that a problem does not have a polynomial time algorithm. No polynomial time algorithm is known for TSP. But there is also no proof that a polynomial time algorithm for TSP does not exist. We see in Lecture 6 that the simplex algorithm for LP is an exponential time algorithm. It was not until the 1970s that a polynomial time algorithm was discovered for LP.

### 5.3 Decision Problems

A decision problem (or recognition problem) is one that takes the form of a question with a *yes/no* answer. For example, decision-TSP is

$$\text{Given the points } S, \text{ and } L \text{ is there a tour of length } \leq L? \quad (5.1)$$

This differs from optimization-TSP: *find the shortest tour*, or the evaluation-TSP: *find the length of the shortest tour*. Of course the three types of problem are closely related. We focus on decision problems. This avoids problems in which the size of the input or output causes non-polynomial time behaviour.

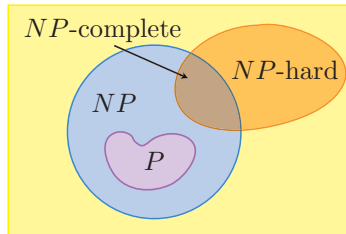
### 5.4 $\mathcal{P}$ and $\mathcal{NP}$ problems

A decision problem is in  $\mathcal{P}$  if its answer is computable in polynomial time. I.e., there exists a deterministic Turing machine which, given any instance (with input data  $x$ ), can compute the answer within a number of steps bounded by  $|x|^k$  (for some fixed  $k$ ).

A decision problem belongs to  $\mathcal{NP}$  if there exists a checking function  $r(x, y)$  such that the answer to the decision problem is *yes* iff there exists a  $y$  (called a certificate) such that  $r(x, y) = 1$  and  $r(x, y)$  is computable in polynomial time. For example, if the answer to (5.1) is *yes* then  $y$  could be the order in which the points should be visited. It takes only time  $O(n)$  to add up the length of this tour and check it is less than  $L$  (this being the computation  $r(x, y)$ ).

' $\mathcal{NP}$ ' stands for **nondeterministic polynomial**. An equivalent definition of  $\mathcal{NP}$  is that it is the class of decision problems whose answers can be computed in polynomial time on a 'nondeterministic Turing machine' (NDTM). A NDTM

consists of many DTMs working in parallel, any one of which can answer *yes* in polynomial time without consulting the others. Essentially, these computers are carrying out parallel calculations of  $r(x, y)$  for all possible  $y$ . Either one of them produces the answer *yes* within time  $n^k$ , or the answer is *no*. A NDTM for (5.1) could consist of  $(n - 1)!$  DTMs, each of them checking one possible tour to see if its length is less than  $L$ . Clearly,  $\mathcal{P} \subseteq \mathcal{NP}$ . It is believed that  $\mathcal{P} \subset \mathcal{NP}$ : that is, there are problems in  $\mathcal{NP}$  which are not in  $\mathcal{P}$ . However, this is a major unsolved problem.



## 5.5 Polynomial reduction

When is problem  $\Pi_1$  no harder than another problem  $\Pi_2$ ? We say that  $\Pi_1$  **reduces to**  $\Pi_2$  if we can construct an algorithm for  $\Pi_1$  as follows.

1. Make a (polynomial time) transformation of the instance of  $\Pi_1$  into an instance of  $\Pi_2$ .
2. Apply some algorithm to solve this instance of  $\Pi_2$ .
3. Make a (polynomial time) transformation of this solution of  $\Pi_2$  back into a solution of  $\Pi_1$ .

The idea is that we can use an algorithm that solves  $\Pi_2$  to solve  $\Pi_1$ , with additional work in steps 1 and 3 that requires at most polynomial time. Thus  $\Pi_1$  is really no harder than  $\Pi_2$ .

## 5.6 $\mathcal{NP}$ -completeness

Now we can talk about the hardest problems in  $\mathcal{NP}$ . A problem  $\Pi$  is said to be  $\mathcal{NP}$ -hard if every problem in  $\mathcal{NP}$  can be reduced to it. It is said to be  $\mathcal{NP}$ -complete if moreover  $\Pi \in \mathcal{NP}$ . Thus all  $\mathcal{NP}$ -complete problems can be reduced to one another and are as difficult as all problems in  $\mathcal{NP}$ .

There are many  $\mathcal{NP}$ -complete problems. LP in which all variable are restricted to be 0 or 1 is  $\mathcal{NP}$ -complete. TSP is  $\mathcal{NP}$ -complete. So all problems in  $\mathcal{NP}$  are no

harder than either of these problems. If you can find a polynomial time algorithm for TSP then you have found a polynomial time algorithm for all problems in  $\mathcal{NP}$  and it would be true that  $\mathcal{P} = \mathcal{NP}$ . As we said, since no one has ever found a polynomial time algorithm for any  $\mathcal{NP}$ -complete problem, it is believed that  $\mathcal{P} \neq \mathcal{NP}$ . To show that a new problem,  $\Pi$ , is  $\mathcal{NP}$ -complete we must (i) show that  $\Pi \in \mathcal{NP}$ , and (ii) show that a known  $\mathcal{NP}$ -complete problem reduces to  $\Pi$ .

## 5.7 Examples of $\mathcal{NP}$ -complete problems

**Satisfiability (Cook (1971))** Given a propositional formulae with AND's, NOT's, OR's and Boolean (T or F) variables  $X_1, X_2, \dots, X_r$ , for example,

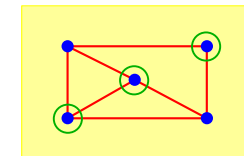
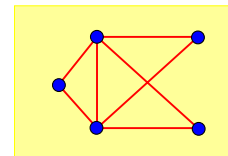
$$(X_1 \text{ OR NOT } X_2) \text{ AND } (X_3 \text{ AND } X_4)$$

is there an assignment of truth values to the variables that makes the formulae true? (e.g.  $X_1 = X_2 = X_3 = X_4 = T$  in the above example.)

**Hamiltonian circuit** Given a graph  $G$ . Is there a set of edges forming a tour of all the vertices? To see that an instance of this is no harder than as TSP, think of a TSP instance with  $d(s_i, s_j) = 1$  if there is an edge from  $i$  to  $j$  and  $d(s_i, s_j) = 2$  if there is not. Ask, 'is there a tour of length  $\leq n$ '

**Subgraph isomorphism** Given two graphs  $G, G'$ . Does  $G$  contain a subgraph isomorphic to  $G'$ ? Interestingly, Graph isomorphism (i.e., 'Are graphs  $G$  and  $G'$  isomorphic?') is known to be in  $\mathcal{NP}$ , but it is suspected to be neither in  $\mathcal{P}$  or  $\mathcal{NP}$ -complete.

**Clique decision problem** Given a graph  $G$  and number  $k$ . Does  $G$  contain a clique of size  $k$ ? (i.e.,  $k$  vertices all pairs of which are connected together). E.g., below left:  $k = 3$ , *yes*;  $k = 4$ , *no*.



**Vertex cover decision problem** Given a graph  $G$  and number  $k$ . Is there a set of  $k$  vertices such that every edge of  $G$  starts or finishes at one of them? Such a set of vertices is called a vertex cover. E.g., above right:  $k = 2$ , *no*;  $k = 3$ , *yes*.

# 6 Computational Complexity of LP

## 6.1 Running time of the simplex algorithm

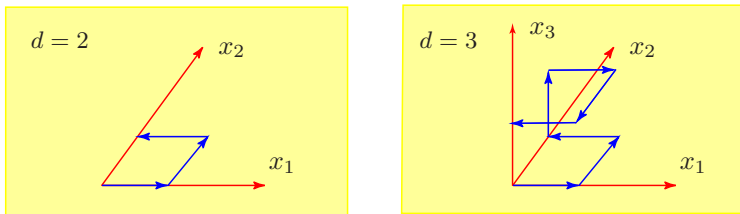
### Worst-case running time

The simplex algorithm moves from one basic feasible solution to an adjacent one, each time improving the value of the objective function. However, it can take an exponentially large number of steps to terminate.

Suppose the feasible region is the cube in  $\mathbb{R}^d$  defined by the constraints

$$0 \leq x_i \leq 1, \quad i = 1, \dots, d$$

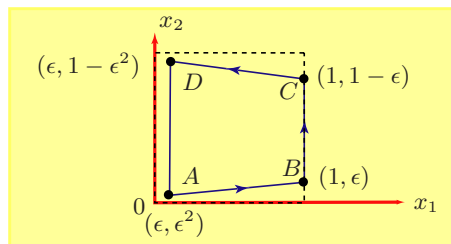
and we seek to maximize  $x_d$ . There are  $2^d$  vertices. The paths shown below visit all vertices before terminating at  $(0, 0, \dots, 1)$ .



Given  $0 < \epsilon < 1/2$ , consider the perturbed unit cube given by the constraints

$$\epsilon \leq x_1 \leq 1, \quad \epsilon x_{i-1} \leq x_i \leq 1 - \epsilon x_{i-1} \quad i = 2, \dots, d.$$

It can be verified that the cost function increases strictly with each move along the path. For example, for  $d = 2$  we have



Note that  $x_2$  increases along the route ABCD. So if our pivoting rule is always to move to the adjacent bfs for which the entering variable has the least index (so-called Bland's rule), then the simplex algorithm will require  $2^d - 1$  pivoting steps before terminating. With this pivoting rule the simplex algorithm has exponential

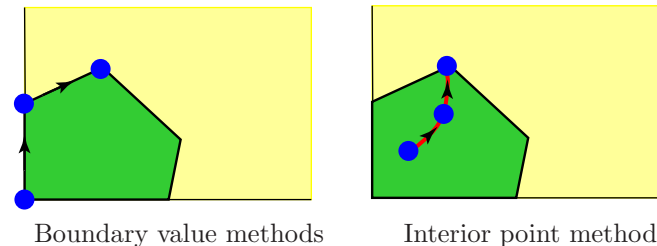
worst-case time complexity. Observe that the initial and final vertices are adjacent and a different pivoting rule could reach the optimum in only one step. However, for all common pivoting rule that have been studied there is some instance for which the number of steps until termination is exponential. It is unknown whether there is a pivoting rule that might make the simplex algorithm more efficient. This is related to the **Hirsch conjecture**: that for a polytope in  $\mathbb{R}^n$  defined by  $m$  inequality constraints the number of pivots required to move from one bfs to another bfs is never more than  $m - n$ .

### Average-case running time

Not all algorithms with exponential time complexity are bad in practice. Often, they can have very good performance. The simplex algorithm appears to perform well on average. The difficulty is defining what is meant by 'on average'.

## 6.2 The complexity of LP

We shall investigate alternative algorithms for solving LP. In particular, we seek a polynomial time algorithm. If we can show that  $LP \in P$  then this is likely to tell us something about good algorithms in practice. There are two classes of methods for solving LP.



### The size of an LP instance

Any non-negative integer,  $r$ , ( $r \leq U$ ) can be written in binary form

$$r = a_k 2^k + a_{k-1} 2^{k-1} + \dots + a_1 2^1 + a_0 2^0 \leq 2^{\log_2 U}$$

where  $a_0, a_1, \dots, a_k$  are 0 or 1. The number  $k$  is at most  $\lfloor \log_2 U \rfloor$ . Thus, using an extra bit for the sign, we can represent any integer  $r$  where  $|r| \leq U$  by at most  $(\lfloor \log_2 U \rfloor + 2)$  bits.

An instance of an LP problem is given by a  $m \times n$  matrix  $A$ , a  $m$ -vector  $b$  and a  $n$ -vector  $c$ . So, assuming that the largest magnitude of any of the components

is  $U$ , an LP instance has a size in bits of

$$(mn + m + n)(\lceil \log_2 U \rceil + 2).$$

### 6.3 Feasibility problem

Consider the primal/dual pair:

$$P : \text{minimize } \{c^\top x : Ax \geq b\}$$

$$D : \text{maximize } \{b^\top \lambda : A^\top \lambda = c, \lambda \geq 0\}.$$

By the strong duality of linear programming, each problem has an optimal solution if and only there is a feasible solution to

$$b^\top \lambda = c^\top x \quad Ax \geq b, \quad A^\top \lambda = c, \quad \lambda \geq 0.$$

Thus we can solve LP if we can solve a feasibility problem like this. We shall therefore focus on feasibility and the decision problem

*Is the polyhedron  $P = \{x \in \mathbb{R}^n : Ax \geq b\}$  non-empty?*

The algorithm that we shall use is known as the **ellipsoid method**.

### 6.4 Preliminaries for ellipsoid method

#### Definitions 6.1

1. Let  $D$  be a  $n \times n$  positive definite symmetric matrix. The set

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

is called an **ellipsoid** with centre at  $z \in \mathbb{R}^n$ .

2. Let  $D$  be a  $n \times n$  non-singular matrix and  $t \in \mathbb{R}^n$ . The mapping  $S : \mathbb{R}^n \mapsto \mathbb{R}^n$  defined by  $S(x) = Dx + t$  is called an **affine transformation**.

3. The **volume** of a set  $L \subset \mathbb{R}^n$ , denoted by  $\text{Vol}(L)$ , is defined by

$$\text{Vol}(L) = \int_{x \in L} dx.$$

We shall use the result that if  $S$  is given by the affine transformation  $S(x) = Dx + t$  then

$$\text{Vol}(S(L)) = |\det(D)|\text{Vol}(L).$$

### 6.5 Intuitive description of the ellipsoid method

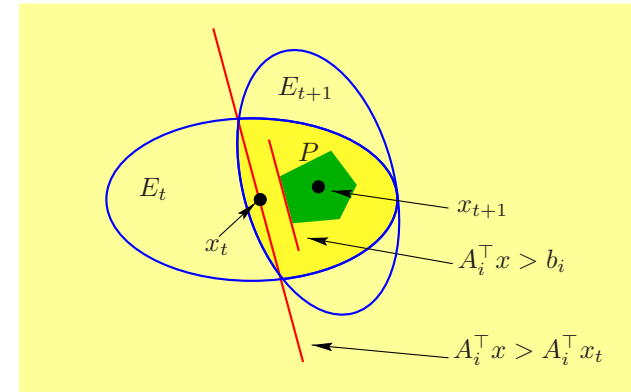
We generate a sequence of ellipsoids  $\{E_t\}$ .  $E_t$  has centre  $x_t$ , such that

- If  $x_t \in P$ , then  $P$  is non-empty and the method stops.
- If  $x_t \notin P$ , then there is a violated constraint with  $A_i x_t < b_i$ , where  $A_i$  is some row of  $A$  and  $b_i$  is the matching element of  $b$ .

Thus,  $P$  is contained in the half-space  $\{x \in \mathbb{R}^n : A_i x \geq A_i x_t\}$ . Call the intersection of this half-space with  $E_t$  a half-ellipsoid.

We construct the ellipsoid  $E_{t+1}$  in such a way that it covers this half-ellipsoid and has volume only a fraction of that of  $E_t$ .

We repeat this procedure until either we find a point of  $P$  or we conclude that the volume of  $P$  is very small and therefore can be taken as empty.



The key result is as follows.

**Theorem 6.1** Let  $E = E(z, D)$  be an ellipsoid in  $\mathbb{R}^n$ , and  $a$  be a non-zero  $n$ -vector. Consider the half-space  $H = \{x \in \mathbb{R}^n : a^\top x \geq a^\top z\}$  and let

$$\bar{z} = z + \frac{1}{n+1} \frac{Da}{\sqrt{a^\top Da}},$$

$$\bar{D} = \frac{n^2}{n^2 - 1} \left( D - \frac{2}{n+1} \frac{Daa^\top D}{a^\top Da} \right).$$

Then the matrix  $\bar{D}$  is symmetric and positive definite and thus  $E' = E(\bar{z}, \bar{D})$  is an ellipsoid. Moreover,

- (a)  $E \cap H \subset E'$ ,
- (b)  $\text{Vol}(E') < e^{-1/(2(n+1))} \text{Vol}(E)$ .

## 7 The Ellipsoid Method

### 7.1 Ellipsoid algorithm

Khachiyan's ellipsoid method (1979):

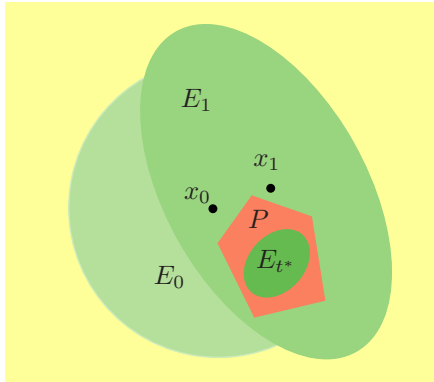
#### Input

- (a) A matrix  $A$  and vector  $b$  defining the polyhedron  $P = \{x \in \mathbb{R}^n : A_i^\top x \geq b_i, i = 1, \dots, m\}$ .
- (b) A number  $v$ , such that either  $P$  is empty or  $\text{Vol}(P) > v$ .
- (c) An ellipsoid (in fact, a ball)  $E_0 = E(x_0, r^2 I)$  with volume at most  $V$ , such that  $P \subset E_0$ .

**Output** A feasible point  $x^* \in P$  if  $P$  is non-empty, or a statement that  $P$  is empty.

We will show subsequently that  $v = n^{-n}(nU)^{-n^2(n+1)} > \text{Vol}(E_{t^*})$  and

$$V = (2n)^n (nU)^{n^2} > \text{Vol}(E_0) > \text{Vol}(E_1) > \dots > \text{Vol}(E_{t^*}).$$



**Initialize step** Let

$$t^* = \lceil 2(n+1) \log(V/v) \rceil, \quad E_0 = E(x_0, r^2 I), \quad D_0 = r^2 I, \quad t = 0.$$

**Main iteration**

- (a) If  $t = t^*$  stop;  $P$  is empty.
- (b) If  $x_t \in P$  stop;  $P$  is non-empty.

- (c) If  $x_t \notin P$  find a violated constraint,  $i$ , such that  $A_i^\top x_t < b_i$ .
- (d) Let  $H_t = \{x \in \mathbb{R}^n : A_i^\top x \geq A_i^\top x_t\}$ .

Construct an ellipsoid  $E_{t+1} = E(x_{t+1}, D_{t+1})$  containing  $E_t \cap H_t$  with

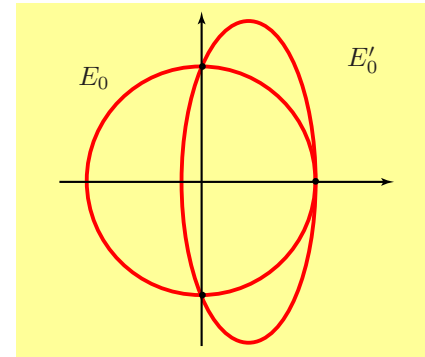
$$x_{t+1} = x_t + \frac{1}{n+1} \frac{D_t A_i}{\sqrt{A_i^\top D_t A_i}},$$

$$D_{t+1} = \frac{n^2}{n^2-1} \left( D_t - \frac{2}{n+1} \frac{D_t A_i A_i^\top D_t}{A_i^\top D_t A_i} \right)$$

- (e)  $t := t+1$ , return to (a).

### 7.2 Proof that $E \cap H \subset E'$

First, consider the case  $z = 0$ ,  $D = I$  and  $a = e_1 = (1, 0, \dots, 0)^\top$ . So,  $E_0 = \{x \in \mathbb{R}^n : x^\top x \leq 1\}$  and  $H_0 = \{x \in \mathbb{R}^n : x_1 \geq 0\}$ .



Hence,

$$E'_0 = E \left( \frac{e_1}{n+1}, \frac{n^2}{n^2-1} \left( I - \frac{2}{n+1} e_1 e_1^\top \right) \right).$$

Re-writing this, we have

$$E'_0 = \left\{ x \in \mathbb{R}^n : \left( \frac{n+1}{n} \right)^2 \left( x_1 - \frac{1}{n+1} \right)^2 + \frac{n^2-1}{n^2} \sum_{i=2}^n x_i^2 \leq 1 \right\}$$

$$= \left\{ x \in \mathbb{R}^n : \frac{n^2-1}{n^2} \sum_{i=1}^n x_i^2 + \frac{2(n+1)}{n^2} x_1^2 + \left( \frac{n+1}{n} \right)^2 \left( -\frac{2x_1}{n+1} + \frac{1}{(n+1)^2} \right) \leq 1 \right\}$$

So that

$$E'_0 = \left\{ x \in \mathbb{R}^n : \frac{n^2 - 1}{n^2} \sum_{i=1}^n x_i^2 + \frac{1}{n^2} + \frac{2(n+1)}{n^2} x_1(x_1 - 1) \leq 1 \right\}.$$

Now suppose  $x \in E_0 \cap H_0$ . Then  $0 \leq x_1 \leq 1$ , and so  $x_1(x_1 - 1) \leq 0$ . Also,  $\sum_{i=1}^n x_i^2 \leq 1$ . Hence

$$\frac{n^2 - 1}{n^2} \sum_{i=1}^n x_i^2 + \frac{1}{n^2} + \frac{2(n+1)}{n^2} x_1(x_1 - 1) \leq \frac{n^2 - 1}{n^2} + \frac{1}{n^2} = 1,$$

which verifies that  $x \in E'_0$ , proving that  $E_0 \cap H_0 \subset E'_0$ .

Now, consider the general case and construct an affine transformation  $T(\cdot)$  such that  $T(E) = E_0$ ,  $T(H) = H_0$  and  $T(E') = E'_0$ . The result then follows since affine transformations preserve set inclusion, i.e. if  $A \subset B \subset \mathbb{R}^n$  and  $T(\cdot)$  is an affine transformation, then  $T(A) \subset T(B)$ .

