Mathematics of Operational Research

R.R. Weber

This course is accessible to a candidate with mathematical maturity who has no previous experience of operational research; however it is expected that most candidates will already have had exposure to some of the topics listed below.

- Lagrangian sufficiency theorem. Strong Lagrangian problems. Supporting hyperplane theorem. Sufficient conditions for convexity of the optimal value function. Fundamentals of linear programming. Dual linear program. Shadow prices. [2]

- Simplex algorithm. Two-phase method. Dual simplex algorithm. Gomory's cutting plane methods for integer linear programming. [3]

- Complexity of algorithms: typical and worst-case behaviour. Classes of NP-hard and NP-complete. Exponential complexity of the simplex algorithm. Polynomial time algorithms for linear programming. Ellipsoid algorithm. [2]

- Network simplex algorithm, transportation and assignment problems, general minimum-cost circulation problems. Ford-Fulkerson algorithm, max-flow/min-cut theorem. Shortest and longest path problems, Dijkstra's algorithm, project management, critical paths. Minimal spanning tree problem, Prim's algorithm. MAX CUT. Semidefinite programming. Rendezvous problem. Interior point methods. [5]

- Branch and bound method. Dakin's method. The travelling salesman problem. Heuristic methods. Neighbourhood search. Simulated annealing. [3]

- Two-person zero-sum games. Cooperative and non-cooperative games. Nash equilibria: existence and construction. Lemke-Howson algorithm. Bargaining games. Coalitional games: core, nucleolus, Shapley value. Market games. Auctions, revenue equivalence theorem, optimal auctions. Mechanism design. Pricing and auction methods for decentralized control. [9]

Books

1. L.C. Thomas, Games, Theory and Application, Wiley, Chichester (1984).

2. M.S. Bazaran, J.J. Harvis and H.D. Shara'i, Linear Programming and Network Flows, Wiley (1988).

3. D. Bertsimas and J.N. Tsitsiklis, Introduction to Linear Optimization, Athena Scientific (1997). Undoubtedly the best book on the market.

Contents

1 Lagrangian Methods

1.1 Lagrangian methods

Let $P(b)$ denote the optimization problem

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

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 λ 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 λ 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 $a_1x_1 + a_2x_2 = b$, $x_1, x_2 \geq 0$. Let us suppose a_1, a_2, and b are all nonnegative.

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

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

This has a stationary point where $(x_1, x_2) = (\lambda a_1/2, \lambda a_2/2)$. Now we choose λ 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 λ 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 λ

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

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 λ it is interesting to make this lower bound as large as possible. Of course we can restrict ourselves to λ 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

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

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 λ such that

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

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 λ 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 λ 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. I.e. the inequality in (1.2) is actually an equality.