Given  $E = E(z, D)$ , introduce the affine transformation

$$T(x) = RD^{-1/2}(x - z)$$

where  $R$  is a rotation matrix which rotates the unit ball so that  $D^{1/2}a$  is aligned with the unit vector  $e_1 = (1, 0, \dots, 0)^\top$  i.e.

$$R^\top R = I \quad \text{and} \quad RD^{1/2}a = \|D^{1/2}\|e_1$$

So now,  $T(E) = E_0$  since

$$\begin{aligned} x \in E &\iff (x - z)^\top D^{-1}(x - z) \leq 1 \\ &\iff (x - z)D^{-1/2}R^\top RD^{-1/2}(x - z) \leq 1 \\ &\iff RD^{-1/2}(x - z) \in E_0 \\ &\iff T(x) \in E_0. \end{aligned}$$

Similarly,  $T(H) = H_0$  since

$$\begin{aligned} x \in H &\iff a^\top(x - z) \geq 0 \\ &\iff a^\top D^{1/2}R^\top RD^{-1/2}(x - z) \geq 0 \\ &\iff e_1^\top T(x) \geq 0 \\ &\iff T(x) \in H_0 \end{aligned}$$

Similarly, one can show  $T(E') = E'_0$ . Above, we proved that  $E_0 \cap H_0 \subset E'_0$ , which is equivalent to  $T(E) \cap T(H) \subset T(E')$ , which implies  $E \cap H \subset E'$ .  $\blacksquare$

### 7.3 Proof that $\text{Vol}(E') < e^{-1/(2(n+1))}\text{Vol}(E)$

We have that

$$\frac{\text{Vol}(E')}{\text{Vol}(E)} = \frac{\text{Vol}(T(E'))}{\text{Vol}(T(E))} = \frac{\text{Vol}(E'_0)}{\text{Vol}(E_0)}.$$

Now,

$$E'_0 = E \left( \frac{e_1}{n+1}, \frac{n^2}{n^2-1} \left( I - \frac{2}{n+1} e_1 e_1^\top \right) \right).$$

So, introduce the affine transformation

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

One can easily check that  $E'_0 = F(E_0)$ . So

$$\text{Vol}(E'_0) = \sqrt{\det \left( \frac{n^2}{n^2-1} \left( I - \frac{2}{n+1} e_1 e_1^\top \right) \right)} \text{Vol}(E_0)$$

Hence,

$$\begin{aligned} \frac{\text{Vol}(E'_0)}{\text{Vol}(E_0)} &= \left( \frac{n^2}{n^2-1} \right)^{n/2} \left( 1 - \frac{2}{n+1} \right)^{1/2} \\ &= \frac{n}{n+1} \left( \frac{n^2}{n^2-1} \right)^{(n-1)/2} \\ &= \left( 1 - \frac{1}{n+1} \right) \left( 1 + \frac{1}{n^2-1} \right)^{(n-1)/2} \\ &< e^{-1/(n+1)} \left( e^{1/(n^2-1)} \right)^{(n-1)/2} \\ &= e^{-1/(2(n+1))}, \end{aligned}$$

using (twice) the inequality  $1 + a < e^a$  ( $a \neq 0$ ). Therefore,

$$\frac{\text{Vol}(E')}{\text{Vol}(E)} < e^{-1/(2(n+1))}.$$

$\blacksquare$

## 8 Complexity of the Ellipsoid Algorithm

### 8.1 The bound $V$

**Lemma 8.1** *Let  $A$  be a  $m \times n$  integer matrix and let  $b$  be a vector in  $\mathbb{R}^m$ . Let  $U$  be the largest absolute value of the entries of  $A$  and  $b$ . Then,*

(a) *Every extreme point of the polyhedron  $P = \{x \in \mathbb{R}^n : Ax \geq b\}$  satisfies*

$$-(nU)^n \leq x_j \leq (nU)^n, \quad j = 1, \dots, n.$$

(b) *Every extreme point of the (standardized) polyhedron  $P = \{x \in \mathbb{R}^n : Ax = b\}$  satisfies*

$$-(mU)^m \leq x_j \leq (mU)^m, \quad j = 1, \dots, n.$$

**Proof.** Consider first (a). We are assuming here that  $m > n$ . Let  $x$  be an extreme point of  $P$ . Choose  $n$  linearly independent active constraints and write  $\hat{A}x = \hat{b}$  where  $\hat{A}$  is  $n \times n$  invertible submatrix of  $A$  and  $\hat{b}$  is the matching  $n$ -dimensional subvector of  $b$ . So, we have  $x = \hat{A}^{-1}\hat{b}$ .

By Cramer's rule, we can write the solution

$$x_j = \frac{\det(\hat{A}^j)}{\det(\hat{A})},$$

where  $\hat{A}^j$  is the same as  $\hat{A}$  except that the  $j$ th column is replaced by  $\hat{b}$ . Now

$$\left| \det(\hat{A}^j) \right| = \left| \sum_{\sigma} (-1)^{|\sigma|} \prod_{i=1}^n \hat{a}_{i,\sigma(i)} \right| \leq n!U^n \leq (nU)^n, \quad j = 1, \dots, n,$$

where  $\sigma$  is one of the  $n!$  permutations of  $1, \dots, n$ , with  $|\sigma|$  giving the number of inversions (i.e.,  $i < j$  and  $\sigma(i) > \sigma(j)$ ).

Finally, since  $\hat{A}$  is invertible,  $\det(\hat{A}) \neq 0$  and all entries in  $A$  are integer so  $|\det(\hat{A})| \geq 1$ . Therefore, the extreme point  $x$  satisfies

$$|x_j| \leq (nU)^n, \quad \text{for all } j.$$

Exactly, the same argument may be used for (b) except that we use a basis matrix  $A_B$ .  $A_B$  is  $m \times m$  and we can replace  $n$  by  $m$  throughout. ■

By part (a) of the previous lemma all the extreme points of  $P = \{x \in \mathbb{R}^n : Ax \geq b\}$  are contained in the bounded polyhedron  $P_B$  defined by

$$P_B = \{x \in P : |x_j| \leq (nU)^n, j = 1, \dots, n\}.$$

$P$  is nonempty if and only if it contains an extreme point. Hence, we can test for the emptiness of  $P_B$  instead of  $P$ . But  $P_B$  is contained in the ball  $E(0, n(nU)^{2n}I)$  whose volume is at most

$$V = (2n(nU)^n)^n = (2n)^n (nU)^{n^2}.$$

### 8.2 The bound $v$

We say a polyhedron  $P$  is **full-dimensional** if it has positive volume. For example,  $P = \{(x_1, x_2) : x_1 + x_2 = 1, x_1, x_2 \geq 0\}$  has  $\text{Vol}(P) = 0$  and so is not full-dimensional.

**Lemma 8.2** *Let  $P = \{x \in \mathbb{R}^n : Ax \geq b\}$  and assume that  $A$  and  $b$  have integer entries which are bounded in absolute value by  $U$ . Let*

$$\epsilon = \frac{1}{2(n+1)} [(n+1)U]^{-(n+1)}, \quad P_\epsilon = \{x \in \mathbb{R}^n : Ax \geq b - \epsilon e\}$$

where  $e^\top = (1, 1, \dots, 1)$ . Then

(a) *If  $P$  is empty, then  $P_\epsilon$  is empty.*

(b) *If  $P$  is non-empty, then  $P_\epsilon$  is full-dimensional.*

**Proof of (a)** Suppose  $P$  is empty and consider the infeasible linear program minimize  $\{0^\top x : Ax \geq b\}$  and its dual maximize  $\{\lambda^\top b : \lambda^\top A = 0^\top, \lambda \geq 0\}$ . Since the primal is infeasible the dual problem has value  $+\infty$ . Therefore, there exists a  $\lambda \geq 0$  with

$$\lambda^\top A = 0^\top \quad \lambda^\top b = 1.$$

So, using the previous lemma, we can find a bfs  $\hat{\lambda}$  to the constraints  $\lambda^\top A = 0^\top, \lambda^\top b = 1, \lambda \geq 0$  such that

$$\hat{\lambda}_i \leq ((n+1)U)^{n+1}, \quad \text{for all } i.$$

Since  $\hat{\lambda}$  is a bfs, at most  $n+1$  of its components are non-zero so that

$$\sum_{i=1}^m \hat{\lambda}_i \leq (n+1) ((n+1)U)^{n+1}.$$

Therefore,

$$\hat{\lambda}^\top (b - \epsilon e) = 1 - \epsilon \sum_{i=1}^m \hat{\lambda}_i \geq \frac{1}{2} > 0.$$

Hence, when we replace  $b$  by  $b - \epsilon e$  the value of the dual remains  $+\infty$  and the primal problem is again infeasible and  $P_\epsilon$  is also empty.



**Proof of (b)** Let  $x$  be an element of  $P$  so that  $Ax \geq b$ . Let  $y$  be a vector such that

$$x_j - \frac{\epsilon}{nU} \leq y_j \leq x_j + \frac{\epsilon}{nU}, \quad \text{for all } j.$$

Then the  $i$ th row of  $Ay$  satisfies

$$\begin{aligned} \sum_{j=1}^n a_{ij}y_j &\geq \sum_{j=1}^n a_{ij}x_j - \frac{\epsilon}{nU} \sum_{j=1}^n |a_{ij}| \\ &\geq b_i - \frac{\epsilon}{nU}nU \\ &= b_i - \epsilon. \end{aligned}$$

Therefore, any such vector  $y$  belongs to  $P_\epsilon$  and the set of all such vectors  $y$  (a cube) has positive volume (of  $(2\epsilon/nU)^n$ ) and so is full-dimensional. ■

The following lemma can also be proved.

**Lemma 8.3** Let  $P = \{x \in \mathbb{R}^n : Ax \geq b\}$  be a full-dimensional bounded polyhedron, where the entries of  $A$  and  $b$  are integer and have absolute value bounded by  $U$ . Then,

$$\text{Vol}(P) > n^{-n}(nU)^{-n^2(n+1)}.$$

### 8.3 Running time of the ellipsoid method

We have the values

$$V = (2n)^n(nU)^{n^2} \quad \text{and} \quad v = n^{-n}(nU)^{-n^2(n+1)}$$

and know that the ellipsoid method takes at most  $t^* = \lceil 2(n+1) \log(V/v) \rceil$  steps. This gives  $t^* = O(n^4 \log(nU))$ .

In practice, we apply the ellipsoid algorithm to  $P_\epsilon$ . We know that  $P$  is nonempty if and only if  $P_\epsilon$  is nonempty. Let  $P_\epsilon = \{x : (1/\epsilon)Ax \geq (1/\epsilon)b - e\}$ . Recall  $U$  is an integer, so  $1/\epsilon$  is also an integer and so this writes the constraints of  $P_\epsilon$  so that all coefficients are integers that are bounded in magnitude by  $U_\epsilon = U/\epsilon$ . We know that  $P_\epsilon$  is contained in a ball of volume at most  $V_\epsilon = (2n)^n(nU_\epsilon)^{n^2}$ . Also, when  $P$  is nonempty  $P_\epsilon$  has volume of at least  $v_\epsilon = (2\epsilon/nU)^n$  (by the proof in part (b) of Lemma 8.2). Thus the ellipsoid algorithm applied to  $P_\epsilon$  will terminate in a number of steps at most  $\lceil 2(n+1) \log(V_\epsilon/v_\epsilon) \rceil = O(n^4 \log(nU))$ .

There are a few wrinkles to consider: on a computer we cannot actually calculate the square root  $\sqrt{A_i^\top D_t A_i}$  that is needed to find  $x_{t+1}$  from  $x_t$ . There is also a worry that in multiplying numbers together we might be forced to use ones as large as  $2^U$ . However, it can be shown that if we carry out the algorithm to

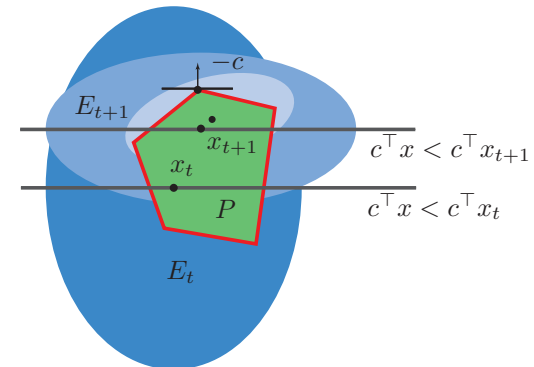
a certain accuracy it will still correctly decide whether  $P$  is nonempty. At each step the calculation of  $A_i^\top D_t A_i$  requires  $(n^2)$  operations, which makes the running time of the algorithm  $O(n^6 \log(nU))$ .

### 8.4 Sliding objective ellipsoid method

Suppose we wish to solve the problem minimize  $\{c^\top x : Ax \geq b, x \geq 0\}$ . First, use the ellipsoid method to find a feasible solution  $x_0 \in P$ , where  $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ . Now we apply the ellipsoid method again (note the strict inequality) to the new polyhedron given by

$$P \cap \{x \in \mathbb{R}^n : c^\top x < c^\top x_0\}.$$

If this is empty then  $x_0$  is optimal. Otherwise, we have a new solution  $x_1 \in P$ , say, with strictly smaller objective function than  $c^\top x_0$ . Now we reapply the ellipsoid method to the new polyhedron.



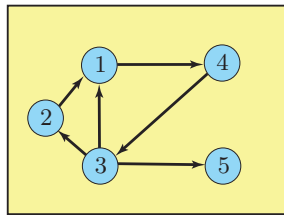
At each iteration we add a new constraint in the direction of the vector  $c$ . All the constraints  $c^\top x < c^\top x_t$  are parallel to one another. One can show that by this procedure we reach the optimum in polynomial running time.

## 9 The Network Simplex Algorithm

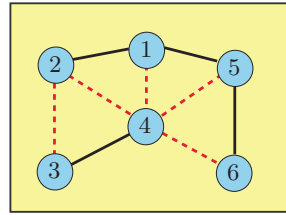
### 9.1 Graph terminology

The next four lectures are about network flow problems. They include transportation, assignment, maximum flow and shortest path problems.

A **graph**  $G = (N, A)$  consists of a set of **nodes**,  $N$ , and a set of **arcs**,  $A$ . In an **undirected graph** the arcs are unordered pairs of nodes  $\{i, j\} \in A, i, j \in N$ . In a **directed graph** (also called a **network**) the arcs are ordered pairs of nodes  $(i, j)$ . A **walk** is an ordered list of nodes  $i_1, i_2, \dots, i_t$  such that, in an undirected graph,  $\{i_k, i_{k+1}\} \in A$ , or, in a directed graph, that either  $(i_k, i_{k+1}) \in A$  or  $(i_{k+1}, i_k) \in A$ , for  $k = 1, \dots, t-1$ . A walk is a **path** if  $i_1, i_2, \dots, i_k$  are distinct, and a **cycle** if  $i_1, i_2, \dots, i_{k-1}$  are distinct and  $i_1 = i_k$ . A graph is **connected** if there is a path connecting every pair of nodes.



a directed graph



a spanning tree (dotted)

A network is **acyclic** if it contains no cycles. A network is a **tree** if it is connected and acyclic. A network  $(N', A')$  is a **subnetwork** of  $(N, A)$  if  $N' \subset N$  and  $A' \subset A$ . A subnetwork  $(N', A')$  is a **spanning tree** if it is a tree and  $N' = N$ .

### 9.2 The minimum cost flow problem

Let  $f_{ij}$  denote the amount of flow of some material on arc  $(i, j) \in A$ . Let  $b_i, i \in N$ , denote the amount of flow that enters the network at node  $i \in N$ . If  $b_i > 0$  we say the node is a **source** (supplying  $b_i$  units of flow). If  $b_i < 0$  we say that the node is a **sink** (with a demand of  $|b_i|$  units of flow).

Suppose there is a **cost** of  $c_{ij}$  per unit flow on arc  $(i, j) \in A$ . The **minimum cost flow problem** is

$$\text{minimize } \sum_{(i,j) \in A} c_{ij} f_{ij}$$

subject to

$$\begin{aligned} b_i + \sum_{j:(j,i) \in A} f_{ji} &= \sum_{j:(i,j) \in A} f_{ij}, \quad \text{for all } i \in N \\ m_{ij} &\leq f_{ij} \leq M_{ij}, \quad \text{for all } (i, j) \in A. \end{aligned}$$

These say that flows must be feasible and conserve flow at each node. For feasible flows to exist we must also have  $\sum_{i \in N} b_i = 0$ . An important special case is that of **uncapacitated flows**,  $m_{ij} = 0$  and  $M_{ij} = \infty$ .

Note that the **minimum cost flow** problem is a special form of linear program. Its simple structure allows for special algorithms. Constraints are of the form  $Ax = b$ , where

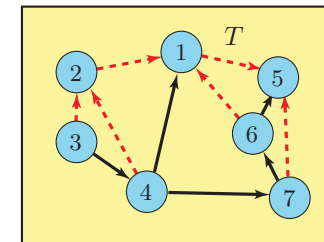
$$(A)_{ik} = \begin{cases} +1 & \text{node } i \text{ is start of } k\text{th arc;} \\ -1 & \text{node } i \text{ is end of } k\text{th arc;} \\ 0 & \text{otherwise.} \end{cases}$$

### 9.3 Spanning tree solutions

Assume that the network is connected. A **spanning tree solution**,  $f_{ij}$ , is one that can be constructed as follows

1. Pick a set  $T \subset A$  of  $n - 1$  arcs forming a spanning tree and partition the remaining arcs  $A \setminus T$  into the two sets  $L$  and  $U$ .
2. Set  $f_{ij} = m_{ij}$  for each arc  $(i, j) \in L$  and  $f_{ij} = M_{ij}$  for each arc  $(i, j) \in U$ .
3. Use the flow conservation constraints to determine the flows  $f_{ij}$  for arcs  $(i, j) \in T$ . We begin by determining the flows on arcs incident to leaves of the tree  $T$ . Subsequently we determine the flows on other arcs of  $T$ .

A spanning tree solution with  $m_{ij} \leq f_{ij} \leq M_{ij}$  is a **feasible spanning tree solution**.



**Theorem 9.1** A flow vector is a **spanning tree solution** if and only if it is a **basic solution**.

## 9.4 Optimality conditions

Consider the **Lagrangian** of the minimum cost flow problem

$$\begin{aligned} L(f; \lambda) &= \sum_{(i,j) \in A} c_{ij} f_{ij} - \sum_{i \in N} \lambda_i \left( \sum_{j: (i,j) \in A} f_{ij} - \sum_{j: (j,i) \in A} f_{ji} - b_i \right) \\ &= \sum_{(i,j) \in A} (c_{ij} - \lambda_i + \lambda_j) f_{ij} + \sum_{i \in N} \lambda_i b_i. \end{aligned}$$

Minimizing  $L(f; \lambda)$  over  $m_{ij} \leq f_{ij} \leq M_{ij}$  gives dual feasibility and complementary slackness conditions:

$$\begin{aligned} \bar{c}_{ij} = c_{ij} - \lambda_i + \lambda_j > 0 &\implies f_{ij} = m_{ij} \\ \bar{c}_{ij} = c_{ij} - \lambda_i + \lambda_j < 0 &\implies f_{ij} = M_{ij} \\ \bar{c}_{ij} = c_{ij} - \lambda_i + \lambda_j = 0 &\iff m_{ij} < f_{ij} < M_{ij} \end{aligned}$$

Observe that if  $T$  is a spanning tree then we can solve the following equations in a unique way, where  $n = |N|$ .

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

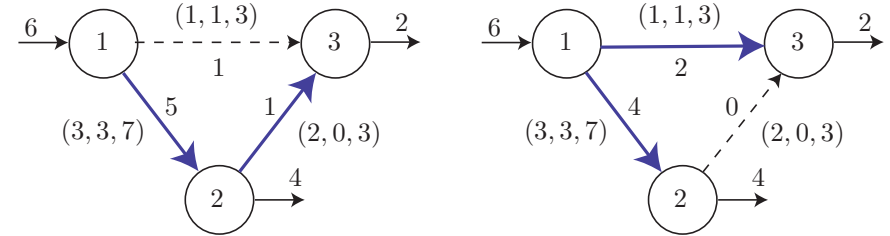
## 9.5 Pivoting to change the basis

We compute the **reduced costs**  $\bar{c}_{ij} = c_{ij} - (\lambda_i - \lambda_j)$  for each arc  $(i, j) \notin T$ . Recall  $\bar{c}_{ij} = 0$  for all arcs  $(i, j) \in T$  by construction.

If  $\bar{c}_{ij} \geq 0$  for all  $(i, j) \in L$  and  $\bar{c}_{ij} \leq 0$  for all  $(i, j) \in U$  then the current basic feasible solution is optimal. Otherwise, choose an arc  $(i, j)$  where there is a violation. This arc together with the tree  $T$  forms a cycle. Add (or subtract) as much flow as possible around this cycle so as to increase (or decrease)  $f_{ij}$ . Note that  $\sum_{k\ell} \bar{c}_{k\ell} = \sum_{k\ell} c_{k\ell} = \bar{c}_{ij}$ , where the sums are taken around the arcs of the cycle. Thus if  $\bar{c}_{ij}$  is negative we can decrease the total cost by increasing the flow  $f_{ij}$ . Similarly, if  $\bar{c}_{ij}$  is positive we can decrease cost by decreasing the  $f_{ij}$ .

**Example** Consider the minimum cost flow problem below. On each arc we give the values of  $(c_{ij}, m_{ij}, M_{ij})$ . There is  $b_1 = 6$ ,  $b_2 = -4$ , and  $b_3 = -2$ . The spanning tree consists of 2 arcs (shown undashed). In the left hand figure, we set  $\lambda_1 = 0$  and find  $\lambda_2 = -3$  (so  $c_{12} = 3 = \lambda_1 - \lambda_2$ ). Similarly,  $\lambda_3 = -5$ . On the arc  $(1, 3)$  the value of  $c_{13} - \lambda_1 + \lambda_3 = 1 - (0) + (-5) = -4$ . Since this is  $< 0$  we can decrease cost by increasing  $f_{13}$ . Inserting the arc  $(1, 3)$  into the tree produces the cycle  $(1, 3, 2, 1)$ . We increase the flow  $f_{13}$  as much as possible shifting flow around this

cycle (i.e., by 1). This produces the flows shown in the right diagram. The tree is now arcs  $(1, 3), (1, 2)$ . We recalculate:  $\lambda_1 = 0$ ,  $\lambda_2 = -3$  and  $\lambda_3 = -1$ . The value of  $c_{23} - \lambda_2 + \lambda_3 = 2 - (-3) + (-1) = 4$ . Since this is  $> 0$  we want flow on  $(2, 3)$  to be minimal, which it is. So we now have the optimal solution.



## 9.6 Finding the initial feasible tree solution

1. Every network flow problem can be reduced to one with exactly one source node and one sink node (by adding in two nodes).
2. Every network flow problem can be reduced to one without sources or sinks (by connecting the above two nodes with an edge). The constraints are just  $Af = 0$ . Any  $f$  satisfying this is called a **circulation** and such flow problems are called circulation problems.
3. In the case that  $m_{ij} = 0$ , for all  $i, j$ , the zero flow is a feasible tree solution. If  $m_{ij} \neq 0$  for some arc  $(i, j)$  we can replace the flows by  $f_{ij} - m_{ij}$  and adjust the supplies  $b_i$  accordingly.

## 9.7 Integrality of optimal solutions

Suppose the input data  $(m_{ij}, M_{ij}$  and  $b_i)$  are all integers. Then the above algorithm leads to optimal integer solutions. There are no multiplications or divisions.

**Theorem 9.2 (Integrality theorem)** *For every network flow problem with integer data, every basic feasible solution and, in particular, every basic optimal solution assigns integer flow to every arc.*

This theorem is important for the many practical problems in which an integer solution is required for a meaningful interpretation (for example, the assignment problems). Later, we investigate linear programming problems subject to the additional constraint that the solution be in integers. Such problems are usually much harder to solve than the problem without the integer constraint. However, for network flow problems we get integer solutions for free.