1.4 Hyperplanes

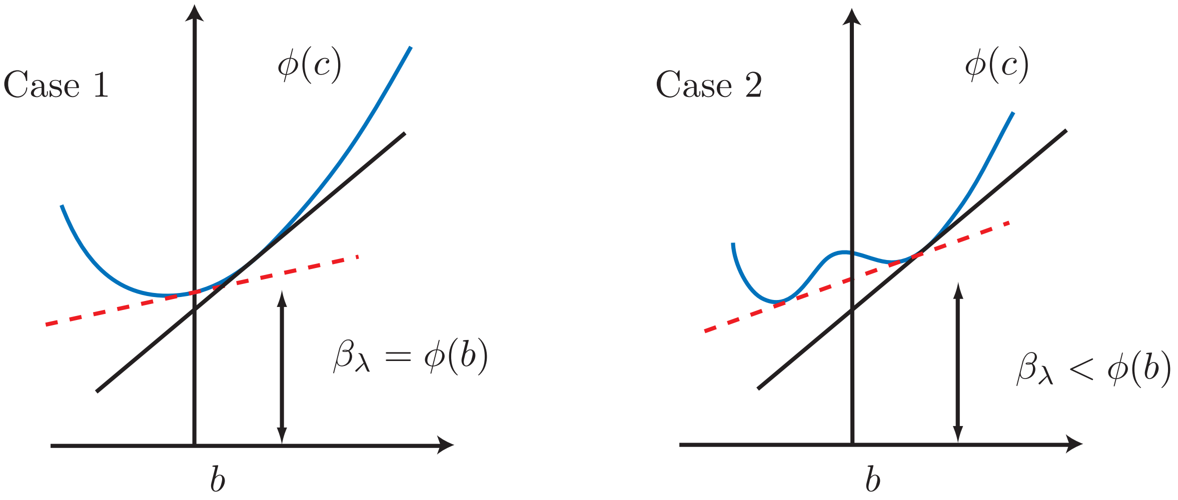
Let the hyperplane (c, α) be given by

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

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

1. For each λ, find $\beta_\lambda \equiv \max \beta$ such that the hyperplane lies completely below the graph of ϕ.

2. Now choose λ to maximize β_λ.



Lagrangian methods work in Case 1 because of the existence of a tangent to ϕ at b. Define a **supporting hyperplane** (c, α) 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

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

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 ϕ 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 ϕ convex.

Proof. Suppose there exists a (non-vertical) supporting hyperplane to ϕ at b. This means that there exists λ such that

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

This implies

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

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 λ such that for all $x \in X$

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

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

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

This is true for all c, and hence

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

Hence, ϕ 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 $0 \leq \delta \leq 1$

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

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 ϕ is convex and b lies in the interior of the set of points where ϕ is finite. Then there exists a (non-vertical) supporting hyperplane to ϕ at b.*

So, we are interested in conditions on the problem that make ϕ convex.

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

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

If X is a convex set and f and h are convex then ϕ 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 ϕ 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 ϕ is convex. ■

Remark. Consider the constraint $h(x) = b$. This is the same as $h(x) \leq b$ and $-h(x) \leq -b$. So ϕ 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[1]

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

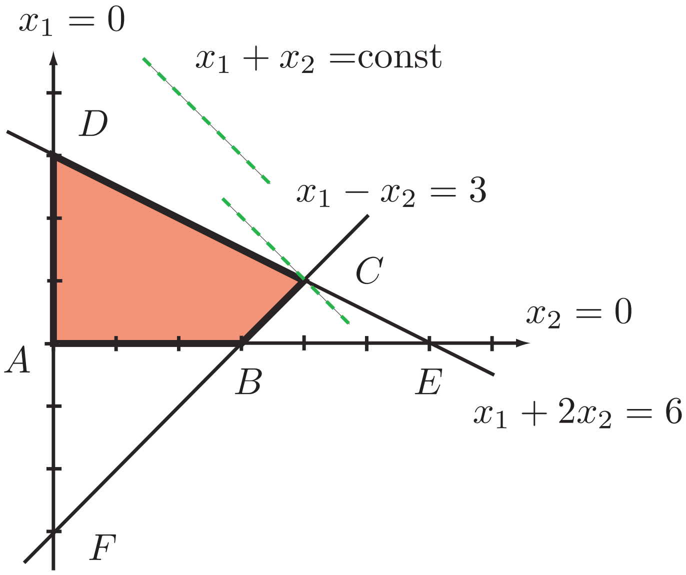
where x and c are n-vectors, b is a m-vector and A is a $m \times n$ matrix. Such problems are also written out in the longer form

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

Example

$$\begin{array}{l} \text{minimize } \; -(x_1 + x_2) \\ \text{subject to} \end{array}$$

$$\begin{array}{cccl} x_1 & + & 2x_2 & \leq 6 \\ x_1 & - & x_2 & \leq 3 \\ & x_1, & x_2 & \geq 0\,. \end{array}$$



2.3 Duality of linear programs

The primal LP optimization problems

$$\begin{array}{ll} (LP =): & \text{minimize}\{c^\top x \,:\, Ax = b,\, x \geq 0\} \\ (LP \geq): & \text{minimize}\{c^\top x \,:\, Ax \geq b,\, x \geq 0\} \end{array}$$

[1] 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

$$\begin{aligned} &\text{Dual of}\,(LP =): &&\text{maximize}\{b^\top\lambda \,:\, A^\top\lambda \le c\} \\ &\text{Dual of}\,(LP \ge): &&\text{maximize}\{b^\top\lambda \,:\, A^\top\lambda \le c,\, \lambda \ge 0\} \end{aligned}$$

2.4 Derivation of the dual LP problem

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

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

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

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

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

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

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

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

The minimum of $L((x, z); \lambda)$ for $\lambda \in Y$ is attained where both $\left(c^\top - \lambda^\top A\right)x = 0$ 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 λ_i are also known as **shadow prices**.