# 10 Transportation and Assignment Problems

## 10.1 Transportation problem

In the transportation problem there are  $m$  suppliers of a good and  $n$  customers. Suppose supplier  $i$  produces  $s_i$  units of the good, customer  $j$  demands  $d_j$  units of the good, and there is a balance between demand and supply so that

$$\sum_{i=1}^m s_i = \sum_{j=1}^n d_j.$$

Suppose the cost of transporting a unit of good from supplier  $i$  to consumer  $j$  is  $c_{ij}$ . The problem is to match suppliers with consumers to minimize the total transportation cost. We can easily formulate the transportation problem as a minimum cost flow problem as follows

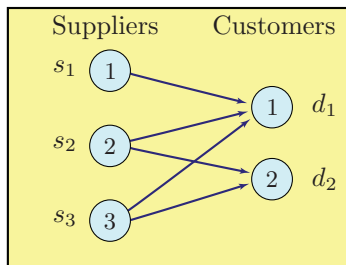
$$\text{minimize } \sum_{i=1}^m \sum_{j=1}^n c_{ij} f_{ij}$$

subject to

$$\sum_{i=1}^m f_{ij} = d_j, \quad j = 1, \dots, n, \quad \sum_{j=1}^n f_{ij} = s_i, \quad i = 1, \dots, m,$$

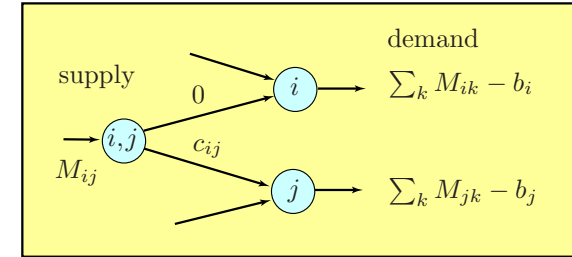
$$f_{ij} \geq 0, \quad \text{for all } i, j.$$

This is a special case of the minimum cost flow problem with  $m_{ij} = 0, M_{ij} = \infty$  and the graph structure of a **bipartite graph**. That is, the nodes divide into disjoint sets  $S$  (suppliers) and  $C$  (customers) and  $A \subset S \times C$  (the only arcs are those which connect suppliers to consumers).



**Lemma 10.1** Every minimum cost flow problem is equivalent to a transportation problem.

**Proof.** Consider the minimum cost flow problem with  $m_{ij} = 0, M_{ij} < \infty$ , and input data  $G = (N, A), M_{ij}, c_{ij}$  and  $b_i$ . For every arc  $(i, j) \in A$  construct a source node with supply  $M_{ij}$ . For every node  $i \in N$  construct a sink node with demand  $\sum_{k: (i,k) \in A} M_{ik} - b_i$ . Now connect every source node  $(i, j)$  to each of the sink nodes  $i$  and  $j$  with infinite upper bounds on capacities. Let  $c_{ij,i} = 0$  and  $c_{ij,j} = c_{ij}$ .



There is a 1-1 correspondence between feasible flows in the two problems and these flows have the same costs. To see this put a flow of  $f_{ij}$  on the arc from  $i, j$  to  $j$ , and a flow of  $M_{ij} - f_{ij}$  on the arc from  $i, j$  to  $i$ . The total amount flowing into node  $i$  is then  $\sum_j (M_{ij} - f_{ij}) + \sum_j f_{ji}$ , which must equal  $\sum_j M_{ij} - b_i$ . Thus we have the flow conservation constraints of the minimum cost flow problem. ■

For this reason new algorithms are often first tested on transportation problems. The case in which there is an arc from every supplier to every consumer is known as the **Hitchcock transportation problem**.

## 10.2 Tableau form

It is convenient to present the input data and spanning tree solutions (i.e., the bfs's) for the transportation problem in tableau form. (This is a different form of tableau to that of the simplex tableau). We express the input data in a tableau

	$\lambda_j s$	-5	-1	7				
$\lambda_i s$	0	7	5	1	6	-7	1	7
	3	3	8	8	4	-4	3	11
	8	13	1	14	- $\theta$	9	4	18
	13	18	1	14	3	12	- $\theta$	12
		10	22	16				

$\theta$  suppliers  $s_i$

demands  $d_j$

$\longrightarrow$

		-5	-1	7		
0	7	5	1	6	-7	1
3	3	8	8	4	-4	3
8	13	1	2	9	16	1
2	7	1	12	3	-5	6

The  $\lambda_i$  are computed using the fact that we require  $\lambda_i - \lambda_j = c_{ij}$  wherever  $f_{ij} > 0$ . At the first iteration we increase (by as much as possible) the flow in an empty cell where  $\lambda_i - \lambda_j > c_{ij}$  (i.e.,  $\bar{c}_{ij} > 0$ ). We do this by adding and subtracting  $\theta$  around some cycle of cells in which  $f_{ij} > 0$ .

further iterations

		-1	-2	-1			
0	1	5	2	6	7	1	
2	3	8	10 + $\theta$	4	1	1 - $\theta$	3
0	10 - $\theta$	1	2	9	8 + $\theta$	1	
1	2 + $\theta$	1	12 - $\theta$	3	2	6	

		-1	-3	-1		
0	1	5	3	6	7	1
1	2	8	11	4	2	3
0	9	1	3	9	9	1
0	1	1	11	3	1	6

The final tableau above contains the optimal solution because we have  $\lambda_i - \lambda_j = c_{ij}$  everywhere that  $f_{ij} > 0$  and  $\lambda_i - \lambda_j \leq c_{ij}$  everywhere else.

### 10.3 Assignment problem

Given a set  $P$  of  $m$  people and a set  $T$  of  $m$  tasks and a cost,  $c_{ij}$ , the **assignment problem** is one of choosing variables  $f_{ij}$  to

$$\text{minimize } \sum_{i=1}^m \sum_{j=1}^m c_{ij} f_{ij},$$

subject to

$$f_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned to task } j \\ 0 & \text{otherwise.} \end{cases}$$

$$\sum_{j=1}^m f_{ij} = 1, \quad \text{for all } i = 1, \dots, m$$

$$\sum_{i=1}^m f_{ij} = 1, \quad \text{for all } j = 1, \dots, m.$$

These constraints say that each person is assigned to exactly one task and that every task is covered. Except for the integer constraints, the assignment problem is a special case of the Hitchcock transportation problem

### 10.4 Integer constraints

The problem in which the integer constraints are replaced with  $0 \leq f_{ij} \leq 1$  is known as the **LP-relaxation** of the assignment problem. If we use the spanning tree method then our solution will take values 0 or 1 and hence be optimal for both the LP-relaxation and the assignment problem.

Had we used a non-simplex type method to solve the underlying linear program (e.g., some interior point projective algorithm) then an integer-valued optimal solution may not be guaranteed. It is a feature of the method and not the problem. Many LP-relaxations of problems have multiple non-integer solutions.

### 10.5 Maximum flow problem

Suppose we have a network with a single source node, 1 and a single sink node  $n$  and upper bounds  $M_{ij}$  on all the arcs. Also, assume for convenience that  $m_{ij} = 0$ . The maximum flow problem is then to send as much flow from 1 to  $n$ . We write this as

$$\text{maximize } \delta$$

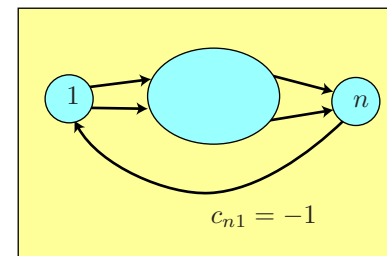
subject to

$$\sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} = \begin{cases} \delta & i = 1 \\ 0 & i \neq 1, n \\ -\delta & i = n \end{cases}$$

$$0 \leq f_{ij} \leq C_{ij}, \quad \text{for all } (i, j) \in A.$$

We can formulate the maximum flow problem as a minimum cost flow problem by adding an additional arc  $(n, 1)$  to the network with  $m_{n1} = 0$  and  $M_{n1} = \infty$  and then assign cost  $c_{n1} = -1$  to the arc  $(n, 1)$  and zero cost to all the original arcs.

Since, the only arc with non-zero cost has negative cost it follows that the optimal solution to this minimum cost flow problem will circulate as much flow as possible across the network, constrained only by the original arc capacities — i.e., it also solves the maximum flow problem.



# 11 Maximum Flow and Shortest Path Problems

## 11.1 Max-flow min-cut theorem

We return to the max-flow problem of Section 10.5. For  $S \subset N$  define the capacity of the cut  $[S, N \setminus S]$  as

$$C(S, N \setminus S) = \sum_{i \in S, j \notin S} C_{ij}.$$

**Theorem 11.1 (Max-flow min-cut theorem)**

$$\text{Max-flow, } \delta = \text{min cut capacity} = \min_{S: 1 \in S, n \notin S} C(S, N \setminus S)$$

There are two parts to the proof. First

**value of any flow  $\leq$  capacity of any cut**

Define

$$f(X, Y) = \sum_{i \in X, j \in Y: (i, j) \in A} f_{ij}$$

and suppose that  $1 \in S, n \notin S$ . Then

$$\begin{aligned} \delta &= \sum_{i \in S} \left( \sum_{j: (i, j) \in A} f_{ij} - \sum_{j: (j, i) \in A} f_{ji} \right) \\ &= f(S, N) - f(N, S) \\ &= f(S, S) + f(S, N \setminus S) - f(N \setminus S, S) - f(S, S) \\ &= f(S, N \setminus S) - f(N \setminus S, S) \\ &\leq f(S, N \setminus S) \\ &\leq C(S, N \setminus S). \end{aligned}$$

We now complete the proof using the **Ford-Fulkerson algorithm**. Suppose that  $f_{ij}$  is optimal and recursively define  $S \subset N$  as follows

1.  $1 \in S$
2. If  $i \in S$  and  $f_{ij} < C_{ij}$  then  $j \in S$
3. If  $i \in S$  and  $f_{ji} > 0$  then  $j \in S$ .

So,  $S$  is the set of nodes to which you can increase flow. Either  $n \in S$  in which case we can increase flow along a path from 1 to  $n$ , or  $n \notin S$  so that  $[S, N \setminus S]$  is a cut with  $1 \in S$  and  $n \notin S$ . But for  $i \in S, j \notin S, f_{ij} = C_{ij}, f_{ji} = 0$  and

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

We can take zero flow  $f_{ij} = 0$  as the initial flow. If all capacities and initial flows are integer then every step increases the flow by at least one unit. Thus the algorithm will converge in a finite number of steps.

## Dual formulation

We can recast the max-flow problem as a minimum cost flow problem:

$$\begin{aligned} &\text{minimize } -f_{n1} \\ &\text{subject to } \sum_{j: (i, j) \in A} f_{ij} - \sum_{j: (j, i) \in A} f_{ji} = 0, \quad \text{for all } i \in N \\ &0 \leq f_{ij} \leq C_{ij}, \quad \text{for all } (i, j) \in A, \quad f_{n1} \geq 0. \end{aligned}$$

Consider the Lagrangian in the usual way with dual variables,  $\lambda_i, i \in N$ . For optimality on arc  $(n, 1)$  we have  $(c_{n1} = -1)$

$$\bar{c}_{n1} = c_{n1} - \lambda_n + \lambda_1 = 0,$$

so that  $\lambda_1 = 1 + \lambda_n$ . On all the other arcs the costs are zero so that the reduced costs are just  $\bar{c}_{ij} = \lambda_j - \lambda_i$  and at an optimal solution

$$\begin{aligned} \lambda_j - \lambda_i > 0 &\implies f_{ij} = 0 \\ \lambda_j - \lambda_i < 0 &\implies f_{ij} = C_{ij}. \end{aligned}$$

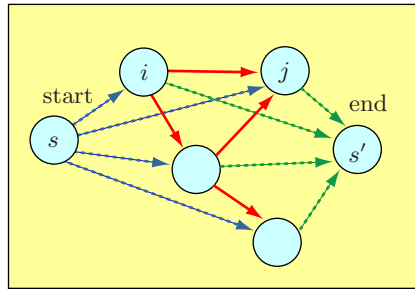
So  $\lambda_i = 1$  for  $i \in S$  and  $\lambda_j = 0, j \in N \setminus S$ .

## 11.2 Project management

A project that is described by a set of jobs that must be completed in a certain order. Job  $i$  has a duration  $\tau_i$ . How can we determine the least time in which the project can be completed?

Consider a graph in which there is an arc  $(i, j)$  whenever job  $i$  must be completed before job  $j$ . Introduce two additional jobs,  $s$  and  $s'$ , each of zero duration, to indicate the start and finish of the project, and introduce arcs  $(s, i)$  and  $(i, s')$  for every job  $i$ . Suppose we start job  $i$  at time  $t_i$ . We wish to

$$\text{minimize } (t_{s'} - t_s), \quad \text{subject to } t_j - t_i \geq \tau_i, \quad \text{for all } (i, j) \in A.$$



The dual of this problem is

$$\text{maximize } \sum_{(i,j) \in A} \tau_i f_{ij}$$

subject to

$$\sum_{j: (j,i) \in A} f_{ji} - \sum_{j: (i,j) \in A} f_{ij} = -b_i, \quad \text{for all } i, \text{ and } f_{ij} \geq 0, \text{ for all } (i,j) \in A,$$

where  $b_s = 1$ ,  $b_{s'} = -1$  and  $b_i = 0$  for  $i \neq s, s'$ . This is a minimum cost flow problem with each arc cost being  $-\tau_i$ . The path of arcs for which  $f_{ij} = 1$  defines the **critical path**.

### 11.3 The shortest path problem

Shortest path problems have applications in transportation and communications, and are often subproblems embedded in more complex problems. Although they are special forms of minimum cost flow problems they can be solved more efficiently by specialized algorithms. Given a network  $(N, A)$  we think of each arc having a length  $c_{ij} \geq 0$  and consider paths in which arcs are traversed in the forward direction only. The length of a path is the sum of the lengths of the associated arcs. A shortest path between a given pair of nodes is the path between them of minimum length. It is convenient to consider the problem of finding the shortest paths from all nodes to a given destination node.

Take some node, say  $n = |N|$ , as a root node. Put a demand of  $n - 1$  at this node (that is,  $b_n = -(n - 1)$ ) and a supply of one unit at every other node ( $b_1 = \dots = b_{n-1} = 1$ ), so that total supply and demand are equal. Let the cost  $c_{ij}$  of arc  $(i, j)$  be given by its length and solve this minimum cost flow problem by the network simplex algorithm. Then the shortest path from any node  $i$  to  $n$  is given by following the arcs of the spanning tree from  $i$  to  $n$ .

Let  $v_i$  be the shortest distance from node  $i$  to the root node  $n$ . These quantities are known as **labels**. Algorithms which systematically determine their values in some order are called **label-setting algorithms**. Algorithms which find their values through a sequence of iterations are called **label-correcting algorithms**.

### 11.4 Bellman's equations

Consider the minimum cost flow problem formulation of the shortest path problem. Suppose that the  $\lambda_i$ s are the optimal dual variables associated with the optimal spanning tree solution. Recall that on each arc of the tree, where  $f_{ij} > 0$ , we must have

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

Taking  $\lambda_n = v_n = 0$  and adding these equalities along a path from  $i$  to  $n$  we conclude that  $\lambda_i = v_i$ , the length of the shortest path from  $i$  to  $n$ .

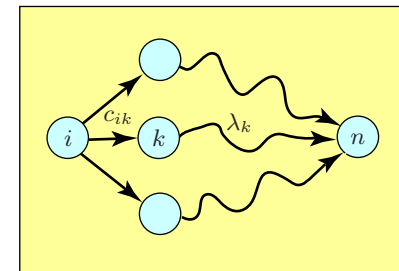
Moreover, as  $b_1 = \dots = b_{n-1} = 1$ , the dual problem, with  $\lambda_n = 0$ , is

$$\text{maximize } \sum_{i=1}^{n-1} \lambda_i, \quad \text{subject to } \lambda_i \leq c_{ij} + \lambda_j \text{ for all } (i,j) \in A.$$

It follows that in the optimal solution,  $\lambda$ , if all components are fixed except for  $\lambda_i$ , then we should set  $\lambda_i$  as large as possible subject to the feasibility constraint. That is,  $\lambda_i$  satisfies

$$\lambda_i = \min_{k: (i,k) \in A} \{c_{ik} + \lambda_k\}, \quad i = 1, \dots, n - 1.$$

with  $\lambda_n = 0$ . These are known as **Bellman's equations**.



The idea is that if we are looking for the shortest path from  $i$  to  $n$  then we should choose the first arc of the path  $(i, k)$  by minimizing over path lengths  $c_{ik} + \lambda_k$ . This method is also known as **dynamic programming**.

## 12 Algorithms for Shortest Path Problems

### 12.1 Bellman-Ford algorithm

Let  $v_i(t)$  be the length of the shortest path from  $i$  to  $n$  which uses at most  $t$  arcs. We have  $v_n(t) = 0$  for all  $t$  and  $v_i(0) = \infty$  for all  $i \neq n$ . Then

$$v_i(t+1) = \min_{k: (i,k) \in A} \{c_{ik} + v_k(t)\}, \quad i = 1, \dots, n-1$$

defines the **Bellman-Ford algorithm** for solving the shortest path problem.

It is a label-correcting algorithm. If we assume that there are no negative length cycles then  $v_i(n-1) = v_i$  and allowing further additional arcs cannot reduce the length, so that  $v_i(n) = v_i(n-1)$ .

The Bellman-Ford algorithm has running time  $O(mn)$ , where  $n$  is the number of nodes and  $m$  is the number of arcs, since there are at most  $n$  iterations and at each iteration each arc is examined once.

To find the shortest paths and not just their length  $v_i$  we could record a successor node,  $s(i)$  to  $i$  as the first node along the shortest path from  $i$  to  $n$ . Whenever we have  $v_i(t+1) < v_i(t)$ , we delete the old successor of  $i$ , if any, and let  $s(i)$  be such that  $v_i(t+1) = c_{is(i)} + v_{s(i)}(t)$ .

### 12.2 Dijkstra's algorithm

Dijkstra's algorithm is a label-setting algorithm. It can only be applied when all arc lengths  $c_{ij}$  are non-negative. The idea is to collect up nodes in the order of their increasing shortest path lengths, starting from the node with shortest path length. To ease exposition, suppose all arcs are present, taking  $c_{ij} = \infty$  for some node pairs if necessary.

**Lemma 12.1** Suppose that  $c_{ij} \geq 0$  for all  $i, j$ . Let  $\ell \neq n$  be such that

$$c_{\ell n} = \min_{i \neq n} c_{in}.$$

Then  $v_\ell = c_{\ell n}$  and  $v_\ell \leq v_k$  for all  $k \neq n$ .

**Proof** A path from node  $k$  to  $n$  has a last arc, say  $(i, n)$  whose length  $c_{in}$  is at least  $c_{\ell n}$ . For node  $\ell$ , we also have  $v_\ell \leq c_{\ell n}$ . Thus  $v_\ell = c_{\ell n} \leq v_k$  for all  $k \neq n$ . ■

Dijkstra's algorithm is

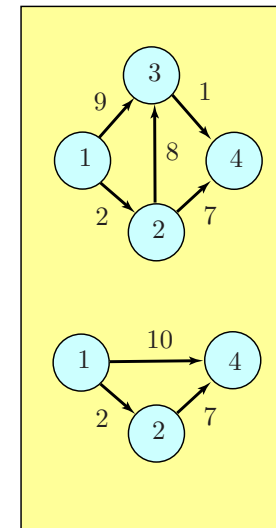
1. Find a node  $\ell \neq n$  such that  $c_{\ell n} \leq c_{in}$  for all  $i \neq n$ . Set  $v_\ell = c_{\ell n}$ .
2. For every node  $i \neq \ell, n$  set  $c_{in} = \min\{c_{in}, c_{i\ell} + c_{\ell n}\}$ .
3. Remove node  $\ell$  from the graph and return to step 1 to apply the same steps to the new graph.

**Remarks**

1. The running time is  $O(n^2)$  where  $n$  is the number of nodes. This follows since there are  $n$  iterations each involving a comparison and update of arc lengths from each remaining node.
2. In the case of dense graphs, with arcs numbering  $m = O(n^2)$ , this improves on the Bellman-Ford algorithm (which has computational complexity  $O(mn)$ ). Dijkstra's algorithm is the best possible since any shortest path algorithm would need  $\Omega(n^2)$  operations just to examine every arc at least once.

**Example** ( $n = 4$ )

1. Iteration 1 gives  $\ell = 3$  and  $v_3 = 1$ .
2. Modify arc lengths  
 $c_{14} = \min\{\infty, 9 + 1\} = 10$  and  
 $c_{24} = \min\{7, 8 + 1\} = 7$ .
3. Eliminate node  $\ell = 3$  from the graph.
4. Iteration 2 gives  $\ell = 2$  and  $v_2 = 7$ .
5. Modify arc length  
 $c_{14} = \min\{10, 2 + 7\} = 9$ .
6. Eliminate node  $\ell = 2$ .
7. Node 1 is only node remaining so set  
 $v_1 = c_{14} = 9$ .





### 12.3 Reformulation with non-negative $c_{ij}$

If  $v_i$  ( $i \neq n$ ) is the shortest path length from node  $i$  to node  $n$  then from the Bellman equations (dual feasibility) we have that

$$v_i \leq c_{ij} + v_j, \quad \text{for all } (i, j).$$

So that  $\bar{c}_{ij} = c_{ij} + v_j - v_i \geq 0$  are non-negative arc lengths and along any path visiting nodes  $i_1, \dots, i_p$

$$\sum_{k=1}^{p-1} \bar{c}_{i_k i_{k+1}} = \sum_{k=1}^{p-1} (c_{i_k i_{k+1}} + v_{i_{k+1}} - v_{i_k}) = v_{i_p} - v_{i_1} + \sum_{k=1}^{p-1} c_{i_k i_{k+1}}.$$

Hence, the shortest paths under the new arc lengths are the same as those under the original (possibly negative) arc lengths.

This is useful when we wish to solve the all-pairs problem, that is, to find the shortest distances between all pairs of nodes. Here, if we have negative arc lengths, we would use the Bellman-Ford algorithm to obtain  $v_i$  for a given root node and then apply Dijkstra's algorithm to solve the  $n - 1$  remaining problems using the non-negative costs,  $\bar{c}_{ij}$  which are defined in terms of the  $v_i$  just calculated.

For dense graphs, with  $m = O(n^2)$ , the overall complexity is

$$O(n^3) + (n - 1)O(n^2) = O(n^3).$$

This compares with a computational complexity of  $O(n^4)$  for the Bellman-Ford algorithm to solve the all-pairs problem.

### 12.4 Minimal spanning tree problem

Given a network  $(N, A)$ , with cost  $c_{ij}$  associated with arc  $(i, j) \in A$ , find the spanning tree of least cost. This problem arises, for example, when we wish to design a communications network connecting a set of cities at minimal cost.

**Theorem 12.1 (MST property)** *Let  $U$  be some proper subset of the set of nodes,  $N$ . If  $(u, v)$  is an arc of least cost such that  $u \in U$  and  $v \in N \setminus U$  then there is a minimal cost spanning tree that includes  $(u, v)$  as an arc.*

**Proof** Suppose to the contrary that there is no minimal spanning tree that includes  $(u, v)$ . Let  $T$  be any minimal spanning tree. Adding  $(u, v)$  to  $T$  must introduce a cycle, since  $T$  is a spanning tree. Thus, there must be another arc  $(u', v')$  in  $T$  such that  $u' \in U$  and  $v' \in N \setminus U$ . If not, there would be no way for the cycle to get from  $u$  to  $v$  without following the arc  $(u, v)$  a second time.

Deleting the arc  $(u', v')$  breaks the cycle and yields a spanning tree  $T'$  whose cost is certainly no greater than the cost of  $T$  since by assumption  $c_{uv} \leq c_{u'v'}$ . Thus,  $T'$  contradicts our assumption that there is no minimal spanning tree that includes  $(u, v)$ . ■

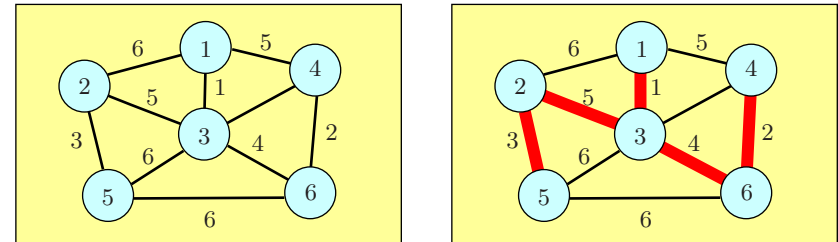
### 12.5 Prim's greedy algorithm for MST

Labels the nodes,  $N = \{1, 2, \dots, n\}$  and set  $U = \{1\}$ . Now construct  $U$  recursively using the property above.

1. Find the cheapest arc  $(u, v)$  connecting  $U$  and  $N \setminus U$  (breaking ties at random).
2. Add  $v$  to  $U$  and repeat until  $U = N$ .

Prim's algorithm takes  $O(n^2)$  steps. Suppose each time we start step 1 we already know the shortest distance between  $U$  and every  $j \notin U$ , say  $c_{Uj} = \min_{i \in U} c_{ij}$ . Then it takes no more than  $n$  comparisons to find the lowest cost arc between  $U$  and  $N \setminus U$  (by comparing all the  $c_{Uj}$  for  $j \in N \setminus U$ ). Having found a node  $v$  to add to  $U$ , we can now find the shortest distance between  $U' = U + \{v\}$  and any  $j$  in  $N \setminus U'$ , say  $c_{U'j} = \min\{c_{vj}, c_{Uj}\}$ . Thus each step of the algorithm requires at most  $n$  comparisons, and the algorithm has  $n - 1$  steps.

**Example.** In this example, Prim's algorithm adds arcs in the sequence  $\{1, 3\}$ ,  $\{3, 6\}$ ,  $\{6, 4\}$ ,  $\{3, 2\}$ ,  $\{2, 5\}$ .



## 13 Branch and Bound

### 13.1 Integer programming problems

Can one obtain an optimal solution to an **integer linear program** (pathILP) by rounding the solution to its LP relaxation? No, not easily.

#### Rounding may not be optimal

Consider the problem

$$\text{maximize } z = x_1 + 5x_2,$$

subject to

$$x_1 + 10x_2 \leq 20, \quad x_1 \leq 2, \quad x_1, x_2 \in \{0, 1, \dots\}.$$

Without the integer constraints the optimum is:

$$x_1 = 2, \quad x_2 = \frac{9}{5}; \quad z = 11.$$

If we round  $x_2$  (in the feasible direction) then  $x_2 = 1, z = 7$ .

However, the optimal solution is

$$x_1 = 0, \quad x_2 = 2; \quad z = 10.$$

The best integer solution is not the closest to the best non-integer solution.

#### Rounding may not be feasible

Suppose a LP has optimal solution  $x_1 = 6\frac{1}{2}, x_2 = 10$ , with feasible set

$$-x_1 + x_2 \leq 3\frac{1}{2}, \quad x_1 + x_2 \leq 16\frac{1}{2}$$

Neither  $x_1 = 6$  nor  $x_1 = 7$  is feasible.

#### A possible algorithm

If all variables must be integers and the feasible set is bounded, then there are only a finite number of feasible solutions. So a possible algorithm is

1. Try all solutions.
2. Compare them.
3. Pick the best.

However, there may be very many solutions to compare. We would like a more efficient method of choosing between all possible solutions.

### 13.2 Branch and Bound technique

A **Branch and Bound** technique can be used for many types of problem. Its speed is very much problem dependent. The idea is to divide up the space of all solutions in some sensible way so that there is a good chance we can reject large subsets of nonoptimal solutions without actually evaluating them. Suppose we wish to solve the problem:

$$\text{minimize } f(x), \quad \text{subject to } x \in X,$$

where  $X$  is a finite set of feasible solutions. The method takes a ‘divide-and-conquer’ approach, in which problems are broken into subproblems. The original problem is broken into one or more subproblems, the  $i$ th of which is to minimize  $f(x)$  over  $x \in X_i$ . Subsequently, we break  $X_i$  into subproblems, continuing in this way until a subproblem is easy to solve.

We also suppose that for any subproblem, in which  $f$  is minimized over a  $x \in X'$ , where  $X'$  is a subset of  $X$ , we can calculate a lower bound such that

$$\ell(X') \leq \min_{x \in X'} f(x).$$

#### Branch and Bound algorithm

The algorithm keeps a list  $L$  of outstanding (active) subproblems and the cost  $U$  of the best feasible solution found so far.

**0. Initialize step.** Set  $U = \infty$ . Discard any obviously infeasible solutions. Treat the remaining solutions as one subset. Go to Step 2.

**1. Branch step.** Use some branch rule to select one of the remaining subsets and break it into two or more subsets. Two common rules are:

**Best bound rule.** We partition the subset with the lowest bound, hoping that this gives the best chance of an optimal solution and of being able to discard other, larger, subsets by the fathom test.

**Newest bound rule.** We partition the most recently created subset, breaking ties with best bound rule. This has book-keeping advantages in that we don’t need to keep jumping around the tree too often. It can save some computational effort in calculating bounds.

**2. Bound step.** For each new subset,  $Y$ , calculate  $\ell(Y)$ .

**3. Fathom step.** Exclude from further consideration any new subsets,  $Y$ , such that

- $\ell(Y) \geq U$ .
- $Y$  contains no feasible solutions.
- We can find an optimal solution to the problem of minimizing  $f$  over  $Y$ , say  $x'$ , so  $\ell(Y) = f(x')$ . If  $\ell(Y) \geq U$ , we eliminate  $Y$  by test (a). If  $\ell(Y) < U$ , we reset  $U = \ell(Y)$ , store  $x'$  as the best solution so far and re-apply test (a) to all other active subsets.

**4. Stopping rule.** If there are no remaining active subsets, stop. The best solution obtained so far is optimal. Otherwise, go to Step 1.

### 13.3 A knapsack problem

A hiker wishes to take some items on a journey. Which items he should take so that the total value is at least 9, but the total weight is a minimum?

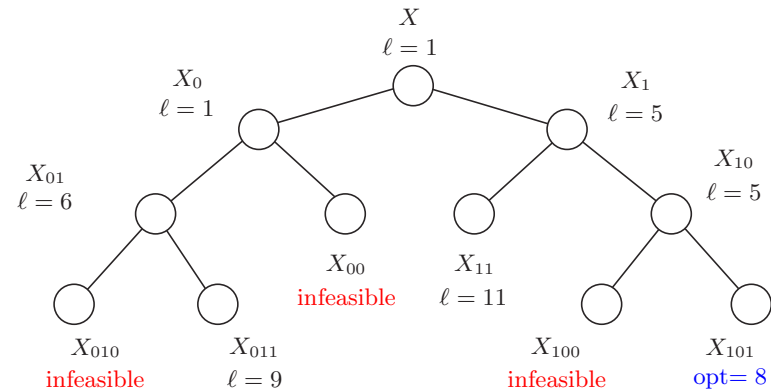
$i$	1	2	3	4
$v_i$	5	5	4	2
$w_i$	5	6	3	1
$w_i/v_i$	1	1.2	0.75	0.5

Each of the 16 subsets of  $\{1, 2, 3, 4\}$  is a possible solution. However, only 8 of these are feasible. The hiker's problem is

$$\text{minimize } \sum_{i=1}^4 x_i w_i, \quad \text{subject to } \sum_{i=1}^4 x_i v_i \geq 9, \text{ and } x_i \in \{0, 1\}, \text{ for all } i.$$

- Starting with  $X$  as the only subset, we take items in index order until the total value is at least 9. This gives  $U = 11$ . Since we must include at least one item and the least item has weight 1, we have  $\ell(X) = 1$ .
- Break  $X$  into two subproblems,  $X_1$  and  $X_0$ , such that the hiker does or does not include item 1 in his backpack. Clearly  $\ell(X_0) = 1$  (since he must include at least one of the remaining items) and  $\ell(X_1) = 5$  (since item 1 is in the backpack). Neither subproblem can be eliminated by tests (a) or (b). So  $L = \{X_0, X_1\}$ .
- Break  $X_0$  into the subproblems  $X_{01}$ ,  $X_{00}$ , such that the backpack does not include item 1, and does or does not include item 2.  $X_{00}$  is infeasible and so we eliminate it. For  $X_{01}$  we have  $\ell(X_{01}) = 6$ . Now  $L = \{X_{01}, X_1\}$ .

- Break  $X_1$  into two subproblems,  $X_{11}$  and  $X_{10}$ , which contain item 1, and do or do not contain item 2. We have  $\ell(X_{10}) = 5$  and  $\ell(X_{11}) = 11$ . Hence  $X_{11}$  can be eliminated by test (a) and  $L = \{X_{01}, X_{10}\}$ .
- Break  $X_{10}$  into subproblems  $X_{101}$  and  $X_{100}$ , which contain item 1, do not contain item 2, and do or do not contain item 3. We eliminate  $X_{100}$  by test (b). Clearly problem  $X_{101}$  is solved by  $x_1 = \{1, 3\}$ ,  $f(x_1) = 8$ . So following (c) we set  $U = 8$  and  $L = \{X_{01}\}$ .
- Break  $X_{01}$  into subproblems  $X_{011}$  and  $X_{010}$ . Since  $\ell(X_{011}) > U$ , we eliminate  $X_{011}$  by test (a). As for  $X_{010}$  it is infeasible, and so eliminated by test (b).  $L$  is now empty and so we are done. The optimal solution  $x_1 = \{1, 3\}$ .



**Note 1.** There is a trade off between the number of iterations that the method takes and the effort we put into calculating the bounds.

For example, in calculating  $\ell(X_0)$  we could notice that for items 2, 3 and 4 the value of  $w_i/v_i$  is  $6/5, 3/4, 1/2$ . So to fill the backpack to total value 9, without using item 1, requires a weight of at least  $1 + 3 + 3(6/5) = 38/5$ , and we can put  $\ell(X_0) = 7.6$ . Similarly,  $\ell(X_1) = 5 + 1 + 2(3/4) = 7.5$ .

By a similar calculation we have  $\ell(X_{01}) = 6 + 1 + 2(3/4) = 8.5$ . So at after Step 5 we could eliminate  $X_{01}$  and so Step 6 would be unnecessary.

**Note 2.** Suppose we want all optimal solutions. In this case, we replace the fathom test (a) with  $\ell(Y) > U$  and change fathom test (c) so that if  $\ell(Y) = U$  we add additional incumbent solutions to the collection, and if  $\ell(Y) < U$  we throw away all current incumbent solutions and replace by new one(s).

# 14 Integer Programming

## 14.1 Dakin's method

Consider the problem

$$\text{minimize } z = \sum_{j=1}^n c_j x_j$$

subject to

$$\sum_{j=1}^n a_{ij} x_j \geq b_i, \quad i = 1, \dots, m$$

$$x_j \geq 0, \quad j = 1, \dots, n$$

$$x_j \in \mathbb{Z}, \quad j = 1, \dots, I \quad (I \leq n)$$

If only some but not all of the variables ( $I < n$ ) are constrained to be integer-valued then we say the optimization problem is a **mixed integer program**. If all the variables are constrained to be integer-valued ( $I = n$ ) then it is said to be a **pure integer program**.

**Dakin's method** applies to both pure and mixed forms of integer programs. It nicely combines the primal and dual simplex algorithm with Branch and Bound.

**Initialize step.** Recall that the problem without the integer constraints is the **linear programming relaxation**. We use this as our method of generating lower bounds.

We set  $U = \infty$  and solve the LP relaxation of the problem with the primal simplex method. If optimal solution, say  $\hat{x}$ , has  $\hat{x}_j \in \mathbf{Z}$ , for  $j = 1, \dots, I$  then stop. This solution is optimal. Otherwise, pick a variable, say  $x_j$ , that is meant to be integer but is not. Partition into two subsets by adding one or other of the constraints

$$x_j \leq \lfloor \hat{x}_j \rfloor \quad \text{or} \quad x_j \geq \lceil \hat{x}_j \rceil.$$

Observe that  $\hat{x}_i$  violates both of these constraints and that if  $\hat{x}$  is the unique optimal solution to the LP relaxation then the optimal cost of each of the two subproblems will be strictly larger.

**Bound step.** Solve the resulting LP relaxation of the problem with the new constraint. Use the dual simplex method, starting with the basic solution that was optimal before adding the new constraint. The dual simplex method works well for this task. Use the newest bound rule for greatest efficiency.

**Fathoming.** The fathoming tests become:

1. If new optimal solution has  $\ell(Y) \geq U$ .
2. If dual simplex implies that there are no feasible solutions.
3. If the optimal solution is feasible for the original problem, i.e., has integer values for  $x_1, \dots, x_I$  and if  $\ell(Y) < U$ , then reset  $U = \ell(Y)$  and store  $x$  as the incumbent solution.

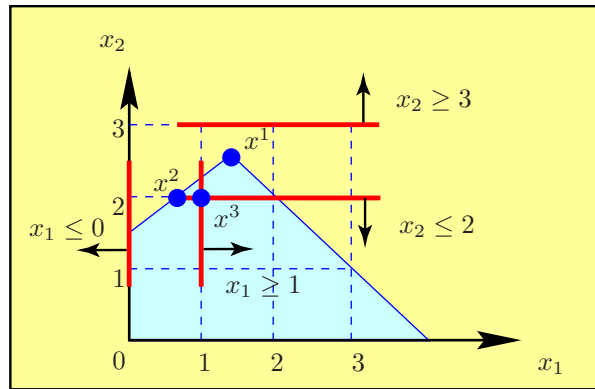
## 14.2 Example of Dakin's method

Consider the pure integer programming problem

$$\text{minimize } x_1 - 2x_2,$$

$$\text{subject to } -4x_1 + 6x_2 \leq 9, \quad x_1 + x_2 \leq 4, \quad x_1, x_2 \geq 0, \quad x_1, x_2 \in \mathbb{Z}.$$

1. Set  $U = \infty$ .
2. Solve the LP relaxation to obtain  $x^1 = (1.5, 2.5)$  with optimal cost  $-3.5$ .
3. Create two subproblems by adding the constraints  $x_2 \geq 3$  (subproblem  $P_1$ ) or  $x_2 \leq 2$  (subproblem  $P_2$ ).
4. The LP relaxation of  $P_1$  is infeasible and we can eliminate this subset of possible solutions.
5. The optimal solution to  $P_2$  is  $x^2 = (0.75, 2)$ , with optimal cost of  $-3.25$ .
6. Partition  $P_2$  into two subproblems according to the additional constraint  $x_1 \geq 1$  (subproblem  $P_3$ ) or  $x_1 \leq 0$  (subproblem  $P_4$ ).
7. The optimal solution to  $P_3$  is  $x^3 = (1, 2)$  which is integer and therefore record this as the best solution so far and set  $U = -3$ .
8. The optimal solution to  $P_4$  is  $(0, 1.5)$  with optimal cost  $-3 \geq U$ . So delete  $P_4$ .
9. There are no more unfathomed subproblems so we stop with optimal solution  $x^3 = (1, 2)$ .



### 14.3 Decision variables

We have seen pure and mixed integer programming problems. A further class of practically useful programs arises from the constraints that the variables may take one of the two values 0 or 1. We met one of these in Section 13.3. Such problems are called binary, or **zero-one programs**. Their interpretation as decision variables can capture many modelling questions. We give some examples.

#### Binary choice, the 0–1 knapsack problem

We have  $n$  items with item  $j$  having a size  $w_j$  and value  $v_j$ . Given a bound  $K$  on the overall size of our knapsack, how should we select the items to maximize the total value?

$$\text{maximize } \sum_{j=1}^n v_j x_j, \quad \text{subject to } \sum_{j=1}^n w_j x_j \leq K, \quad x_j \in \{0, 1\}, \quad j = 1, \dots, n.$$

#### Contingent decisions

A typical feature of discrete problems is that the variables are dependent in that decision  $x$  can be made only if  $y$  is also made. We can capture this dependence by the constraint  $x \leq y$ . So that if  $y = 0$  then  $x = 0$ .

#### Example: the facility location problem

Suppose that there are  $n$  potential facilities and that it costs  $c_j$  to build facility  $j = 1, \dots, n$ . There are  $m$  clients wishing to use one of the facilities and suppose that it costs  $d_{ij}$  for client  $i = 1, \dots, m$  to use facility  $j$ .

The **facility location problem** is to choose which facilities to build in order to minimize the total cost of construction and use. We can formulate the problem as follows.

$$\text{minimize } \sum_{j=1}^n c_j y_j + \sum_{i=1}^m \sum_{j=1}^n d_{ij} x_{ij}$$

subject to

$$\begin{aligned} \sum_{j=1}^n x_{ij} &= 1, \quad i = 1, \dots, m \\ x_{ij} &\leq y_j, \quad i = 1, \dots, m, \quad j = 1, \dots, n \\ x_{ij}, y_j &\in \{0, 1\}, \quad j = 1, \dots, n. \end{aligned}$$

Note that feasible solutions satisfy  $x_{ij} = 0$  for all  $i = 1, \dots, m$  whenever  $y_j = 0$ . So that clients do not attempt to use a facility that is not constructed.

#### Relations between variables

Consider constraints of the form

$$\sum_{j=1}^n x_j \leq 1, \quad \text{or} \quad \sum_{j=1}^n x_j = 1,$$

where all variables are binary. The first implies that at most one of the variables can take the value 1. The second implies that precisely one of the variables must be 1.

#### Restricted ranges of values

Suppose a variable  $x$  is restricted to the values  $\{a_1, \dots, a_m\}$ . Introduce  $m$  binary variables  $y_j$ ,  $j = 1, \dots, m$  together with the constraints

$$x = \sum_{j=1}^m a_j y_j, \quad \sum_{j=1}^m y_j = 1, \quad y_j \in \{0, 1\}.$$

# 15 Travelling Salesman Problem

## 15.1 Categories of algorithms

Given an undirected graph  $G = (N, A)$  consisting of  $n$  nodes and  $m$  arcs together with costs  $c_{ij}$  for each arc  $\{i, j\} \in A$ , the **travelling salesman problem** (TSP) is to find a tour of minimum cost.

1. **Exact algorithms** are guaranteed to find an optimal solution but may take an exponential number of iterations. An exact algorithm for TSP is to write it as an ILP and solve it using branch and bound.
2. **Approximation algorithms** have polynomial worst-case time complexity, supplying a suboptimal solution with a guaranteed bound on the degree of suboptimality. We shall look at such an algorithm for TSP, based on the minimum spanning tree.
3. **Heuristic algorithms** supply suboptimal solutions without any bound on their quality. They do not necessarily provide polynomial running times, but empirically they often provide a successful tradeoff between optimality and speed. We look at two approaches that have been used for TSP: local search and simulated annealing.

## 15.2 Exact methods

Set  $x_{ij} = 1$  or  $0$  as  $(i, j) \in A$  is or is not present in the tour. Define

$$\delta(S) = \{(i, j) \in A : i \in S, j \notin S\}.$$

For a tour there must be two arcs incident to every node so

$$\sum_{(i,j) \in \delta(\{i\})} x_{ij} = 2, \quad i \in N. \tag{15.1}$$

Furthermore, for any partition of the nodes into subsets  $S$  and  $N \setminus S$  there must be at least two edges connecting  $S$  and  $N \setminus S$ . So we must also have

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 2, \quad \text{for all } S \subset N, \text{ such that } S \neq \emptyset \text{ or } N. \tag{15.2}$$

The so-called **cutset formulation of the TSP** is therefore

$$\text{minimize } \sum_{(i,j) \in A} c_{ij} x_{ij}$$

subject to (15.1), (15.2) and  $x_{ij} \in \{0, 1\}$ . Notice that we have exponentially many constraints, since there are  $2^n - 2$  constraints of type (15.2).

Alternatively, we can replace (15.2) by

$$\sum_{(i,j) : i,j \in S} x_{ij} \leq |S| - 1, \quad \text{for all } S \subset N \text{ such that } S \neq \emptyset \text{ or } N. \tag{15.3}$$

This constraint ensures there is no cycle involving less than all  $n$  nodes. Again we have an exponential number of constraints. This is called the **subtour elimination formulation of the TSP**. The LP relaxations of these two formulations have the same feasible sets (though this is not obvious).

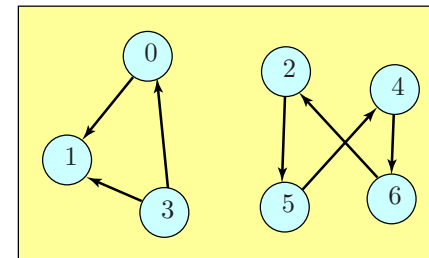
## 15.3 Polynomial formulation of TSP

Think of the undirected formulation of the TSP. The the salesman must on leaving a city he must next visit one and only one city, and, similarly, on arriving at a city he must have come from one and only one city. Therefore we must have

$$\sum_{j : (i,j) \in A} x_{ij} = 1, \quad i = 0, 1, \dots, n-1 \tag{15.4}$$

$$\sum_{i : (i,j) \in A} x_{ij} = 1, \quad j = 0, 1, \dots, n-1. \tag{15.5}$$

These constraints are not sufficient to ensure that the solutions do not consist of several subtours such as is shown here.



Consider a tour  $s_0 = 0, s_1, s_2, \dots, s_{n-1}$ . Let  $t_i$  be the position in the tour at which city  $i$  is visited. So, we have  $t_0 = 0, t_{s_1} = 1$ , and in general,

$$t_{s_i} = i, \quad i = 0, 1, \dots, n-1.$$

We require that if  $x_{ij} = 1$  then

$$t_j = t_i + 1.$$

Also,  $t_i$  is an integer between 0 and  $n - 1$ . Hence,

$$t_j \geq \begin{cases} t_i - (n - 1) & \text{if } x_{ij} = 0 \\ t_i + 1 & \text{if } x_{ij} = 1. \end{cases}$$

These constraints can be written as

$$t_j \geq t_i + 1 - n(1 - x_{ij}), \quad i \geq 0, j \geq 1, i \neq j. \quad (15.6)$$

It turns out that these constraints also rule out subtours. To see this, suppose we have a solution which satisfies these constraints and consists of two or more subtours. Consider the subtour that does not include city 0, and suppose it has  $r \geq 2$  arcs. Summing the constraints over the arcs of this subtours leads to the condition

$$0 \geq r,$$

and hence there can only be a single tour visiting all the cities.