2.6 Conditions for optimality

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

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

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!solutionbasic 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., $\operatorname{rank}(A) = m$.
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 $x \geq 0 \Longrightarrow x_N \geq 0$)

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

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

$$Ax \le b\,, \qquad b \ge 0\,.$$

We can write this as

$$Ax + z = b\,, \qquad z \ge 0$$

and take an initial basic feasible solution of

$$x = 0\,, \qquad z = b \ge 0\,.$$

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 $a_{0j} \le 0$ for all $j \ge 1$ 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 $j \ge$ 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 $a_{ij} \le 0$ 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	$\boxed{1}$	-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	$\boxed{3}$	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

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

Unfortunately, the basic solution

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

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

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

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	$\boxed{2}$	-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	$\boxed{\frac{1}{2}}$	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 preceeding tableau without the y_1, y_2 columns.

Begin Phase II.

	x_1	x_2	z_1	z_2	z_3	
z_2	0	$\boxed{3}$	-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}.

$$
\begin{array}{c|ccccc|c|}
 & x_1 & x_2 & z_1 & z_2 & z_3 & \\
\cline{2-7}
x_2 & 0 & 1 & -\frac{2}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\
x_1 & 1 & 0 & -\frac{1}{3} & -\frac{1}{3} & 0 & \frac{2}{3} \\
z_3 & 0 & 0 & 2 & -1 & 1 & 1 \\
\cline{2-7}
 & 0 & 0 & -4 & -1 & 0 & 5 \\
\cline{2-7}
\end{array}
$$

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 minimize$\{c^\top x : Ax = b, x \geq 0\}$. This has dual 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

$$
\boxed{\begin{array}{c}\text{primal}\\ \text{feasibility}\\ Ax = b \text{ and } x \geq 0\end{array}}
+
\boxed{\begin{array}{c}\text{dual}\\ \text{feasibility}\\ (c^\top - \lambda^\top A) \geq 0\end{array}}
+
\boxed{\begin{array}{c}\text{complementary}\\ \text{slackness}\\ (c^\top - \lambda^\top A)x = 0\end{array}}
$$

$\Longrightarrow$ 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

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

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
$\boxed{-2}$	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	$\boxed{-\frac{5}{2}}$	$-\frac{5}{2}$	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

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

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

Hence

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

So the dual variables, λ, 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

$$\begin{array}{|c|c|c|}
\multicolumn{1}{c}{\text{non-basic}} & \multicolumn{1}{c}{\text{basic}} & \multicolumn{1}{c}{} \\
\hline
 & I & +\text{ve} \\
\hline
+\text{ve} & 0 & \\
\hline
\end{array}$$

Now we wish to add a new constraint

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b\,.$$

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

$$\begin{array}{|c|c|c|c|}
\hline
 & & 0 & \\
 & I & \vdots & +\text{ve} \\
 & & 0 & \\
\hline
a_N & a_B & 1 & b \\
\hline
+\text{ve} & 0 & 0 & \\
\hline
\end{array}
\longrightarrow
\begin{array}{|c|c|c|c|}
\hline
 & & 0 & \\
 & I & \vdots & +\text{ve} \\
 & & 0 & \\
\hline
 & 0 & 1 & \text{-ve?} \\
\hline
+\text{ve} & 0 & 0 & \\
\hline
\end{array}$$

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

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

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

$$x_2 + x_3 - 1z_1 + 0z_2 \le x_2 + x_3 - \tfrac{2}{5}z_1 + \tfrac{1}{5}z_2 = \tfrac{2}{5}\,.$$

This is because $z_1, z_2 \ge 0$ 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

$$x_2 + x_3 - z_1 \le 0\,.$$

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	$-\frac{3}{5}$	$\boxed{-\frac{1}{5}}$	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 interation.