Thus the TSP can be formulated as an ILP in  $n^2 + n$  variables and  $2n + n(n - 1)$  constraints. Namely,

$$\text{minimize } \sum_{i,j} c_{ij} x_{ij}$$

subject to (15.4), (15.5), (15.6),  $x_{ij} \in \{0, 1\}$ ,  $t_0 = 0$  and  $t_i \in \{0, 1, 2, \dots\}$ .

## 15.4 Solution using branch and bound

Notice that by dropping the constraints establishing the lack of subtours we are left with an assignment problem, which can be efficiently solved by the network simplex to provide a lower bound on the optimal solution.

We need not worry about the relaxation to non-integral solutions since the network simplex algorithm will always find an integer solution. Thus we consider

$$\text{minimize } \sum_{i,j} c_{ij} x_{ij}, \quad \text{subject to (15.4), (15.5) and } x_{ij} \in \{0, 1\}.$$

If the optimal solution corresponds to a tour visiting all the cities then it is optimal for the original travelling salesman problem.

If not, we continue with a branch and bound algorithm, using a branching rule that breaks the problem in two by an additional constraint of the form  $x_{ij} = 0$ .

Think of this as re-setting a cost,  $c_{ij} = \infty$ . The addition of such a constraint leaves us with a valid travelling salesman problem and a valid assignment problem which provides a corresponding lower bound.

A natural way to select an  $x_{ij} = 0$  constraint is to choose one or more of the subtours and eliminate one of their arcs.

If the current assignment problem has a unique optimal solution, this solution becomes infeasible with the addition of a constraint during branching. Hence, the optimal cost of each subproblem is strictly larger, and increasing lower bounds are obtained.

## 15.5 Approximation algorithm for the TSP

**Definition 15.1** An  $\epsilon$ -*approximation algorithm* for a minimization problem with optimal cost  $Z_{opt}$  runs in polynomial time and returns a feasible solution with cost  $Z_{app}$ , such that

$$Z_{app} \leq (1 + \epsilon)Z_{opt}.$$

It is usually difficult to determine approximation algorithms for any  $\epsilon > 0$  and we shall develop one for the TSP only for  $\epsilon = 1$ , and when the costs  $c_{ij}$  satisfy the triangle inequality.

Consider the undirected TSP with costs satisfying

$$c_{ij} \leq c_{ik} + c_{kj}, \quad \text{for all } i, j, k.$$

Now suppose that  $M$  is the cost of the minimal spanning tree. This can be obtained easily using Prim's greedy algorithm. Consider any starting node and traverse the minimal spanning tree to visit all the nodes. This uses each arc of the spanning tree exactly twice, with total cost  $2M$ .

This path can be converted into a tour visiting all the cities by skipping any intermediate node that has already been visited. By the triangle inequality, the resulting tour will have cost bounded above by  $2M$ . Also, every tour contains a spanning tree (since dropping any one arc leaves a spanning tree) and so has cost at least  $M$ .

Thus a straight-forward algorithm based on the minimal spanning gives

$$Z_{app} \leq 2M \leq 2Z_{opt}.$$

It is an approximation algorithm with  $\epsilon = 1$ . Observe the essential importance played by the triangle inequality.

## 16 Heuristic Algorithms

### 16.1 Heuristics for the TSP

**Nearest neighbour heuristic.** Start at some city and then visit the nearest city. Continue to visit the nearest city that has not yet been visited, continuing until a tour is complete.

Although usually rather bad, such tours may only contain a few severe mistakes. They can serve as good starts for local search methods.

**Cheapest insertion heuristic.** This is also a greedy algorithm. Start with a single node and then, one by one, add the node whose insertion makes the smallest increase to the length of the tour.

**Furthest insertion heuristic.** Insert the node whose minimal distance to the existing tour node is greatest. The idea is to determine the overall layout of the tour early in the process.

**Savings heuristic.** Rank the arcs in ascending order of cost. Add the arcs in this order, so long as they do not violate any constraints, and until all cities have been visited.



### 16.2 Neighbourhood search

Consider the general problem

$$\text{minimize } c(x), \text{ subject to } x \in X.$$

Suppose that for any point  $x \in X$  we have a set of 'neighbouring points',  $N(x) \subset X$ . The basic approach of local search is as follows

1. Select some  $x \in X$ .

2. Evaluate  $c(x)$ .
3. Pick some  $y \in N(x)$  and evaluate  $c(y)$ .  
If  $c(y) < c(x)$  then select  $y$  as new value for  $x$  and return to step 2.  
If there is no such  $y \in N(x)$  with  $c(y) < c(x)$  then stop with solution  $x$ .

#### Remarks

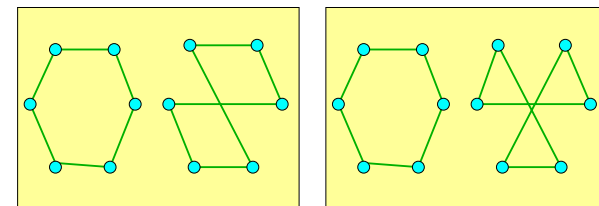
1. A specific implementation must specify, in a problem-dependent way:
  - (a) the neighbourhood sets,  $N(x)$ , for all  $x \in X$ ;
  - (b) the procedure for selecting  $y \in N(x)$ .
2. There are various ways to modify local search.
  - (a) We might use some rule to guess a good starting point or try multiple starting points.
  - (b) We might choose the best neighbour  $y \in N(x)$  with least value of  $c(y)$  not just the first  $y$  that improves the solution.
  - (c) We might choose the best neighbour amongst the first  $r$  considered.
3. The simplex algorithm for linear programs is a local search method. We can say that two basic feasible solutions are neighbours if they are connected by an edge of the constraint set.

In linear programming any local optimum is the global optimum.

### 16.3 Neighbourhood search methods for TSP

Consider the TSP. By a feasible solution  $x$ , we mean the indicator function for the arcs in some tour of the network.

There is a fairly natural family of neighbourhoods for any tour  $x$  generated by the operation of removing any  $k \geq 2$  arcs from the tour and replacing them with  $k$  new arcs that also make a new tour. For example, when  $k = 2$  (known as **2OPT**) each tour has  $O(n^2)$  neighbours. For  $k = 3$  there are  $O(n^3)$  neighbours for each tour  $x$ .





Note that it only takes time that is  $O(1)$  to compute the change in cost between neighbouring tours.

Empirical evidence is that 3OPT performs better than 2OPT, but there is little further gain in taking  $k > 3$ .

In general, there is trade-off of solution quality and speed. The larger the neighbourhoods  $N(x)$  the fewer local minima there are, and the better the solution. However, more work must be done per iteration so the method is slower for larger neighbourhoods.

In practice, we fix on a certain neighbourhood size but then repeat the algorithm with a variety of starting values.

### Example TSP using 2OPT

Suppose we have a distance matrix

	A	B	C	D	E
A	-	1	0	4	4
B	4	-	1	0	4
C	4	4	-	1	0
D	0	4	4	-	1
E	1	0	4	4	-

A feasible solution is a cycle that visits all the nodes (without re-using the arcs). Here are the  $4! = 24$  feasible tours and costs  $c(x)$

ABCDE ( 5)	ACBDE ( 6)	ADBCE (10)	AEBDC ( 6)
ABCED ( 6)	ACBED (12)	ADBEC (20)	AEBDC (12)
ABDCE ( 6)	ACDBE (10)	ADCBE (17)	AECBD (12)
ABDEC (10)	ACDEB ( 6)	ADCEB ( 9)	AECDB (17)
ABECD (10)	ACEBD ( 0)	ADEBC (10)	AEDBC (17)
ABEDC (17)	ACEDB (12)	ADECB (17)	AEDCB (20)

So, ACEBD is global optimum but we can get stuck in ABCDE since none of its neighbours (under 2OPT) is better.

## 16.4 Simulated annealing

In this method we try to overcome difficulties of getting stuck in potentially poor local minima by permitting the algorithm to jump out of them.

The basic idea is to allow up jumps to worse neighbours in initial phase but get gradually more reluctant to permit up jumps.

The simulated annealing method permits a move from  $x$  to  $y \in N(x)$  with probability

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

where  $T$  starts large and decreases with each iteration. It can be shown, under suitable conditions, that if you start with  $T$  large enough and decrease it slowly enough then

$$\lim_{t \rightarrow \infty} P(x(t) \text{ is optimal}) = 1.$$

As motivation for this claim, imagine that all solutions have  $k$  neighbours, and at each step of the algorithm one of these neighbours is chosen at random. Then  $x$  moves next to its neighbour  $y$  with probability  $P_{xy} = p_{xy}/k$ . By checking that the ‘detailed balance’ of  $\pi(x)P_{xy} = \pi(y)P_{yx}$  holds we have that the stationary distribution of this Markov chain is

$$\pi(x) = \frac{e^{-c(x)/T}}{A}, \quad \text{where } A = \sum_{z \in X} e^{-c(z)/T}.$$

Let  $Y$  be the set of optimal solutions. In this stationary distribution  $\pi(Y)/(1 - \pi(Y)) \rightarrow \infty$  as  $T \rightarrow 0$ .

A common temperature schedule is to let  $T$  decrease with iteration number,  $t$ , according to

$$T(t) = \frac{c}{\log t}.$$

## 16.5 Genetic algorithms

**Genetic algorithms** can be applied to many problems, are often easy to implement, but may get stuck in a local optimum. We illustrate the basic steps for the TSP.

**Create a random initial state.** This is a population of tours, each of which is list of cities, analogous to a list of chromosomes.

**Evaluate fitness.** A value for fitness is assigned to each solution, e.g., the length of the tour.

**Reproduce.** Those chromosomes with a higher fitness value are more likely to reproduce offspring. E.g. ‘greedy crossover’ selects the first city of one parent, compares the cities leaving that city in both parents, and chooses the closer one to extend the tour. If one city has already appeared in the tour, choose the other city. If both cities have appeared, randomly select a non-selected city.

**Mutate.** Randomly swap over a pair of cities.

# 17 Two-person Zero-sum Games

## 17.1 Terminology

**Game theory** is the study of multi-player decision problems. The common theme is conflicts of interest between the different players. It assumes that each player plays the ‘best way’ he can. This may not happen in practice. Game-theoretic analysis is often more descriptive than prescriptive.

- **Players.** Labelled 1, 2, 3, ..., or I, II, III, ...
- **Moves.** A move is either a decision by a player or the outcome of a chance event.
- **Game.** A game is a sequence of moves, some of which may be simultaneous.
- **Payoffs.** At the end of a game each player receives a return. The payoff to each player is a real number. If a move has a random outcome we use an expected payoff.
- **Strategy.** A strategy is a description of the decisions that a player will make at all possible situations that can arise in the game.
- **Zero-sum.** The game is said to be **zero-sum** if the sum of the players’ payoffs is always zero.
- **Perfect information.** A game is said to have **perfect information** if at every move in the game all players know all the moves that have already been made (including any random outcomes.)

## 17.2 Two-person zero-sum games

We begin with zero-sum games between two players, labelled I and II. Each player has a finite collection of **pure strategies**. Player I has strategies  $I_1, I_2, \dots, I_n$  and player II has strategies  $II_1, II_2, \dots, II_m$ .

Let  $e_{ij}$  denote the (expected) payoff to player I when he uses strategy  $I_i$  and player II uses strategy  $II_j$ . The **normal form representation** of the game is given by the matrix of payoffs  $(e_{ij})$ .

This representation is in terms of strategies. It does not include detailed information about the sequences of moves or whether or not the players have perfect information.

## 17.3 Maximin criterion

In a non-zero-sum game, we must record both players’ payoffs, say  $e_1(i, j)$  and  $e_2(i, j)$ . So in a zero-sum game  $e_1(i, j) = -e_2(i, j) = e_{ij}$ . Player I wins what player II loses. Thus when player II tries to maximize his payoff he is also trying to minimize the payoff of player I. This means that player I should look at the payoff he would receive if he plays strategy  $I_i$ , i.e.,  $\min_j e_{ij}$ , and choose the strategy which has the largest of these minimum payoffs. This is known as the **maximin criterion**.

Using this criterion, player I can guarantee that his payoff is at least,  $v_L$ , the **lower value** of the game, where

$$v_L = \max_i \min_j e_{ij}.$$

Similarly, player II can guarantee that player I’s payoff is no more than,  $v_U$ , the **upper value** of the game,

$$v_U = \min_j \max_i e_{ij}.$$

### Example (a)

		$II_1$	$II_2$	$II_3$	$II_4$	$II_5$	$II_6$	min using $I_i$
$I_1$	(	1	1	-1	1	1	-1	-1
$I_2$		-1	1	-1	-1	1	-1	-1
$I_3$		1	1	1	-1	-1	-1	-1
max using $II_j$		1	1	1	1	1	-1	

Thus,  $v_L = \max\{-1, -1, -1\} = -1$  and  $v_U = \min\{1, 1, 1, 1, 1, -1\} = -1$ . When, as in this example, we have  $v_L = v_U$  for a pair of **pure strategies**, there is said to be a **saddle point solution**.

### Example (b)

		$II_1$	$II_2$	min using $I_i$
$I_1$	(	0	-1	-1
$I_2$		-1/2	0	-1/2
max using $II_j$		0	0	

Thus,  $v_L = \max\{-1, -1/2\} = -1/2$  and  $v_U = \min\{0, 0\} = 0$ .

## 17.4 Mixed strategies

In Example (b)  $v_L$  is strictly less than  $v_U$ . Suppose we enlarge the set of possible strategies by randomization, allowing player I to choose strategy  $I_i$  with probability  $p_i$  and player II to choose strategy  $II_j$  with probability  $q_j$ . We say that player I adopts the strategy  $\mathbf{p} = (p_1, p_2, \dots, p_n)$  and II adopts the strategy  $\mathbf{q} = (q_1, q_2, \dots, q_m)$ . The expected payoff to I is

$$e(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^n \sum_{j=1}^m p_i e_{ij} q_j.$$

Suppose that in Example (b) player I takes  $\mathbf{p} = (p, 1-p)$  and player II takes  $\mathbf{q} = (q, 1-q)$ . Then the expected payoff to player I is

$$\begin{aligned} e(p, q) &= 0p_1q_1 - 1p_1q_2 - \frac{1}{2}p_2q_1 + 0p_2q_2 \\ &= -p_1 - \frac{1}{2}q_1 + \frac{3}{2}p_1q_1 \\ &= \frac{3}{2}(p_1 - \frac{1}{3})(q_1 - \frac{2}{3}) - \frac{1}{3}. \end{aligned}$$

Define

$$v_L^M = \max_p \min_q e(p, q) \quad \text{and} \quad v_U^M = \min_q \max_p e(p, q).$$

Notice that if I plays  $p^* = (1/3, 2/3)$  and II plays  $q^* = (2/3, 1/3)$  then

$$e(p^*, q) = -1/3 \quad \text{for all } q, \quad \text{and} \quad e(p, q^*) = -1/3 \quad \text{for all } p.$$

It is clear that  $v_L^M \leq v_U^M$ . Playing  $p^*$  guarantees I at least  $-1/3$ , so  $v_L^M \geq -1/3$ . If II plays  $q^*$  then I will be restricted to no more than  $-1/3$ , so  $v_U^M \leq -1/3$ . Hence, we must have

$$v_L^M = v_U^M.$$

## 17.5 Minimax theorem

**Theorem 17.1 (Minimax theorem)** Consider a two-person zero-sum game in which I has  $n$  strategies and II has  $m$  strategies (both finite). Then

$$v_L^M = \max_p \min_q e(p, q) = \min_q \max_p e(p, q) = v_U^M.$$

If  $p^*$  and  $q^*$  achieve the maximin criterion of the theorem then

$$e(p^*, q^*) = v_L^M = v_U^M = v.$$

We say that  $v$  is the **value** of the game and that the value together with the optimal strategies,  $p^*$  and  $q^*$  are the **solution** to the game.

## 17.6 Dominating strategies

Consider the payoff matrix

$$\begin{array}{c|cccc} & II_1 & II_2 & II_3 & II_4 \\ \hline I_1 & 4 & 5 & 6 & 4 \\ I_2 & 4 & 2 & 3 & 4 \\ I_3 & 2 & 4 & 5 & 5 \end{array}$$

No matter which strategy player I follows his payoff is always the same or less under  $II_1$  than  $II_4$ . Thus, II will always do as well to use  $II_1$  rather than  $II_4$ . We say that  $II_1$  **dominates**  $II_4$ , and we may remove it from the strategy set. Similarly,  $II_2$  dominates  $II_3$ . So the game reduces to

$$\begin{array}{c|cc} & II_1 & II_2 \\ \hline I_1 & 4 & 5 \\ I_2 & 4 & 2 \\ I_3 & 2 & 4 \end{array}$$

But now we may iterate this procedure. Since  $I_1$  dominates both  $I_2$  and  $I_3$  in the reduced game, though not in the original game, we have

$$\begin{array}{c|cc} & II_1 & II_2 \\ \hline I_1 & 4 & 5 \end{array}$$

**Lemma 17.1** If a dominated strategy is removed from a game then the solution of the reduced game is a solution of the original game.

Suppose that  $I_2$  dominates  $I_1$  in the original  $n \times m$  game and that  $p^*, q^*$  and  $v$  is a solution to the reduced game when  $I_1$  is omitted.

Then  $e(p^*, q) \geq v$  for all  $q$  and  $e(p, q^*) \leq v$  for all  $p = (0, p_2, \dots, p_n)$ . We need to show that

$$e(p^*, q) \geq v, \quad \text{for all } q, \quad \text{and} \quad e(p, q^*) \leq v, \quad \text{for all } p.$$

Take any  $p = (p_1, p_2, \dots, p_n)$  then

$$\begin{aligned} e(p, q^*) &= \sum_{i=1}^n \sum_{j=1}^m p_i e_{ij} q_j^* = p_1 \sum_{j=1}^m e_{1j} q_j^* + \sum_{i=2}^n \sum_{j=1}^m p_i e_{ij} q_j^* \\ &\leq p_1 \sum_{j=1}^m e_{2j} q_j^* + \sum_{i=2}^n \sum_{j=1}^m p_i e_{ij} q_j^* = e(p', q^*) \end{aligned}$$

where  $p' = (0, p_1 + p_2, p_3, \dots, p_n)$  is a strategy in the reduced game. Thus

$$e(p, q^*) \leq e(p', q^*) \leq v, \quad \text{for all } \mathbf{p} = (p_1, p_2, \dots, p_n).$$

Similarly, we can show  $e(p^*, q) \geq v$  for all  $q$ . ■

# 18 Solution of Two-person Games

## 18.1 Equilibrium pairs

A pair of strategies  $p^*$  and  $q^*$  is an **equilibrium pair** if for any  $p$  and  $q$

$$e(p, q^*) \leq e(p^*, q^*) \leq e(p^*, q).$$

It is possible for there to be more than one equilibrium pair. In the game

$$\begin{array}{ccc} & \text{II}_1 & \text{II}_2 & \text{II}_3 \\ \text{I}_1 & \left( \begin{array}{ccc} 1 & 4 & 1 \\ 0 & 2 & -1 \end{array} \right) \\ \text{I}_2 & & & \end{array}$$

both  $(\text{I}_1, \text{II}_1)$  and  $(\text{I}_1, \text{II}_3)$  are equilibrium pairs. Indeed,  $p^* = (1, 0)$  and  $q^* = (q, 0, 1 - q)$  are equilibrium pairs for any  $0 \leq q \leq 1$ . But in all such cases  $e(p^*, q^*) = 1$ .

**Lemma 18.1** *If  $(p, q)$  and  $(p', q')$  are both equilibrium pairs then  $e(p, q) = e(p', q')$ .*

**Proof.** Since  $(p, q)$  and  $(p', q')$  are both equilibrium pairs, we have

$$e(p', q) \leq e(p, q) \leq e(p, q') \quad \text{and} \quad e(p, q') \leq e(p', q') \leq e(p', q).$$

Together, these imply  $e(p, q) = e(p', q')$ . ■

**Theorem 18.1** *A pair of strategies  $(p^*, q^*)$  in a two-person zero-sum game is an equilibrium pair if and only if  $(p^*, q^*, e(p^*, q^*))$  is a solution to the game.*

**Proof.** If  $(p^*, q^*)$  is an equilibrium pair then

$$\max_p e(p, q^*) \leq e(p^*, q^*) \leq \min_q e(p^*, q).$$

Then

$$\begin{aligned} v_U^M &= \min_q \max_p e(p, q) \leq \max_p e(p, q^*) \leq e(p^*, q^*) \\ &\leq \min_q e(p^*, q) \leq \max_p \min_q e(p, q) = v_L^M. \end{aligned}$$

So, since  $v_L^M \leq v_U^M$  we must have  $v_L^M = v_U^M = e(p^*, q^*)$  so that  $(p^*, q^*, e(p^*, q^*))$  is a solution of the game.

Conversely, if  $(p^*, q^*, e(p^*, q^*))$  is a solution of the game then

$$\begin{aligned} e(p, q^*) &\leq \max_p e(p, q^*) = \min_q \max_p e(p, q) = e(p^*, q^*) \\ &= \max_p \min_q e(p, q) = \min_q e(p^*, q) \leq e(p^*, q). \end{aligned}$$

Hence,  $(p^*, q^*)$  is an equilibrium pair. ■

## 18.2 Solving two-person zero-sum games

We want to find probability vectors  $p^*$ ,  $q^*$  and a  $v$  such that

$$e(p, q^*) \leq e(p^*, q^*) = v \leq e(p^*, q)$$

for any  $p$  and  $q$ . The first inequality implies that

$$e(\text{I}_i, q^*) = \sum_{j=1}^m e_{ij} q_j^* \leq v, \quad \text{for all } i = 1, 2, \dots, n.$$

Player II chooses  $q^*$  so as to make  $v$  as small as possible. So his problem is

$$\text{minimize } \left\{ v : \sum_{j=1}^m e_{ij} q_j \leq v, \quad q_j \geq 0, \quad \sum_{j=1}^m q_j = 1 \right\}.$$

Let,  $Q_j = q_j/v$  then  $Q_1 + Q_2 + \dots + Q_m = \sum_{j=1}^m q_j/v = 1/v$ . Thus, assuming  $v > 0$ , minimizing  $v$  is equivalent to maximizing  $1/v$  so that the final problem is

$$\begin{aligned} &\text{maximize } Q_1 + Q_2 + \dots + Q_m \\ &\text{subject to } \sum_{j=1}^m e_{ij} Q_j \leq 1, \quad i = 1, \dots, n, \quad \text{and } Q_j \geq 0, \quad j = 1, \dots, m. \end{aligned}$$

To ensure  $v > 0$  we can add a constant to every payoff. This will not change the optimal strategy only the value. Consider the dual problem given by

$$\begin{aligned} &\text{minimize } P_1 + P_2 + \dots + P_n \\ &\text{subject to } \sum_{i=1}^n P_i e_{ij} \geq 1, \quad j = 1, 2, \dots, m, \quad \text{and } P_i \geq 0, \quad i = 1, 2, \dots, n. \end{aligned}$$

Interpret this as  $P_i = p_i/v$  for  $i = 1, 2, \dots, n$  and rewrite as

$$\text{maximize } \left\{ v : \sum_{i=1}^n p_i e_{ij} \geq v, \quad p_i \geq 0, \quad \sum_{i=1}^n p_i = 1 \right\}.$$

Thus we can solve the game by solving the primal (or dual) LP. Solving the primal gives the value of the game and player II's optimal strategy  $q^*$  while the dual problem gives player I's optimal strategy  $p^*$ .