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 asnwer, $f(x)$. 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 Turning 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 σ that minimizes the sum of distances

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

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

$$\textit{Given the points } S\textit{, and } L \textit{ is there a tour of length} \leq L? \tag{5.1}$$

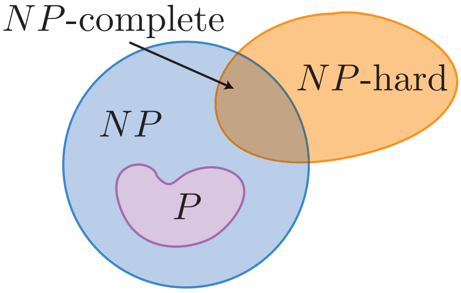
This differs from optimization-TSP: *find the shortest tour*, or the evaluation-TSP: *find the length of the shortest tour*. Of course the three types of problem are closely related. We focus on decision problems. This avoids problems in which the size of the input or output causes non-polynomial time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

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

A decision problem in is $\mathcal{P}$ 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 Π_1 no harder than another problem Π_2? We say that Π_1 **reduces to** Π_2 if we can construct an algorithm for Π_1 as follows.

1. Make a (polynomial time) transformation of the instance of Π_1 into an instance of Π_2.

2. Apply some algorithm to solve this instance of Π_2.

3. Make a (polynomial time) transformation of this solution of Π_2 back into a solution of Π_1.

The idea is that we can use an algorithm that solves Π_2 to solve Π_1, with additional work in steps 1 and 3 that requires at most polynomial time. Thus Π_1 is really no harder than Π_2.

5.6 $\mathcal{NP}$-completeness

Now we can talk about the hardest problems in $\mathcal{NP}$. A problem Π 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, Π, 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 Π.

5.7 Examples of 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,

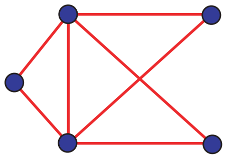
$$(X_1 \text{ OR NOT } X_2) \text{ AND } (X_3 \text{ AND } X_4)$$

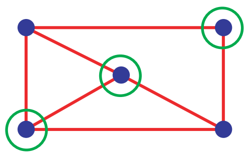
is there an assignment of truth values to the variables that makes the formulae true? (e.g. $X_1 = X_2 = X_3 = X_4 = T$ in the above example.)

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 ismorphism (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

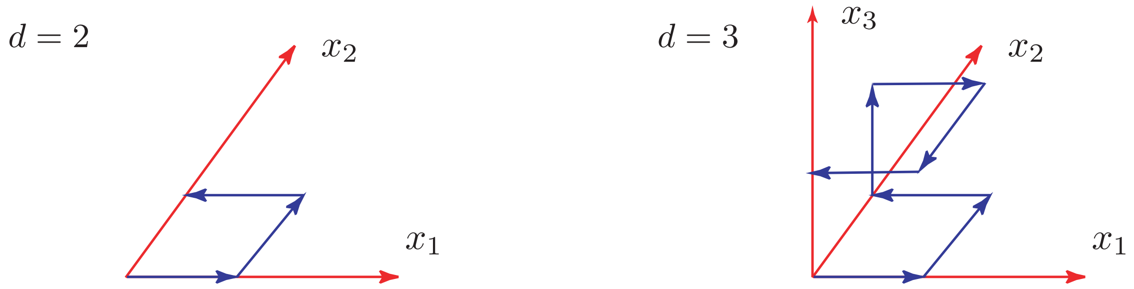
6.1 Running time of the simplex algorithm

The simplex algorithm moves from one basic feasible solution to an adjacent one such that the value of the objective function improves. However, in the **worst-case** 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

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

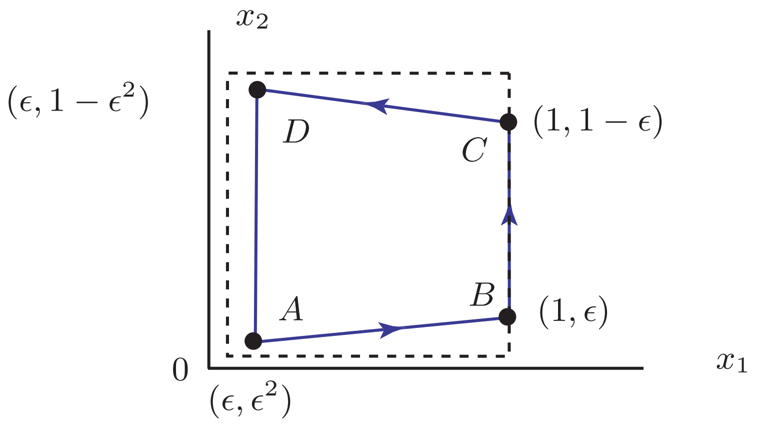
and we seek to maximize x_d. There are 2^d vertices. The paths shown below visit all vertices before terminating at $(0, 0, \dots, 1)$.



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

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

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

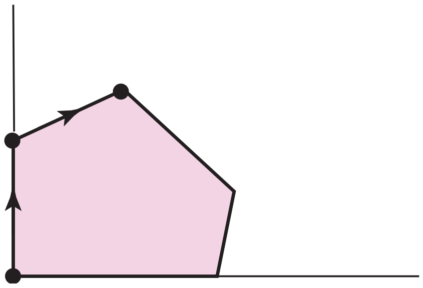


Note that x_2 increases along the route ABCD. So if our pivoting rule is always to move to the adjacent bfs for which the entering variable has the least index (so-called Bland's rule), then the simplex algorithm will require $2^d - 1$ pivoting steps before terminating. With this pivoting rule the simplex 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 (1957): that the diameter of a polytope with dimension d with n facets cannot be greater than $n-d$. This conjecture was disproved in 2010 by Francisco Santos, who found a 43-dimensional polytope with 86 facets and diameter more than 43. However, it remains open whether or not the diameter might be bounded by some polynomial function of n and d.

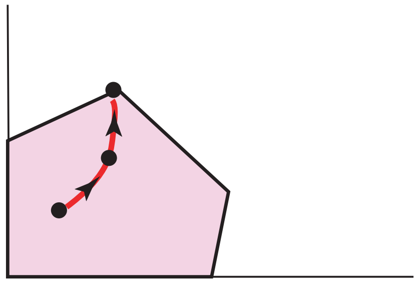
The fact that an algorithm performs badly in the worst-case does not necessarily mean that it is bad in practice. In fact, the **average-case running time** of the simplex algorithm appears to linear in problem size. The difficulty is defining what is meant by 'on average'.

6.2 The size of an LP instance

We have seen that the simplex algorithm is not a polynomial-time algorithm. However, there are other algorithms for solving LP. If we can find a polynomial-time algorithm then LP $\in P$. There are two classes of methods for solving LP.



Boundary value methods



Interior point methods

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

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

where $a_0, a_1, \ldots, a_k$ 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

$$(mn + m + n)(\lfloor \log_2 U \rfloor + 2)\,.$$

6.3 Equivalent feasibility problem

Consider the primal/dual pair:

$$\begin{aligned} P:&\ \text{minimize}\,\{c^\top x\ :\ Ax \geq b\} \\ D:&\ \text{maximize}\,\{b^\top \lambda\ :\ A^\top \lambda = c, \lambda \geq 0\}\,. \end{aligned}$$