### 18.3 Two-person non-zero-sum games

The players are not completely antagonistic to one another and might both be happier with one outcome than another. For non-zero-sum games, (a) a maximin pair is not necessarily an equilibrium pair and vice versa; (b) equilibrium pairs don't necessarily have the same payoffs; (c) there is no obvious solution concept for the game.

Write  $e_1(\cdot, \cdot)$  for player I's payoffs and  $e_2(\cdot, \cdot)$  for player II's payoffs. A pair of strategies  $(p^*, q^*)$  is an **equilibrium pair** if for any  $p$  and  $q$

$$e_1(p, q^*) \leq e_1(p^*, q^*); \quad e_2(p^*, q) \leq e_2(p^*, q^*).$$

#### Example: Prisoner's Dilemma

	Don't confess	Confess
Don't confess	( 5, 5)	(0, 10)
Confess	(10, 0)	( 1, 1)

The equilibrium pair is (Confess, Confess) with payoffs (1, 1). However this is worse for both players than (5, 5), where both players don't confess. The 'confess' strategy dominates the 'don't confess' strategy yet 'most people' would regard (Don't confess, Don't confess) as the 'best' solution.

#### Example: Coordination game

	Opera	Football
Opera	(1, 4)	(0, 0)
Football	(0, 0)	(4, 1)

There are three equilibrium pairs: (Opera, Opera) with payoff (1, 4), (Football, Football) with payoff (4, 1) and a third one consisting of the mixed strategies (1/5, 4/5) and (4/5, 1/5) which gives payoffs of (4/5, 4/5). The difficulty is to persuade the other player to do the same as you. Compare this with flipping a coin and both going to the opera or both to football and sharing the payoffs evenly; this requires **cooperation**.

**Theorem 18.2 (Nash' Theorem)** *Any two-person game (zero-sum or non-zero-sum) with a finite number of pure strategies has at least one equilibrium pair.*

**Proof.** Set  $P = \{p : p_i \geq 0, \sum_i p_i = 1\}$ ,  $Q = \{q : q_j \geq 0, \sum_j q_j = 1\}$  and define

$$S = \{(p, q) : p \in P, q \in Q\}.$$

Then  $S$  is closed, bounded and convex. For any  $p \in P$  and  $q \in Q$  define

$$c_i(p, q) = \max\{0, e_1(I_i, q) - e_1(p, q)\}, \quad i = 1, 2, \dots, n$$

as the amount I gets extra by playing  $I_i$  rather than  $p$  against  $q$ . Let

$$d_j(p, q) = \max\{0, e_2(p, II_j) - e_2(p, q)\}, \quad j = 1, 2, \dots, m$$

be the amount II gets extra by playing  $II_j$  rather than  $q$  against  $p$ .

Define the function  $f(p, q) = (p', q')$ , where

$$p'_i = \frac{p_i + c_i(p, q)}{1 + \sum_{i'} c_{i'}(p, q)} \quad \text{and} \quad q'_j = \frac{q_j + d_j(p, q)}{1 + \sum_{j'} d_{j'}(p, q)}$$

for all  $i = 1, 2, \dots, n, j = 1, 2, \dots, m$ . Then  $f$  is continuous so, by the Brouwer Fixed Point theorem, there is a fixed point  $(p^*, q^*)$  such that

$$f(p^*, q^*) = (p^*, q^*).$$

Observe that we cannot have  $e_1(I_i, q^*) > e_1(p^*, q^*)$  for all  $i = 1, 2, \dots, n$  since that would imply

$$e_1(p^*, q^*) = \sum_{i=1}^n p_i^* e_1(I_i, q^*) > \sum_{i=1}^n p_i^* e_1(p^*, q^*) = e_1(p^*, q^*).$$

Thus for some  $i$  we have that

$$c_i(p^*, q^*) = 0.$$

But since  $(p^*, q^*)$  is a fixed point of  $f$  we have

$$p_i^* = \frac{p_i^* + c_i(p^*, q^*)}{1 + \sum_{i'} c_{i'}(p^*, q^*)}$$

and for the choice of  $i$  with  $c_i(p^*, q^*) = 0$  we see that

$$\sum_{i'=1}^n c_{i'}(p^*, q^*) = 0.$$

Thus, for all  $i' = 1, 2, \dots, n$  we have that  $c_{i'}(p^*, q^*) = 0$  and so

$$e_1(p^*, q^*) \geq e_1(I_i, q^*), \quad i = 1, 2, \dots, n,$$

and hence

$$e_1(p^*, q^*) \geq e_1(p, q^*), \quad \text{for all } p \in P.$$

A similar argument shows that  $e_2(p^*, q^*) \geq e_2(p^*, q)$  for all  $q \in Q$  and so the fixed point  $(p^*, q^*)$  is an equilibrium pair. ■

# 19 Cooperative Games

## 19.1 Finding equilibrium pairs

**Example.** Consider

$$\begin{array}{cc} & \text{II}_1 & \text{II}_2 \\ \text{I}_1 & \left( \begin{array}{cc} (3, 2) & (2, 1) \end{array} \right) \\ \text{I}_2 & \left( \begin{array}{cc} (0, 3) & (4, 4) \end{array} \right) \end{array}$$

Suppose we fix  $q$  and find  $p$  to maximize  $e_1(p, q)$ . If  $q$  is part of an equilibrium pair we have constructed its partner strategy for player I. Then

$$e_1(p, q) = p_1(5q_1 - 2) + 4 - 4q_1.$$

where  $p = (p_1, p_2)$  and  $q = (q_1, q_2)$ . Thus the maximizing  $p_1$  is given by

$$p_1 = \begin{cases} 0 & \text{if } q_1 < 2/5 \\ \text{any } 0 \leq p_1 \leq 1 & \text{if } q_1 = 2/5 \\ 1 & \text{if } q_1 > 2/5. \end{cases}$$

Similarly, for player II, we consider  $e_2(p, q) = q_1(2p_1 - 1) + (4 - 3p_1)$ . and find the maximizing  $q_1$  as

$$q_1 = \begin{cases} 0 & \text{if } p_1 < 1/2 \\ \text{any } 0 \leq q_1 \leq 1 & \text{if } p_1 = 1/2 \\ 1 & \text{if } p_1 > 1/2. \end{cases}$$

Thus we look for the mutual solutions to these two simultaneous maximization problems to find the three equilibrium pairs:

1.  $p_1 = 0, q_1 = 0$ , corresponding to  $(\text{I}_2, \text{II}_2)$  with payoffs  $(4, 4)$ ;
2.  $p_1 = 1, q_1 = 1$ , corresponding to  $(\text{I}_1, \text{II}_1)$  with payoffs  $(3, 2)$ ;
3.  $p_1 = 1/2, q_1 = 2/5$ , corresponding to  $((1/2, 1/2), (2/5, 3/5))$  with payoffs  $(2.4, 2.5)$ .

Notice that the payoffs differ, but that given an equilibrium pair neither player has any incentive to alter his strategy.

## 19.2 Cooperative games

Consider the game

$$\begin{array}{cc} & \text{II}_1 & \text{II}_2 \\ \text{I}_1 & \left( \begin{array}{cc} (0, 0) & (1, 1) \end{array} \right) \\ \text{I}_2 & \left( \begin{array}{cc} (3, 1) & (1, 3) \end{array} \right) \end{array}$$

For cooperative games, the payoff region is larger than in the corresponding non-cooperative game. If  $(u_1, v_1)$  and  $(u_2, v_2)$  are payoffs in the non-cooperative game, then in the cooperative game the strategy which chooses strategies giving payoffs  $(u_1, v_1)$  and  $(u_2, v_2)$  with probabilities  $\beta$  and  $(1 - \beta)$  has expected payoff

$$\beta(u_1, v_1) + (1 - \beta)(u_2, v_2).$$

Thus the payoff region,  $R$ , for the cooperative game is the convex hull of the payoff region for the non-cooperative game, i.e., it is the smallest convex region that covers the payoff region for the non-cooperative game. In fact,  $R$  is most simply obtained by constructing the convex hull of the points corresponding to each player adopting one of his pure strategies. Points corresponding to the use of pure strategies may be in the interior of the convex hull; e.g., the pair of strategies  $(\text{I}_1, \text{II}_2)$  in the game above.

Notice the different way that randomization is used. In a non-cooperative game each player randomizes over his choice of strategy. This is done to confuse the opponent. In a cooperative game players jointly randomize over pairs of strategies, which may themselves be mixed strategies for the game. Randomization is used to average between possible outcomes.

## 19.3 Bargaining

In a cooperative setting, there is a preplay stage or negotiating stage where the players decide on the strategies to be used. Of the possible payoffs, which are the players likely to agree on? We say that a pair of payoffs  $(u, v)$  in a cooperative game is **jointly dominated** by  $(u', v')$  if

$$u' \geq u, \quad v' \geq v \quad \text{and} \quad (u', v') \neq (u, v).$$

A pair of payoffs  $(u, v)$  is said to be **Pareto optimal** if it is not jointly dominated. Certainly, the players will only be interested in agreeing on Pareto optimal payoffs since otherwise there is a payoff such that both can do as well and one can do better.

Players I and II can always guarantee themselves payoffs of at least

$$v_I = \max_{p \in P} \min_{q \in Q} e_1(p, q) \quad \text{and} \quad v_{II} = \max_{q \in Q} \min_{p \in P} e_2(p, q).$$

respectively. So, we would expect the solution of a cooperative game to lie within the **bargaining set** (also called the negotiation set) given by

$$B = \{(u, v) : u \geq v_I, v \geq v_{II}, \text{ and } (u, v) \text{ is Pareto optimal in } R\}.$$

How ought the players agree which point of  $B$  is to be used?

Note that we must not make inter-player comparisons of payoffs, which we are supposing are measured by their own utilities, not necessarily on the same scale. We should not conclude that I prefers (4, 1) to (1, 10) by less than II prefers (1, 10) to (4, 1).

### Status quo point

Nash suggested that within the set of feasible payoffs,  $R$ , jointly available to the players there is also a special payoff,  $(u_0, v_0) \in R$  called the status quo point, which is the outcome the players will receive if they can't agree by a process of negotiation.

An arbitration procedure,  $\Psi$ , is defined as a map from the status quo point and the region  $R$  to some point  $(u^*, v^*) \in R$

$$\Psi((u_0, v_0), R) = (u^*, v^*).$$

## 19.4 Nash bargaining axioms

1. **feasibility.**  $(u^*, v^*) \in R$ .
2. **at least as good as status quo**  $u^* \geq u_0, v^* \geq v_0$ .
3. **Pareto optimality.** If  $(u, v) \in R$  and  $u \geq u^*, v \geq v^*$  then  $u = u^*, v = v^*$ .
4. **symmetry.** If  $R$  is symmetric, so that if  $(u, v) \in R \implies (v, u) \in R$ , and if  $u_0 = v_0$ , then  $u^* = v^*$ .
5. **invariance under linear transformations.** Let  $R'$  be obtained from  $R$  by the transformation

$$u' = au + b, v' = cv + d, \quad a, c > 0.$$

Then if  $(u^*, v^*) = \Psi((u_0, v_0), R)$  then  $\Psi((au_0 + b, cv_0 + d), R') = (au^* + b, cv^* + d)$ .

6. **independence of irrelevant alternatives.** If  $R'$  is a subset of  $R$ ,  $\Psi((u_0, v_0), R) = (u^*, v^*)$  and  $(u^*, v^*) \in R'$ , then we must also have  $\Psi((u_0, v_0), R') = (u^*, v^*)$ .

## 19.5 Nash's arbitration procedure

For  $(u, v) \in R$  with  $u > u_0, v > v_0$  define the function

$$f(u, v) = (u - u_0)(v - v_0).$$

If there exists points  $(u, v) \in R$  with  $u > u_0, v > v_0$  then  $f$  attains a unique maximum at some point  $(u^*, v^*) \in R$ . Define

$$\Psi((u_0, v_0), R) = (u^*, v^*).$$

Nash showed that  $\Psi$  is the only function that satisfies the axioms (1–6).

## 19.6 Maximin bargaining solutions

Nash's result specifies the arbitration procedure  $\Psi$  for a given status quo point  $(u_0, v_0)$ . A natural choice is to take the maximin values  $(v_I, v_{II})$  as the status quo point. This gives the **Maximin bargaining solution**.

**Example.** Consider the two-person non-zero sum game with payoffs

$$\begin{array}{cc} & \begin{array}{c} \text{II}_1 \\ \text{II}_2 \end{array} \\ \begin{array}{c} \text{I}_1 \\ \text{I}_2 \end{array} & \left( \begin{array}{cc} (1,2) & (8,3) \\ (4,4) & (2,1) \end{array} \right) \end{array}$$

Consider the two zero-sum games for each player separately. Using the LP approach we find the maximin values of  $v_I = 3\frac{1}{3}$  and  $v_{II} = 2\frac{1}{2}$ .

The negotiation set of Pareto optimal points is given by

$$B = \{(u, v) : u + 4v = 20, 4 \leq u \leq 8\}.$$

Thus we wish to maximize over  $B$

$$\begin{aligned} f(u, v) &= (u - u_0)(v - v_0) = \left(u - 3\frac{1}{3}\right) \left(v - 2\frac{1}{2}\right) \\ &= \left(u - 3\frac{1}{3}\right) \left(5 - \frac{1}{4}u - 2\frac{1}{2}\right). \end{aligned}$$

This gives  $(u^*, v^*) = (6\frac{2}{3}, 3\frac{1}{3})$  as the unique solution to the Nash arbitration procedure for the maximin bargaining solution.

## 20 Coalitional Games

### 20.1 Characteristic function

We can extend our definition of (Nash) equilibrium pairs to  $n$ -person games. The  $n$ -tuple of strategies  $p_1^*, p_2^*, \dots, p_n^*$ , where player  $i$  plays mixed strategy  $p_i^*$  is an equilibrium  $n$ -tuple if for all other strategies,  $p_1, p_2, \dots, p_n$ ,

$$e_i(p_1^*, p_2^*, \dots, p_i^*, \dots, p_n^*) \geq e_i(p_1^*, p_2^*, \dots, p_i, \dots, p_n^*), \quad i = 1, 2, \dots, n.$$

If there are  $n > 2$  players in the game then there might be cooperation between some, but not necessarily all, of the players. We can ask which **coalitions** of players are likely to form and what are the relative bargaining strengths of the coalitions that do form.

Label the players  $1, 2, \dots, n$ . A **coalition** of players,  $S$ , is then a subset of  $N = \{1, 2, \dots, n\}$ . The worst eventuality is that the rest of the players unite and form a single opposing coalition  $N \setminus S$ . This is then a 2-person non-cooperative game and we can calculate the maximum payoff that  $S$  can ensure for itself using the maximin criterion.

Let  $v(S)$  denote the maximum value  $v(S)$  that coalition  $S$  can guarantee itself by coordinating the strategies of its members, no matter what the other players do. This is called the **characteristic function**. By convention, we take  $v(\emptyset) = 0$ . The characteristic function measures the strengths of possible coalitions. Note that for any two disjoint sets  $S$  and  $T$ , we have **superadditivity**, i.e.,

$$v(S \cup T) \geq v(S) + v(T).$$

### 20.2 Imputations

Given that a coalition forms, how should  $v(S)$  be shared between its members? The distribution of individual rewards will affect whether any coalition is likely to form. Each individual will tend to join the coalition that offers him the greatest reward.

An **imputation** in a  $n$ -person game with characteristic function  $v$  is a vector  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  satisfying

$$(1) \sum_{i=1}^n x_i = v(N), \quad \text{and} \quad (2) x_i \geq v(\{i\}) \text{ for each } i = 1, 2, \dots, n.$$

Think of  $x_i$  as player  $i$ 's reward. Let  $E(v)$  be the set of imputations.

Suppose that  $x$  and  $y$  are imputations. Then we know that

$$\sum_{i=1}^n x_i = v(N) = \sum_{i=1}^n y_i,$$

and so if  $x_i > y_i$  then there must be some  $j$  with  $y_j > x_j$ . So everyone cannot be better off under  $x$  than under  $y$ . However, it is possible that all members of a particular coalition are better off under  $x$  than under  $y$ .

Let  $x, y \in E(v)$ . We say that  $y$  **dominates**  $x$  over  $S$  if

$$(3) y_i > x_i \text{ for all } i \in S; \quad \text{and} \quad (4) \sum_{i \in S} y_i \leq v(S).$$

(4) ensures  $v(S)$  is large enough to pay its members the amounts in  $y$ .

### 20.3 The core

The **core** of a game with characteristic function  $v$  is the set,  $C(v)$ , of all imputations that are not dominated for any coalition. The idea is that only such imputations can persist in pre-game negotiations.

**Theorem 20.1**  *$x$  is in the core if and only if*

$$(5) \sum_{i=1}^n x_i = v(N); \quad (6) \sum_{i \in S} x_i \geq v(S) \text{ for all } S \subset N.$$

**Proof.** Let  $x$  satisfy (5) and (6). Putting  $S = \{i\}$  for each  $i = 1, \dots, n$  shows that  $x$  is an imputation. To show that it is not dominated, suppose, there is a coalition  $S$  with  $y_i > x_i$  for all  $i \in S$ . But using (6)

$$\sum_{i \in S} y_i > \sum_{i \in S} x_i \geq v(S),$$

which violates (4).

Conversely, suppose that  $x$  is in the core. It is an imputation and so (5) must hold. Now suppose, if possible, that for some coalition  $S$  condition (6) doesn't hold so that  $\sum_{i \in S} x_i < v(S)$ . Define  $\epsilon$  by

$$\epsilon = \frac{v(S) - \sum_{i \in S} x_i}{|S|} > 0$$



Then the imputation

$$y_i = \begin{cases} x_i + \epsilon & i \in S \\ v(\{i\}) + \frac{v(N) - v(S) - \sum_{i \notin S} v(\{i\})}{|N \setminus S|} & i \notin S \end{cases}$$

dominates  $x$  on  $S$ , contradicting the assumption that  $x$  is in the core. ■

## 20.4 Oil market game

Country 1 has oil which it can use to run its transportation networks at a profit of  $\mathcal{L}a$  per barrel. Country 2 wants to buy oil for use in farming, for a profit of  $\mathcal{L}b$  per barrel. Country 3 wants to buy oil for use in manufacturing, for a profit of  $\mathcal{L}c$  per barrel. Assume  $a < b < c$ . The characteristic function is

Coalition, $S$	Characteristic function, $v(S)$
$\emptyset, \{2\}, \{3\}, \{2, 3\}$	0
$\{1\}$	$a$
$\{1, 2\}$	$b$
$\{1, 3\}, \{1, 2, 3\}$	$c$

Suppose  $\mathbf{x} = (x_1, x_2, x_3)$  is an imputation in the core. We must have

$$x_1 + x_2 + x_3 = v(N) = v(\{1, 2, 3\}) = c;$$

and using  $\sum_i x_{i \in S} \geq v(S)$ ,

$$\begin{aligned} x_1 &\geq a; & x_2 &\geq 0; & x_3 &\geq 0; \\ x_1 + x_2 &\geq b; & x_2 + x_3 &\geq 0; & x_1 + x_3 &\geq c. \end{aligned}$$

Thus  $x_2 = 0$ ,  $x_1 + x_3 = c$  and  $x_1 \geq b$  and so the core is given by

$$C(v) = \{(x, 0, c - x) : b \leq x \leq c\}.$$

The interpretation is that countries 1 and 3 form a coalition, with 1 selling oil to 3 for  $\mathcal{L}x$  per barrel, which is at least  $\mathcal{L}b$  per barrel (otherwise, country 1 would be better off selling to country 2) and at most  $\mathcal{L}c$  per barrel, otherwise, country 3 would pay more than it values the oil.

## 20.5 The nucleolus

A major drawback of the core is that for many games it is empty.

For any imputation  $\mathbf{x} = (x_1, x_2, \dots, x_n) \in E(v)$  and for any coalition  $S \subset N$  define

$$x(S) = \sum_{i \in S} x_i.$$

The quantity  $v(S) - x(S)$  measures the ‘unhappiness’ that coalition  $S$  feels for the imputation  $\mathbf{x}$ . The larger this value the larger the difference between what the coalition could get and what it actually gets.

Define  $\theta(\mathbf{x})$  to be the vector of  $2^n$  values taken by  $v(S) - x(S)$  as  $S$  varies across the possible coalitions arranged in decreasing order. We use  $\theta(\mathbf{x})$  to order imputations  $\mathbf{x}$  and  $\mathbf{y}$ .

Writing  $\theta(\mathbf{x}) = (\theta(\mathbf{x})_1, \theta(\mathbf{x})_2, \dots, \theta(\mathbf{x})_n)$  we say that  $\theta(\mathbf{x}) < \theta(\mathbf{y})$  if  $\theta(\mathbf{x})_1 < \theta(\mathbf{y})_1$  or if  $\theta(\mathbf{x})_k = \theta(\mathbf{y})_k$  for  $k = 1, 2, \dots, i - 1$  and  $\theta(\mathbf{x})_i < \theta(\mathbf{y})_i$ .

The **nucleolus** is the smallest imputation under this ordering, written

$$N(v) = \{\mathbf{x} \in E(v); \theta(\mathbf{x}) < \theta(\mathbf{y}) \text{ for all } \mathbf{y} \in E(v)\}.$$

It can be shown that *the nucleolus always exists and is unique*. Also, *provided the core is non-empty the nucleolus lies within the core*. To see this, let  $\mathbf{x}$  be in the core. Then for any coalition  $S$ ,  $v(S) - x(S)$  is zero or negative (by definition of the core). Thus all the entries of  $\theta(\mathbf{x})$  for an  $\mathbf{x}$  in the core are zero or negative and hence this property will have to hold for the minimal choice of  $\theta(\mathbf{y})$  over choices of imputation  $\mathbf{y}$ . But this means that such a minimizing imputation  $\mathbf{y}$  is in the core.

Thus the nucleolus is a natural interpretation for a fair division of the reward  $v(N)$ . Consider again the oil market game. To construct the nucleolus we need only consider imputations in the core,  $C(v) = \{(x, 0, c - x) : b \leq x \leq c\}$ .

Computing  $v(S) - x(S)$  for all possible coalitions gives

$$\begin{aligned} v(\emptyset) - x(\emptyset) &= 0 & v(\{1, 2\}) - x(\{1, 2\}) &= b - x \\ v(\{1\}) - x(\{1\}) &= a - x & v(\{2, 3\}) - x(\{2, 3\}) &= x - c \\ v(\{2\}) - x(\{2\}) &= 0 & v(\{1, 3\}) - x(\{1, 3\}) &= 0 \\ v(\{3\}) - x(\{3\}) &= x - c & v(\{1, 2, 3\}) - x(\{1, 2, 3\}) &= 0 \end{aligned}$$

The largest nonzero element is either  $b - x$  or  $x - c$ . Thus we minimize the largest nonzero unhappiness by setting  $b - x = x - c$ , i.e.,  $x = (b + c)/2$ . Thus, the nucleolus is the imputation  $\mathbf{x} = ((c + b)/2, 0, (c - b)/2)$  and

$$\theta(\mathbf{x}) = (0, 0, 0, 0, \frac{1}{2}(b - c), \frac{1}{2}(b - c), \frac{1}{2}(b - c), a - \frac{1}{2}(b + c)).$$

## 21 Shapley Value and Market Games

### 21.1 Shapley value

What might each player reasonably expect to receive as his share of the reward in a cooperative game? Shapley proposed three axioms that one might require for  $\phi_i(v)$ , player  $i$ 's expected share in a game with characteristic function  $v$ .