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

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

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 Vol(L), is defined by*

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

We use the fact 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

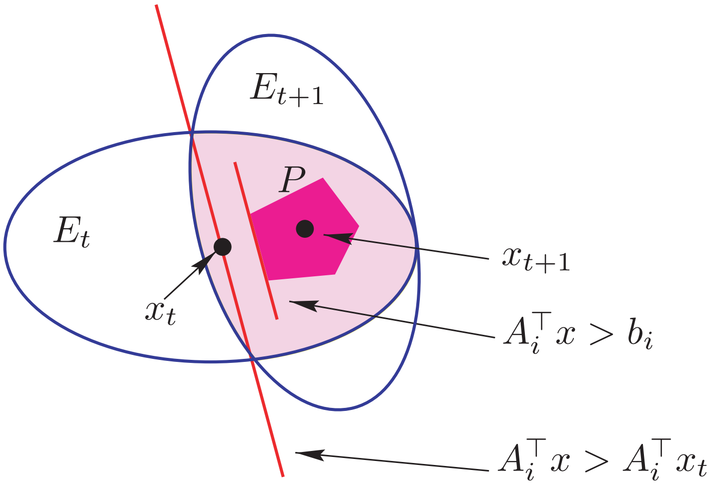
We generate a sequence of ellipsoids $\{E_t\}$. E_t has centre x_t, such that

- 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*

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

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

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

(a) $E \cap H \subset E'$,

(b) $\mathit{Vol}(E') < e^{-1/(2(n+1))}\, \mathit{Vol}(E)$.

7 Ellipsoid Method

7.1 Khachiyan's ellipsoid algorithm

Khachiyan's ellipsoid method (1979):

Input

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

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

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

(i) If $t = t^*$ stop; P is empty.

(ii) If $x_t \in P$ stop; P is non-empty.

(iii) If $x_t \notin P$ find a violated constraint, i, such that $A_i^\top x_t < b_i$.

(iv) 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

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

(v) $t := t + 1$, return to (i).

7.2 Sketch proof for the ellipsoid algorithm

To prove that the algorithm runs in polynomial time we show the following.

(a) E_{t+1} as defined in (iv) above, does in fact contain $E_t \cap H_t$.

Suppose $E_0 = \{x \in \mathbb{R}^n : x^\top x \leq 1\}$ and $H_0 = \{x \in \mathbb{R}^n : x_1 \geq 0\}$. Let $e_1^\top = (1, 0, \dots, 0)$. Then

$$\begin{aligned} E_1 &= E\left(\frac{e_1}{n+1}, \frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^\top\right)\right) \\ &= \left\{x \in \mathbb{R}^n : \frac{n^2-1}{n^2}\sum_{i=1}^n x_i^2 + \frac{1}{n^2} + \frac{2(n+1)}{n^2}x_1(x_1-1) \leq 1\right\}. \end{aligned}$$

One can now check that if $x \in E_0 \cap H_0 \implies x \in E_1$.

(b) $\mathbf{Vol}(E_{t+1}) < e^{-1/(2(n+1))}\mathbf{Vol}(E_t)$.

Suppose E_0 and E_1 are as in (a), and $E_1 = F(E_0)$, where

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

Then, for some affine map $G(x) = t + Sx$,

$$\begin{aligned} \frac{\mathrm{Vol}(E_{t+1})}{\mathrm{Vol}(E_t)} = \frac{\mathrm{Vol}(G(E_1))}{\mathrm{Vol}(G(E_0))} = \frac{\mathrm{Vol}(E_1)}{\mathrm{Vol}(E_0)} &= \sqrt{\det\left(\frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^\top\right)\right)} \\ &= \left(\frac{n^2}{n^2-1}\right)^{n/2}\left(1 - \frac{2}{n+1}\right)^{1/2} \\ &= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2} \\ &= \left(1 - \frac{1}{n+1}\right)\left(1 + \frac{1}{n^2-1}\right)^{(n-1)/2} \\ &< e^{-1/(n+1)}\left(e^{1/(n^2-1)}\right)^{(n-1)/2} = e^{-1/(2(n+1))}, \end{aligned}$$

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

(c) There exists V s.t. if P is nonempty then it lies completely within a ball of volume V.

We start with a lemma.