#### Shapley's axioms

1.  $\phi_i(v)$  should be independent of the player's label,  $1, 2, \dots, n$ .
2.  $\sum_{i=1}^n \phi_i(v) = v(N)$ .
3. If  $u$  and  $v$  are two characteristic functions then

$$\phi_i(u + v) = \phi_i(u) + \phi_i(v).$$

**Theorem 21.1 (Shapley)** *The only function that satisfies Shapley's axioms is given by the **Shapley values***

$$\phi_i(v) = \sum_{S:i \in S} \frac{(|S| - 1)!(n - |S|)!}{n!} [v(S) - v(S \setminus \{i\})].$$

The values arise by imagining the players join the game in random order. Player  $i$  receives the extra amount that he brings to the existing coalition of players  $S \setminus \{i\}$ , i.e.,  $v(S) - v(S \setminus \{i\})$ . This must then be averaged over all the possible ways in which the players can arrive.

For the oil market game we have

$$\phi_1(v) = \frac{1}{2}c + \frac{1}{3}a - \frac{1}{6}b, \quad \phi_2(v) = \frac{1}{6}b - \frac{1}{6}a, \quad \phi_3(v) = \frac{1}{2}c - \frac{1}{6}a - \frac{1}{3}b.$$

The Shapley values give another solution concept for the game. However, note that this imputation is not in the core.

### 21.2 Market games

Some of the earliest examples of game theory can be found in the mathematical economics which was developed to understand the bargaining involved in trading. We will consider simple examples suggested by Edgeworth.

Suppose there are just two commodities (A for apples and B for bread, say). Assume there are  $M$  apple traders and  $N$  bread traders. A trader who has  $a$  units of apples and  $b$  units of bread has utility

$$u_i(a, b), \quad i = 1, 2, \dots, M + N.$$

Assume also that the functions are concave so that every trader prefers some combination of the two commodities rather than either of the two extremes  $(a, 0)$  or  $(0, b)$  for some  $a, b$ . Hence,

$$u_i(\lambda(a_1, b_1) + (1 - \lambda)(a_2, b_2)) \geq \lambda u_i(a_1, b_1) + (1 - \lambda)u_i(a_2, b_2)$$

for  $0 \leq \lambda \leq 1$  and  $i = 1, 2, \dots, M + N$ .

Suppose that each trader has the same utility function  $u(x, y)$  and that each trader of type A or B starts with  $a$  and  $b$  units of commodities A and B respectively. Suppose that coalition  $S$  consists of  $s_1$  traders of type A and  $s_2$  traders of type B. The best  $S$  can ensure for itself is the highest possible sum of the utilities of its members that can be obtained when they trade with each other. Thus,

$$v(S) = \max_{x_1, \dots, x_{s_1+s_2}, y_1, \dots, y_{s_1+s_2}} \sum_{i=1}^{s_1+s_2} u(x_i, y_i)$$

where

$$\sum_{i=1}^{s_1+s_2} x_i = s_1 a; \quad \sum_{i=1}^{s_1+s_2} y_i = s_2 b.$$

By concavity of  $u(\cdot, \cdot)$  we have

$$\sum_{i=1}^{s_1+s_2} u(x_i, y_i) \leq (s_1 + s_2) u\left(\frac{s_1}{s_1 + s_2} a, \frac{s_2}{s_1 + s_2} b\right) = v(S).$$

**[1, 1]-market game.** The characteristic function is

$$v(\{1\}) = u(a, 0); \quad v(\{2\}) = u(0, b); \quad v(\{1, 2\}) = 2u\left(\frac{a}{2}, \frac{b}{2}\right).$$

and so the set of imputations is given by

$$E(v) = \{(u(a, 0) + pc, u(0, b) + (1 - p)c) : 0 \leq p \leq 1\}$$

where  $c = 2u(a/2, b/2) - u(a, 0) - u(0, b)$ .

We can think of  $p$  as the price of the goods, reflecting the number of units of B exchanged for one unit of A.

**[1, N]-market game.** Think of trader A as a monopolist. We suspect he can charge as high a price as he wants, provided that it is still worth the others trading. We illustrate this by showing that imputation

$$\mathbf{x}^* = \left( (N+1)u\left(\frac{a}{N+1}, \frac{Nb}{N+1}\right) - Nu(0, b), u(0, b), \dots, u(0, b) \right)$$

is in the core. This means showing that for any set  $K$  of  $k$  type B traders,

$$x_1^* + \sum_{i \in K} x_i^* \geq v(K \cup \{1\}).$$

Using our expression for  $v(S)$  this corresponds to showing that

$$(N+1)u\left(\frac{a}{N+1}, \frac{Nb}{N+1}\right) - (N-k)u(0, b) \geq (k+1)u\left(\frac{a}{k+1}, \frac{kb}{k+1}\right)$$

which follows directly from the concavity of  $u(\cdot, \cdot)$ .

### 21.3 Competition between firms

We now look at games of competition amongst firms. The case of two firms is known as **duopoly**; the case of more than two firms it is known as **oligopoly**. Duopoly can be regarded as a  $[2, \infty]$ -market game and oligopoly can be regarded as a  $[M, \infty]$ -market game. The first type of traders are the firms who produce a particular product. The second type are the buyers, or consumers, who exchange money for the product.

We represent the consumer's requirements by one utility function

$$u(p_1, p_2, \dots, p_M, q_1, q_2, \dots, q_M)$$

where  $p_i$  is firm  $i$ 's price for the product and  $q_i$  is the amount of that firm's product that is bought by consumers.

Let us assume that consumers are told the prices,  $p_i$ , and then choose the quantities,  $q_i$ , so as to maximize the above utility. Hence, this reduces to a set of price-demand equations which connect the demand  $q_i$  for firm  $i$ 's product with the (announced) prices  $p_1, p_2, \dots, p_M$ , so that, say,

$$q_i = f_i(p_1, p_2, \dots, p_M).$$

Firm  $i$ 's utility is given by its profit

$$e_i(p_1, p_2, \dots, p_M) = p_i q_i - c_i(q_i)$$

where  $c_i(\cdot)$  is the production cost function for firm  $i$ .

A similar story can be told if we suppose that firms decide on the quantities  $q_i$  that they will produce and then the consumers' utility function determines the prices  $p_i$  they will pay for these products.

### 21.4 Cournot equilibrium

A **Cournot equilibrium** is a vector of prices  $(p_1^c, p_2^c, \dots, p_M^c)$  such that

$$e_i(p_1^c, p_2^c, \dots, p_M^c) = \max_{p_i} e_i(p_1^c, p_2^c, \dots, p_i, \dots, p_M^c)$$

for all firms  $i = 1, \dots, M$ . That is,  $p^c$  is an equilibrium  $n$ -tuple in a  $n$ -person non-cooperative game of price competition. Notice that we cannot apply Nash's theorem since there are an infinite number of possible choices of prices, i.e., of pure strategies. Nevertheless, it can be shown that under reasonable assumptions a Cournot equilibrium always exists.

**Example.** Consider a duopoly where the price-demand functions are

$$q_1 = f_1(p_1, p_2) = \max\left\{1 + \frac{1}{3}p_2 - \frac{1}{2}p_1, 0\right\}$$

$$q_2 = f_2(p_1, p_2) = \max\left\{1 + \frac{1}{4}p_1 - \frac{1}{2}p_2, 0\right\}$$

and suppose, for simplicity, that  $c_1(q_1) = c_2(q_2) = 0$ .

We have that  $0 \leq p_1 \leq 2 + \frac{2}{3}p_2$  and  $0 \leq p_2 \leq 2 + \frac{1}{2}p_1$ . The profit functions are then given by

$$e_1(p_1, p_2) = p_1 + \frac{1}{3}p_1 p_2 - \frac{1}{2}p_1^2$$

$$e_2(p_1, p_2) = p_2 + \frac{1}{4}p_1 p_2 - \frac{1}{2}p_2^2.$$

To find the Cournot equilibrium we must solve

$$de_1(p_1, p_2^c)/dp_1 = de_2(p_1^c, p_2)/dp_2 = 0.$$

This gives equations for  $(p_1^c, p_2^c)$  of

$$\frac{de_1(p_1, p_2^c)}{dp_1} = 1 + \frac{1}{3}p_2^c - p_1^c = 0, \quad \frac{de_2(p_1^c, p_2)}{dp_2} = 1 + \frac{1}{4}p_1^c - p_2^c = 0,$$

which gives the Cournot equilibrium as  $p_1^c = \frac{16}{11}$ ,  $p_2^c = \frac{15}{11}$ , and so

$$e_1\left(\frac{16}{11}, \frac{15}{11}\right) = 1.06; \quad e_2\left(\frac{16}{11}, \frac{15}{11}\right) = 0.93.$$

The maximization conditions  $de_i(p_1, p_2)/dp_i = 0$  express the price firm 1 will choose given firm 2's price and vice versa. Thus,

$$p_1 = g_1(p_2) = 1 + \frac{1}{3}p_2, \quad p_2 = g_2(p_1) = 1 + \frac{1}{4}p_1.$$

Suppose firm 1 must announce its price before firm 2. Firm 2 will choose its price to maximize its profit given  $p_1$ . Thus, it will choose  $p_2 = g_2(p_1)$ . Firm 1, realizing this will happen, will maximize its profit by choosing  $p_1$  to maximize  $e_1(p_1, g_2(p_1))$ . Firm 1 then announces this price.

## 22 Evolutionary Games

### 22.1 Stackleberg leadership

**Stackleberg** introduced the type of strategy introduced in Section 21.4 the firm who announces their price first is called the **leader** and the one that reacts is called the **follower**.

Continuing our earlier numerical example, if firm 1 is the leader he should choose  $p_1$  to maximize

$$e_1(p_1, g_2(p_1)) = p_1 + \frac{1}{3}p_1(1 + \frac{1}{4}p_1) - \frac{1}{2}p_1^2$$

which gives  $p_1 = \frac{8}{5}$ . So, firm 2 will choose a price,  $p_2 = g_2(p_1) = \frac{7}{5}$ . The corresponding profits are

$$e_1(\frac{8}{5}, \frac{7}{5}) = 1.067; \quad e_2(\frac{8}{5}, \frac{7}{5}) = 0.98.$$

If firm 2 is the leader then  $p_1 = p_2 = \frac{3}{2}$  and the profits are

$$e_1(\frac{3}{2}, \frac{3}{2}) = 1.125; \quad e_2(\frac{3}{2}, \frac{3}{2}) = 0.9375.$$

What if they both try to be leader? They will choose prices  $p_1 = \frac{8}{5}$  and  $p_2 = \frac{3}{2}$  with profits

$$e_1(\frac{8}{5}, \frac{3}{2}) = 1.12; \quad e_2(\frac{8}{5}, \frac{3}{2}) = 0.975.$$

Thus, a firm does better than the Cournot equilibrium when it announces its price, but does even better when the competing firm also announces its price.

A similar analysis can be made when firms 1 and 2 state the quantities they plan to produce and then the prices adjust so that these quantities are sold. (The names Cournot and Bertrand are sometimes used to distinguish between models of quantity competition and price competition, respectively. In fact, Cournot was the first to study both types of model, so we also associate his name with the price competition case.)

### 22.2 Joint maximization of profits

If we think of duopoly as a cooperative game then we are led to consider the joint maximization of profits. The two firms choose their prices  $(p_1, p_2)$  to maximize

$$e_1(p_1, p_2) + e_2(p_1, p_2).$$

This has the same value, when maximized, as the characteristic function value  $v(\{1, 2\})$  for the game when considered as an  $n$ -person game. Such a price vector is Pareto optimal but there are many other such Pareto optimal price vectors.

**Example.** Continuing the numerical example, we wish to maximize

$$e_1(p_1, p_2) + e_2(p_1, p_2) = p_1 + \frac{1}{3}p_1p_2 - \frac{1}{2}p_1^2 + p_2 + \frac{1}{4}p_1p_2 - \frac{1}{2}p_2^2.$$

Setting derivatives to zero gives

$$1 + \frac{7}{12}p_2 - p_1 = 0; \quad 1 + \frac{7}{12}p_1 - p_2 = 0$$

which gives  $p_1 = p_2 = \frac{12}{5}$ . The profits are then

$$e_1(\frac{12}{5}, \frac{12}{5}) = 1.44; \quad e_2(\frac{12}{5}, \frac{12}{5}) = 0.96.$$

The maximin values for the prices are given by finding

$$\max_{p_1} \min_{p_2} e_1(p_1, p_2) \quad \text{and} \quad \max_{p_2} \min_{p_1} e_2(p_1, p_2).$$

Thus, using the expressions for  $e_1(p_1, p_2)$  and  $e_2(p_1, p_2)$ , we find that firm 1's maximin value ensures a profit of 0.5, where  $p_1 = 1$  and  $p_2 = 0$ . Firm 2's maximin value ensures a profit of 0.5, where  $p_1 = 0$  and  $p_2 = 1$ .

To find the negotiation set for the Nash bargaining game we must find all Pareto optimal prices. That is, price vectors  $(p_1^*, p_2^*)$  such that there are no other  $(p_1, p_2)$  with

$$e_i(p_1, p_2) \geq e_i(p_1^*, p_2^*), \quad i = 1, 2$$

and with strict inequality in at least one component.

We may find such price vectors by solving for all values of  $c$  the problem

$$\text{maximize } e_1(p_1, p_2)$$

subject to

$$e_2(p_1, p_2) \geq c, \quad p_1, p_2 \text{ feasible}.$$

We could apply Lagrangian methods to solve this problem.

### 22.3 Evolutionary games

Suppose that some individual uses a (behavioural) strategy,  $\mathbf{x}$  say, from some set of possible strategies and that on meeting another individual who uses strategy  $\mathbf{y}$ , there is a payoff  $e(\mathbf{x}, \mathbf{y})$  to the first individual. The payoff is a measure of fitness describing the individual's likely reproductive success.

No 'rational thought' is involved in selecting a strategy. Instead, the individual whose genes make it use strategy  $\mathbf{x}$  will have offspring with the same genes. If payoffs resulting from strategy  $\mathbf{x}$ , namely  $e(\mathbf{x}, \cdot)$ , are high then it will produce more offspring using strategy  $\mathbf{x}$  than a less successful gene type and will predominate in the population.

## 22.4 Evolutionary stable strategies

Suppose that changes to a strategy arise through mutation. As usual, we look for equilibrium points, by which we mean strategies whose fitness is greater than that of any mutant strategy that could arise. If this is so then the next generation will have a smaller proportion of the mutant strategy and will eventually die out.

**Definition 22.1** Let  $X$  be the set of strategies. A strategy  $\mathbf{x}^* \in X$  is an **evolutionary stable strategy (ESS)** if for every  $\mathbf{y} \in X$ ,  $\mathbf{y} \neq \mathbf{x}^*$  then

$$e(\mathbf{x}^*, \bar{\mathbf{x}}) > e(\mathbf{y}, \bar{\mathbf{x}}) \quad (22.1)$$

where  $\bar{\mathbf{x}} = (1 - \epsilon)\mathbf{x}^* + \epsilon\mathbf{y}$  for sufficiently small  $\epsilon > 0$ .

Consider  $\mathbf{y}$  as the mutant strategy which affects a proportion  $\epsilon$  of the population.

Notice that as  $\epsilon \rightarrow 0$ ,  $\bar{\mathbf{x}} \rightarrow \mathbf{x}^*$  so for (\*) to hold for small  $\epsilon$  we must have that

$$e(\mathbf{x}^*, \mathbf{x}^*) \geq e(\mathbf{y}, \mathbf{x}^*). \quad (22.2)$$

If the inequality is strict then (22.1) certainly holds for small enough  $\epsilon$ . However, if

$$e(\mathbf{x}^*, \mathbf{x}^*) = e(\mathbf{y}, \mathbf{x}^*)$$

then we see from expanding (22.1) that we need

$$(1 - \epsilon)e(\mathbf{x}^*, \mathbf{x}^*) + \epsilon e(\mathbf{x}^*, \mathbf{y}) > (1 - \epsilon)e(\mathbf{y}, \mathbf{x}^*) + \epsilon e(\mathbf{y}, \mathbf{y}).$$

and so that if  $e(\mathbf{x}^*, \mathbf{x}^*) = e(\mathbf{y}, \mathbf{x}^*)$  we must have that

$$e(\mathbf{x}^*, \mathbf{y}) > e(\mathbf{y}, \mathbf{y}). \quad (22.3)$$

Hence  $\mathbf{x}^* \in X$  is an ESS if and only if for every  $\mathbf{y} \in X$ ,  $\mathbf{y} \neq \mathbf{x}^*$  (22.2) holds and if  $e(\mathbf{x}^*, \mathbf{x}^*) = e(\mathbf{y}, \mathbf{x}^*)$  then (22.3) holds.

We can think of an evolutionary game as a two-person non-zero-sum game in which the payoffs are  $e_1(\mathbf{x}, \mathbf{y}) = e(\mathbf{x}, \mathbf{y})$  and  $e_2(\mathbf{x}, \mathbf{y}) = e(\mathbf{y}, \mathbf{x})$ . So, by the above, if  $\mathbf{x}^*$  is an ESS then  $(\mathbf{x}^*, \mathbf{x}^*)$  is an equilibrium pair for the corresponding game. However, not every equilibrium pair are ESSs.

**Example.** Consider the two pure strategies: ‘Hawk’ vs ‘Dove’. ‘Hawk’ means keep fighting until you or your opponent is injured and ‘Dove’ means run away. Suppose the winner earns  $V$  and injury means fitness decreases by  $D$ . Thus, the payoff matrix is of the following form

$$\begin{array}{cc} & \begin{array}{cc} \text{Hawk} & \text{Dove} \end{array} \\ \begin{array}{c} \text{Hawk} \\ \text{Dove} \end{array} & \left( \begin{array}{cc} \frac{1}{2}(V-D) & V \\ 0 & \frac{1}{2}V \end{array} \right) \end{array}$$

**Case  $V \geq D$ .** For example,  $V = 4$ ,  $D = 2$ . So we get payoffs

$$\begin{array}{cc} & \begin{array}{cc} \text{Hawk} & \text{Dove} \end{array} \\ \begin{array}{c} \text{Hawk} \\ \text{Dove} \end{array} & \left( \begin{array}{cc} 1 & 4 \\ 0 & 2 \end{array} \right) \end{array}$$

But then the pure Hawk strategy  $\mathbf{x}^* = (1, 0)$  is an ESS since for any  $\mathbf{y} = (y, 1 - y)$ ,  $y \neq 1$

$$e(\mathbf{x}^*, \mathbf{x}^*) = 1 > y = e(\mathbf{y}, \mathbf{x}^*).$$

**Case  $V < D$ .** For example, put  $V = 2$ ,  $D = 4$ . Now injury outweighs the advantage of winning and the payoffs are given by

$$\begin{array}{cc} & \begin{array}{cc} \text{Hawk} & \text{Dove} \end{array} \\ \begin{array}{c} \text{Hawk} \\ \text{Dove} \end{array} & \left( \begin{array}{cc} -1 & 2 \\ 0 & 1 \end{array} \right) \end{array}$$

For this game  $\mathbf{x}^* = (\frac{1}{2}, \frac{1}{2})$  is an ESS. Note that  $\mathbf{y} = (y, 1 - y)$ ,  $y \neq \frac{1}{2}$  then

$$e(\mathbf{x}^*, \mathbf{x}^*) = \frac{1}{2} = e(\mathbf{y}, \mathbf{x}^*).$$

So we must check that  $e(\mathbf{x}^*, \mathbf{y}) > e(\mathbf{y}, \mathbf{y})$ . This follows as

$$e(\mathbf{y}, \mathbf{y}) = 1 - 2y^2; \quad e(\mathbf{x}^*, \mathbf{y}) = \frac{3}{2} - 2y.$$

$$e(\mathbf{x}^*, \mathbf{y}) - e(\mathbf{y}, \mathbf{y}) = \left(\frac{3}{2} - 2y\right) - (1 - 2y^2) = 2\left(y - \frac{1}{2}\right)^2 > 0.$$

More generally for  $V, D \geq 0$  we find the following

1.  $V \geq D$ . The Hawk strategy  $\mathbf{x}^* = (1, 0)$  is always an ESS. (For  $V = D$ , (22.2) holds, so we must check (22.3)).
2.  $V < D$ . The mixed strategy  $\mathbf{x}^* = (V/D, (D - V)/D)$  is an ESS.

Notice that as  $D$  increases, the ESS strategy tends to the ‘Dove’ strategy. This is why snakes wrestle each other rather than bite!

## 23 Auctions

### 23.1 Types of auctions

Auctions are bidding mechanisms. They are described by a set of rules that specify the way bidding occurs, what information the bidders have about the state of bidding, how the winner is determined and how much he must pay. They can be viewed as partial information games in which a bidder's valuation of an item is hidden from the auctioneer and other bidders. The game's equilibrium is a function of the auction's rules. These rules can affect the revenue obtained by the seller, as well as how much this varies in successive instants of the auction. An auction is economically efficient, in terms of maximizing social welfare, if it allocates items to bidders who value them most. The design of an auction for a particular situation is an art. There is no single auctioning mechanism that is provably efficient and can be applied in most situations.

Government contracts are often awarded through procurement auctions. Flowers, wines, art, U.S. treasury bonds and real estate are sold in auctions (and indeed the Roman empire was auctioned by the Praetorian Guards in A.D. 193). They are used to sell oil drilling rights, or other natural resources, such as mobile telephone spectrum or satellite positions. Takeover battles for companies can be viewed as auctions.

We can distinguish two important cases. In the **private value** model each bidder knows the value he places on the item, but does not know the values placed by other bidders. As bidding takes place, his valuation does not change, though he gains information from the other players' bids. In the **common value** model the item's actual value is the same for all bidders, but they have different *a priori* information about that value. Think, for example, of a jar of coins. Each player makes an estimate of the value of the coins in the jar, and as bidding occurs he can adjust his estimate based on what other players say. In this case the winner generally overestimates the value (since he had the highest estimate), and so he pays more than the jar of coins is worth. This is called the **winner's curse**.

Auctions can be **oral** (bidders hear each other's bids and make counter-offers) or **written** (bidders submit closed sealed-bids in writing). In an oral auction the number of bidders may be known, but in a sealed-bid auction it may be unknown. Some popular types of auction are the following.

1. **English auction** (or **ascending price auction**): bids increase in small increments until only one bidder remains.
2. **Dutch auction**: the price decreases continuously until some bidder calls stop.
3. **first price sealed-bid**: the winner pays his bid.

4. **second price sealed-bid** (or **Vickrey auction**): the winner pays the second highest bid.
5. **all pay sealed-bid auction**: highest bidder wins, but all pay their bid.

It is not hard to see that 1 and 4 are equivalent (with the item selling for the second greatest valuation), and that 2 and 3 are equivalent (with the item selling for the greatest bid).

### 23.2 The revenue equivalence theorem

The **symmetric independent private values model** (SIPV) concerns the auction of a single item, with risk neutral seller and bidders. Each bidder knows his own valuation of the item, which he keeps secret, and valuations of the bidders can be modelled as i.i.d. random variables. Important questions are

- what type of auction generates the most revenue for the seller?
- if seller or bidders are risk averse, which auction would they prefer?
- which auctions make it harder for the bidders to collude?
- can we compare auctions with respect to strategic simplicity?

Let us begin with an intuitive result.

**Lemma 23.1** *In any SIPV auction in which the bidders bid optimally and the item is awarded to the highest bidder, the order of the bids is the same as the order of the valuations.*

**Proof.** Let  $e(p)$  be the minimal expected payment that a bidder can make if he wants to win the item with probability  $p$ . A bidder who has valuation  $v$  and aims to win with probability  $p$  can make expected profit  $\pi(v) = pv - e(p)$ . Suppose the best  $p$  is  $p^*$  (which depends on  $v$ ) so the maximal profit is defined by

$$\pi^*(v) = \max_p [pv - e(p)] = p^*v - e(p^*) \quad \left. \frac{\partial \pi}{\partial p} \right|_{p=p^*} = v - e'(p^*) = 0. \quad (23.1)$$

Note that  $e(p)$  is convex and that at the stationary point,  $e'(p^*)p^* - e(p^*) > e(0)$ . Now as  $\partial \pi^* / \partial p^* = v - e'(p^*) = 0$ , we have

$$\frac{d\pi^*}{dv} = \frac{\partial \pi^*}{\partial v} + \frac{\partial \pi^*}{\partial p^*} \frac{dp^*}{dv} = p^*, \quad (23.2)$$

If the bidder had a valuation of  $v + \delta$  he could make the same bid that he makes with valuation  $v$ , in which he would win with the same probability and have the same expected payment; his expected profit would increase by  $\delta p^*$ . Hence  $\pi(v + \delta v) > \pi(v) + \delta \pi'(v)$ , implying  $\pi$  is convex in  $v$ . Hence  $d\pi^*(v)/dv$ , and so also  $p^*(v)$ , must increase with  $v$ . Since the item goes to the highest bidder the optimal bid must also increase with  $v$ . ■

We say that two auctions have the same **bidder participation** if any bidder who finds it profitable to participate in one auction also finds it profitable to participate in the other. The following is a remarkable result.

**Theorem 23.1 (Revenue equivalence theorem)** *The expected revenue obtained by the seller is the same for any two SIPV auctions that (a) award the item to the highest bidder, and (b) have the same bidder participation.*

It is remarkable, as various auctions can have completely different sets of rules and strategies. Suppose there are  $n$  bidders and all participate.

**Proof.** From (23.1) we have  $de(p(v))/dv = vdp/dv$ . Integrating this gives

$$e(p(v)) = e(p(0)) + \int_0^v wp'(w) dw = vp(v) - \int_0^v p(w) dw, \quad (23.3)$$

where clearly  $e(p(0)) = e(0) = 0$ , since there is no point in bidding for an item of value 0. Thus  $e(p(v))$  depends only on the function  $p(w)$ . We know from Lemma 23.1 that if bidders bid optimally then bids will be in the same order as the valuations. It follows that if  $F$  is the distribution function of the valuations, then  $p(w) = F(w)^{n-1}$ , independently of the precise auction mechanism. The expected revenue is  $\sum_{i=1}^n E_{v_i} e(p(v_i)) = nE_v e(p(v))$ . ■

**Example 23.1** Assume valuations are i.i.d. with distribution function  $F(u)$ .

(a) Suppose the item is simply offered at price  $p$  and sold if any player values it above  $p$ . The seller computes  $x(p)$ , the probability of making a sale if the price posted is  $p$ , and seeks to maximize  $px(p)$ . Then

$$x(p) = 1 - F^n(p), \quad \text{and} \quad p - \frac{1 - F^n(p)}{nF^{n-1}(p)f(p)} = 0.$$

If the distributions are uniform on  $[0, 1]$ ,  $F(u) = u$ , the optimal price is  $p^* = \sqrt[n]{1/n + 1}$ , and the resulting (expected) revenue is  $[n/(n + 1)] \sqrt[n]{1/n + 1}$ . For  $n = 2$ ,  $p^* = \sqrt{1/3}$ , and the expected revenue is  $(2/3)\sqrt{1/3} = 0.3849$ .

(b) Suppose the item is auctioned by any of the five mechanisms above. Let  $n = 2$ . If all bidders bid optimally then the probability that a bidder with valuation  $v$

wins is  $v$ , i.e.  $p(v) = v$ . From (23.3) we see that  $e(p(v)) = v^2/2$ . So in all these auctions the seller's expected revenue is  $2E[v^2/2] = 1/3 = 0.3333$ .

Let us compute the optimal bids in our auctions. Clearly, in the all-pay sealed-bid auction the optimal bid is  $e(p(v)) = v^2/2$ . In the Dutch or first price sealed-bid auctions, a bidder's expected payment is  $p(v)$  times his bid. Since this must equal  $v^2/2$  we find that the optimal bid must be  $v/2$ . In the second price sealed-bid (Vickrey) auction, the winner pays the bid of the second highest bidder. If bidder 1 bids  $u$ , then his profit is  $(v_1 - v_2)1_{\{u > v_2\}}$ . For every possible value of  $v_2$ , this is maximized by bidding  $u = v_1$ .

The seller prefers (a) to (b). However, in (a) he uses information about the distribution of the valuations. In (b) he makes no use of such information.

## 23.3 Risk aversion

Participants in an auction can have different attitudes to risk. If a participant's utility function is linear then he is said to be **risk-neutral**. If his utility function is concave then he is **risk-averse**; now a seller's average utility is less than the utility of his average revenue, and this discrepancy increases with the variability of the revenue. Hence a risk-averse seller, depending on his degree of risk-aversion, might choose an auction that substantially reduces the variance of his revenue, even though this might reduce his average revenue.

The revenue equivalence theorem holds under the assumption that bidders are risk-neutral. However, if bidders are risk-averse, then first-price sealed-bid auctions give different results from second-price sealed-bid auctions. In a first-price sealed-bid auction, a risk-averse bidder prefers to win more frequently even if his average net benefit is less. Hence he will make higher bids than if he were risk-neutral. This reduces his expected net benefit and increases the expected revenue of the seller. If the same bidder participates in a second-price auction, then his bids do not affect what he pays when he wins, and so his optimal strategy is to bid his true valuation. Hence, a first-price auction amongst risk-averse bidders produces a greater expected revenue for the seller than does a second-price auction.

The seller may also be risk-averse. In such a case, he prefers amongst auctions with the same expected revenue those with a smaller variance in the sale price.

Let us compare auctions with respect to this variance. Suppose bidders are risk-neutral. In the first price sealed-bid auction each bids half his valuation, so the revenue is  $(1/2)\max\{V_1, V_2\}$ . In the all-pay sealed-bid auction each pays half the square of his valuation and the revenue is  $\frac{1}{2}V_1^2 + \frac{1}{2}V_2^2$ , where  $V_1, V_2 \sim U[0, 1]$ . In the Vickrey auction each bids his valuation and the revenue is  $\min\{V_1, V_2\}$ . All these have expectation  $1/3$ , but the variances are  $1/72$ ,  $2/45$  and  $1/18$  respectively. Thus a risk adverse seller prefers the first price auction to the all-pay auction, which is preferred to the Vickrey auction.

## 24 Optimal and Multi-unit Auctions

### 24.1 Optimal auctions

Revenue-maximizing auctions are called **optimal auctions** and are achieved by imposing a **reserve price** or a **participation fee**. This reduces the number of participants, but leads to fiercer competition and higher bids on the average, which compensates for the probability that no sale takes place.

**Example 24.1 (Revenue maximization)** As in Example 23.1, suppose that there are two potential buyers, with unknown valuations,  $v_1, v_2$ , independent and uniformly distributed on  $[0, 1]$ . Consider two more ways of selling the object.

(c) Suppose the item is auctioned with an English auction, but with a participation fee  $c$  (which must be paid if a player chooses to submit a bid). Each bidder must choose whether or not to participate before knowing whether the other participates. Clearly, there is a  $v_0$  such that a player will not participate if he values the item at less than  $v_0$ . A bidder whose valuation is exactly  $v_0$  will be indifferent between participating or not. Hence  $P(\text{winning} | v = v_0)v_0 = c$ . Since a bidder with valuation  $v_0$  wins only if the other bidder has a valuation less than  $v_0$ , we must have  $P(\text{winning} | v = v_0) = v_0$ , and hence  $v_0^2 = c$ . Thus  $v_0 = \sqrt{c}$ .

We note that there are two ways that revenue can accrue to the seller. Either only one bidder participates and the sale price is zero, but the revenue is  $c$ . Or both bidders have valuations above  $v_0$ , in which case the revenue is  $2c$  plus the sale price of  $\min\{v_1, v_2\}$ . The expected revenue is

$$2v_0(1 - v_0)c + (1 - v_0)^2[2c + v_0 + (1 - v_0)/3].$$

This is maximized for  $c = 1/4$  and takes the value  $5/12 (= 0.4167)$ .

(d) Suppose the item is auctioned with an English auction, but with a reserve price  $p$ , such that the bidding starts with a minimum bid of  $p$ . There is no sale with probability  $p^2$ . The revenue is  $p$  with probability  $2p(1 - p)$ . If  $\min\{v_1, v_2\} > p$ , then the sale price is  $\min\{v_1, v_2\}$ . The expected revenue is

$$2p^2(1 - p) + \left(\frac{1}{3} + \frac{2}{3}p\right)(1 - p)^2.$$

This is maximized by  $p = 1/2$  and the expected revenue is again  $5/12$ , exactly the same as in (c).

That cases (c) and (d) give the same expected revenue is not a coincidence. In both auctions a bidder participates if and only if his valuation exceeds  $1/2$ . Let us consider more generally an auction in which a bidder participates only if his valuation exceeds some  $v_0$ . Suppose that with valuation  $v$  it is optimal to bid so

as to win with probability  $p(v)$ , and the expected payment is then  $e(p(v))$ . By a simple generalization of (23.3), we have

$$e(p(v)) = e(p(v_0)) + \int_{v_0}^v w \frac{dp(w)}{dw} dw = vp(v) - \int_{v_0}^v p(w) dw.$$

Assuming the SIPV model, this shows that a bidder's expected payment depends on the auction mechanism only through the value of  $v_0$  that it implies. The seller's expected revenue is

$$\begin{aligned} nE_v[e(p(v))] &= n \int_{v=v_0}^{\infty} \left[ vp(v) - \int_{w=v_0}^v p(w) dw \right] f(v) dv \\ &= n \int_{v=v_0}^{\infty} vp(v)f(v) dv - n \int_{w=v_0}^{\infty} \int_{v=w}^{\infty} p(w)f(v) dv dw \\ &= n \int_{v=v_0}^{\infty} \{vf(v) - [1 - F(v)]\} F(v)^{n-1} dv. \end{aligned}$$

Now differentiating with respect to  $v_0$ , to find the stationary point, we see that the above is maximized where

$$v_0f(v_0) - [1 - F(v_0)] = 0.$$

We call  $v_0$  the **optimal reservation price**. Note that it does not depend on the number of bidders. For example, if valuations are uniformly distributed on  $[0, 1]$ , then  $v_0 = 1/2$ , consistent with the answers found for (c) and (d) above.

If bidders' valuations are independent, but not identically distributed, we can proceed similarly. Let  $p_i(v)$  be the probability that bidder  $i$  wins when his valuation is  $v$ . Let  $e_i(p)$  be the minimum expected amount he can pay if he wants to win with probability  $p$ . Suppose that bidder  $i$  does not participate if his valuation is less than  $v_{0i}$ . Just as above, one can show that the seller's expected revenue is

$$\sum_{i=1}^n E_{v_i} e_i(p_i(v_i)) = \sum_{i=1}^n \int_{v=v_{0i}}^{\infty} \left[ v - \frac{1 - F_i(v)}{f_i(v)} \right] f_i(v) p_i(v) dv. \quad (24.1)$$

The term in square brackets can be interpreted as 'marginal revenue', in the sense that if a price  $p$  is offered to bidder  $i$ , he will accept it with probability  $x_i(p) = 1 - F_i(p)$ , and so the expected revenue obtained by this offer is  $px_i(p)$ . Therefore, differentiating  $px_i(p)$  with respect to  $x_i$ , we define

$$MR_i(p) = \frac{d}{dx_i}(px_i(p)) = \frac{d}{dp}(px_i(p)) \left/ \frac{dx_i}{dp} \right. = p - \frac{1 - F_i(p)}{f_i(p)}.$$

The right hand side of (24.1) is simply  $E[MR_{i^*}(v_{i^*})]$ , where  $i^*$  is the winner of the auction. This can be maximized simply by ensuring that the object is always



awarded to the bidder with the greatest marginal revenue, provided that marginal revenue is positive. We can do this, provided bidders reveal their true valuations. Let us assume that  $MR_i(p)$  is increasing in  $p$  for all  $i$ . Clearly,  $v_{0i}$  should be the least  $v$  such that  $MR_i(v)$  is nonnegative. Consider the auction rule that always awards the item to the bidder with the greatest marginal revenue, and then asks him to pay the maximum of  $v_{0i}$  and the smallest  $v$  for which he would still remain the bidder with greatest marginal revenue  $v_{0i}$ . This has the character of a second-price auction in which the bidder's bid does not affect his payment, given that he wins. So bidders will bid their true valuations and (24.1) will be maximized.

**Example 24.2** An interesting property of optimal auctions with heterogeneous bidders is that the winner is not always the highest bidder.

Consider first the case of homogeneous bidders with valuations uniformly distributed on  $[0, 1]$ . In this case  $MR_i(v_i) = v_i - (1 - v_i)/1 = 2v_i - 1$ . The object is sold to the highest bidder, but only if  $2v_i - 1 > 0$ , i.e., if his valuation exceeds  $1/2$ . The winner pays either  $1/2$  or the second greatest bid, whichever is greatest. In the case of two bidders with the identical uniformly distributed valuations the seller's expected revenue is  $5/12$ . This agrees with what we have found previously.

Now consider the case of two heterogeneous bidders, say  $A$  and  $B$ , whose valuations are uniformly distributed on  $[0, 1]$  and  $[0, 2]$  respectively. So  $MR_A(v_A) = 2v_A - 1$ ,  $v_{0A} = 1/2$ , and  $MR_B(v_B) = 2v_B - 2$ ,  $v_{0B} = 1$ . Under the bidding rules described above, bidder  $B$  wins only if  $2v_B - 2 > 2v_A - 1$  and  $2v_B - 2 > 0$ , i.e., if and only if  $v_B - v_A > 1/2$  and  $v_B > 1$ ; so the lower bidder can sometimes win. For example, if  $v_A = 0.8$  and  $v_B = 1.2$ , then  $A$  wins and pays  $0.7$  (which is the smallest  $v$  such that  $MR_A(v) = 2v - 1 \geq 2v_B - 2 = 0.4$ ).

## 24.2 Multi-unit and combinatorial auctions

Multi-unit auctions are of great practical importance, and have been applied to selling units of bandwidth in computer networks and satellite links, MWs of electric power, capacity of natural gas and oil pipelines. These auctions can be homogeneous or heterogeneous. In a **homogeneous auction** a number of identical units of a good are to be auctioned, and we speak of a **multi-unit auction**. In the simplest multi-unit auction, each buyer wants only one unit. The auction mechanisms above can be generalized. For example, in a **simultaneous auction** of  $k$  units, all bidders could make closed sealed-bids, and the  $k$  objects could be awarded to the  $k$  highest bidders. In a first-price auction each bidder would pay his own bid. In a generalization of the Vickrey auction the  $k$  highest bidders would pay the value of the highest losing bid. It can be shown that the revenue-equivalence theorem still holds for these auctions. Note that in the first-price auction the successful bidders pay differently for the same thing; we call

this is a **discriminatory auction**. By contrast, the Vickrey auction is called a **uniform auction**, because all successful bidders pay the same. A uniform auction is intuitively fairer, and also more likely to reduce the winner's curse.

Things are more complex if bidders want to buy more than one object, or if objects are different, and perhaps complementary. For example, the value of holding two cable television licenses in contiguous geographic regions can be greater than the sum of their values if held alone. This means that it can be advantageous to allow **combinatorial bidding**. Here, bidders may place bids on groups of objects as well as on individual objects. A generalization of the Vickrey auction that can be used with combinatorial bidding is the **Vickrey-Clarke-Groves (VCG) mechanism**. Each bidder submits bids for any combinations of objects he wishes. The auctioneer allocates the objects to maximize the aggregate total of their values to the bidders. Each bidder who wins a subset of the objects pays the 'opportunity cost' that this imposes on the rest of the bidders.

More specifically, let  $L$  be the set of objects and  $P$  be the set of their possible assignments amongst the bidders. Each bidder submits a bid that specifies a value  $v_i(T)$  for each non-empty subset  $T$  of  $L$ . An assignment  $S \in P$  is a partition of  $L$  into subsets  $S_i$ , with one such subset per bidder  $i$  (possibly empty). If social welfare maximization is the objective, then the auctioneer chooses the partition  $S^* = \{S_1^*, \dots, S_n^*\}$  that maximizes  $\sum_{i=1}^n v_i(S_i^*)$ . Bidder  $i$  pays  $p_i$ , where

$$p_i = \max_{S \in P} \sum_{j \neq i} v_j(S_j) - \sum_{j \neq i} v_j(S_j^*). \quad (24.2)$$

The first term on the right of (24.2) is the greatest value that could be obtained by the other bidders if  $i$  were not bidding. The final term is the value that is obtained by the other bidders when bidder  $i$  does participate, and so influences the optimal allocation and takes some value for himself.

This type of auction is **incentive compatible**, in the sense that bidders are incentivized to submit their true valuations, and it leads to an economically efficient allocation of the objects. It has the advantage that the whole market is available to bidders and they can freely express their preferences for substitutable or complementary goods. However, there are drawbacks. Firstly, the complex mechanism of a VCG auction can be hard for bidders to understand. It is not intuitive and bidders may well not follow the proper strategy. Secondly, it is very hard to implement. This is because each bidder must submit an extremely large number of bids, and the auctioneer must solve a  $NP$ -complete optimization problem to determine the optimal partition (and also each of the  $p_i$ ), so the 'winner determination' problem can be unrealistically difficult to solve. There are several ways that bidding can be restricted so that the optimal partitioning problem becomes solvable in polynomial time. Unfortunately, these restrictions are rather strong, and are not applicable in many cases of practical interest.

- $\epsilon$ -approximation algorithm, 62
- 2OPT, 64
- acyclic, 35
- affine transformation, 25
- algorithm, 19
- all pay sealed-bid auction, 92
- approximation algorithms, 59
- arcs, 35
- artificial variables, 13
- ascending price auction, 91
- assignment problem, 41
- bargaining set, 77
- basic, 8
- basic feasible solution, 8, 9
- basic solution, 8, 9, 36
- basis, 8, 9
- Bellman's equations, 46
- Bellman-Ford algorithm, 47
- bidder participation, 93
- bipartite graph, 39
- Branch and Bound, 52
- characteristic function, 79
- circulation, 38
- coalition, 79
- coalitions, 79
- combinatorial bidding, 98
- common value, 91
- complementary slackness conditions, 7
- connected, 35
- convex function, 5
- convex set, 5
- cooperation, 73
- core, 80
- cost, 35
- Cournot equilibrium, 86
- critical path, 45
- cutset formulation of the TSP, 59
- cycle, 35
- Dakin's method, 55
- Dijkstra's algorithm, 48
- directed graph, 35
- discriminatory auction, 98
- dominates, 70, 80
- dual problem, 2, 7
- dual simplex algorithm, 16
- duopoly, 85
- Dutch auction, 91
- dynamic programming, 46
- ellipsoid, 25
- ellipsoid method, 25
- English auction, 91
- equilibrium pair, 71, 73
- evolutionary stable strategy, 89
- exact algorithms, 59
- exponential time, 19
- extreme point, 5
- facility location problem, 58
- feasible, 1
- feasible spanning tree solution, 36
- first price sealed-bid, 91
- follower, 87
- Ford-Fulkerson algorithm, 43
- full-dimensional, 32
- Game theory, 67
- genetic algorithms, 66
- Gomory's cutting plane method, 18
- graph, 35
- heuristic algorithms, 59
- Hirsch conjecture, 24
- Hitchcock transportation problem, 40
- homogeneous auction, 97
- imputation, 79
- incentive compatible, 98
- instance, 19
- instance size, 19
- integer linear program, 51
- jointly dominated, 76
- knapsack problem, 57
- label-correcting algorithms, 46
- label-setting algorithms, 46
- labels, 46
- Lagrangian, 1, 7, 37
- Lagrangian multiplier, 1
- leader, 87
- linear programming relaxation, 55
- lower value, 68
- LP-relaxation, 42
- Maximin bargaining solution, 78
- maximin criterion, 68
- minimum cost flow, 36
- minimum cost flow problem, 35
- mixed integer program, 55
- multi-unit auction, 97
- neighbourhood search, 63
- network, 35
- nodes, 35
- non-basic, 8
- non-degenerate, 8
- nondeterministic polynomial, 20
- normal form representation, 67
- nucleolus, 82
- oligopoly, 85
- optimal auctions, 95
- optimal reservation price, 96
- Pareto optimal, 76
- participation fee, 95
- path, 35
- perfect information, 67
- pivot, 11
- pivot column, 11
- pivot row, 11
- polynomial reduction, 21
- polynomial time, 19
- prices, 7
- primal problem, 2
- private value, 91
- pure integer program, 55
- pure strategies, 67, 68
- reduced costs, 37
- reserve price, 95
- risk-averse, 94
- risk-neutral, 94
- running time, 19
- saddle point solution, 68
- second price sealed bid, 92
- shadow prices, 7
- Shapley values, 83
- simultaneous auction, 97
- sink, 35
- solution, 69
- source, 35
- spanning tree, 35
- spanning tree solution, 36
- Stackleberg, 87
- Strong Lagrangian, 2, 3
- subnetwork, 35
- subtour elimination formulation of the TSP, 60
- superadditivity, 79
- supporting hyperplane, 3
- symmetric independent private values model, 92
- tableau, 9
- time complexity, 19
- travelling salesman problem, 59
- tree, 35
- uncapacitated flows, 36

undirected graph, 35  
uniform auction, 98  
upper value, 68

value, 69  
Vickrey auction, 92  
Vickrey-Clarke-Groves (VCG) mechanism, 98  
volume, 25

walk, 35  
weak duality theorem, 2  
winner's curse, 91  
worst-case, 19

zero-one programs, 57  
zero-sum, 67