Lemma 7.1 *Let A be a $m \times n$ matrix ($m > n$) of integers 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 every extreme point of the polyhedron $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ satisfies*

$$-(nU)^n \leq x_j \leq (nU)^n\,, \qquad j = 1, \ldots, n\,.$$

Proof. Any extreme point of P is of the form $x = \widehat{A}^{-1}\widehat{b}$, where $\widehat{A}$ is an $n \times n$ invertible submatrix of A and $\widehat{b}$ is the matching n-dimensional subvector of b. So By Cramer's rule, we can write

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

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

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

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

Finally, since $\widehat{A}$ is invertible, $\det(\widehat{A}) \neq 0$ and all entries in A are integer so $|\det(\widehat{A})| \geq 1$. So the extreme point x satisfies $|x_j| \leq (nU)^n$, for all j. ■

Thus if x is an extreme point, then $x^\top x \leq n(nU)^{2n}$, and so it lies in the set

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

P is nonempty if and only if it contains an extreme point so we can test for nonemptiness of P_B instead of P.

Now P_B is contained in a ball of radius $\sqrt{n}(nU)^n$, and this ball lies within a cube of volume $V = (2\sqrt{n})^n(nU)^{n^2}$.

(d) There exists $v > 0$ s.t. P is empty or Vol$(P) > v$.

We say 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 Vol$(P) = 0$ and so is not full-dimensional.

Lemma 7.2 *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*

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

where $e^\top = (1, 1, \ldots, 1)$. Then *(a) If P is empty, then P_ϵ is empty.*

(b) If P is non-empty, then P_ϵ is full-dimensional.

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

$$\lambda^\top A = 0^\top \qquad \lambda^\top b = 1\,.$$

Using Lemma 7.1, we can find a bfs $\widehat{\lambda}$ to the constraints $\lambda^\top A = 0^\top$, $\lambda^\top b = 1$, $\lambda \geq 0$ such that

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

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

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

Therefore,

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

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_ϵ is also empty.

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

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

It is easy to show that y belongs to P_ϵ 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 7.3 *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,*

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

(e) The running time of $t^* = \lceil 2(n+1)\log(V/v)\rceil$ is polynomial in n.

We have the values

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

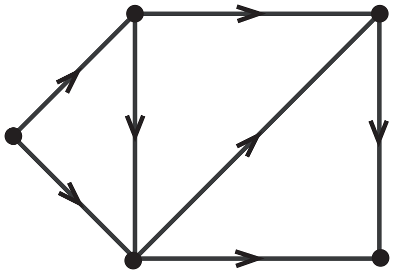
and know that the ellipsoid method takes at most $t^* = \lceil 2(n+1)\log(V/v)\rceil$ steps. This gives $t^* = O(n^4 \log(nU))$.

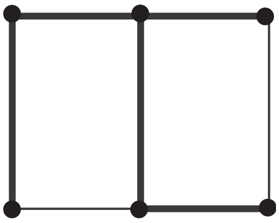
8 The Network Simplex Algorithm

8.1 Graph terminology

Lectures 8–11 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 $i_1, i_2, \ldots, i_t$ 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 $k = 1, \ldots, t-1$. A walk is a **path** if $i_1, i_2, \ldots, i_k$ are distinct, and a **cycle** if $i_1, i_2, \ldots, i_{k-1}$ are distinct and $i_1 = i_k$. A graph is **connected** if there is a path connecting every pair of nodes.





A 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 $N' = N$.

8.2 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

$$b_i + \sum_{j:(j,i)\in A} f_{ji} = \sum_{j:(i,j)\in A} f_{ij}\,, \quad \text{for all } i \in N$$

$$m_{ij} \le f_{ij} \le M_{ij}\,, \quad \text{for all } (i,j) \in A\,.$$

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

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

8.3 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 $m_{ij} \le f_{ij} \le M_{ij}$ is a **feasible spanning tree solution**.

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

8.4 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 $m_{ij} \le f_{ij} \le M_{ij}$ gives dual feasibility and complementary slackness conditions:

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

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

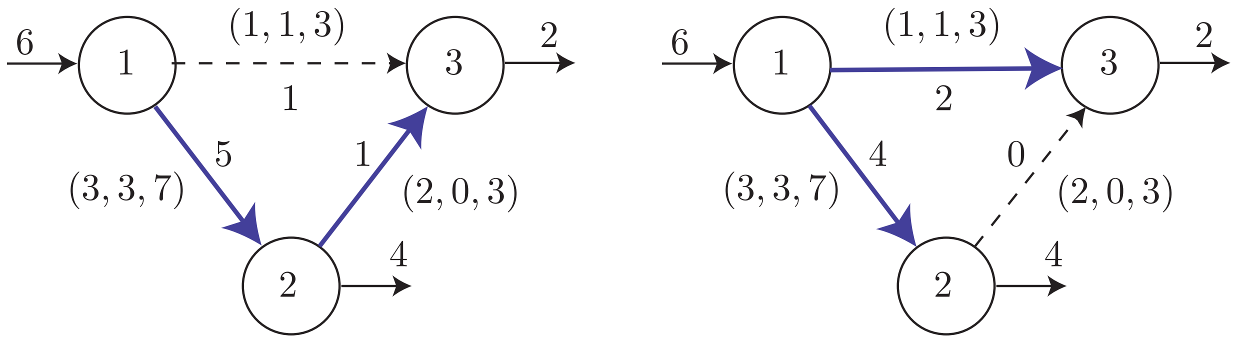
$$\lambda_n = 0\,, \quad \lambda_i - \lambda_j = c_{ij}\,, \quad \text{for all } (i,j) \in T$$

8.5 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 $\bar{c}_{ij} \ge 0$ for all $(i,j) \in L$ and $\bar{c}_{ij} \le 0$ 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: $\lambda_1 = 0$, $\lambda_1 = -3$ and $\lambda_2 = -1$. The value of $c_{23} - \lambda_2 + \lambda_3 = 2 - (-3) + (-1) = 4$. Since this is > 0 we want flow on $(2,3)$ be minimal, which it is. So we now have the optimal solution.



8.6 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.

8.7 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 8.2 (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.

9 Transportation and Assignment Problems

9.1 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

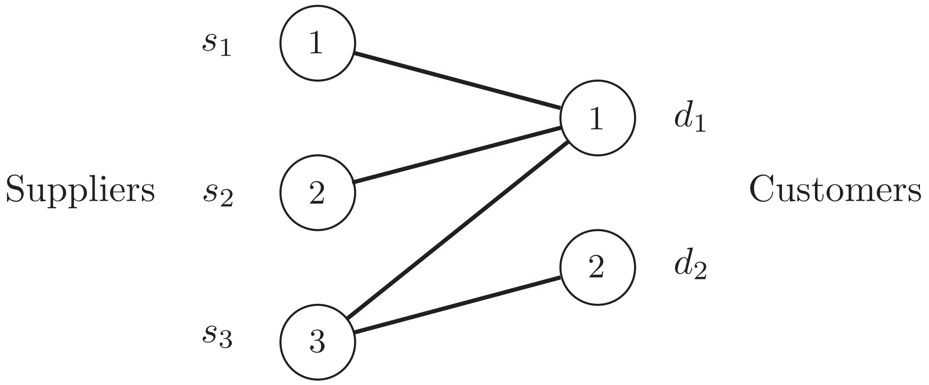
$$\text{minimize} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} f_{ij}$$

subject to

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

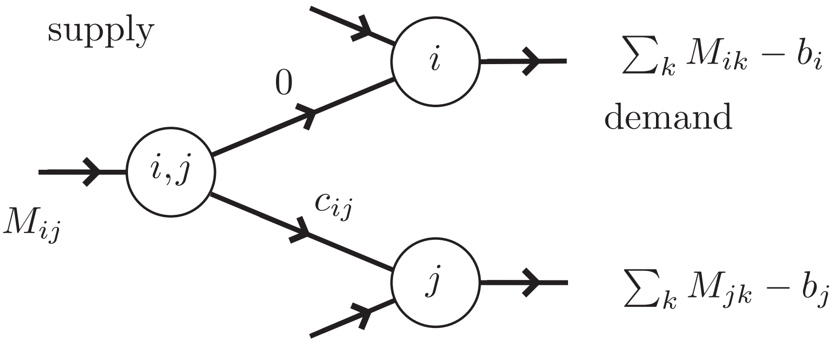
$$f_{ij} \geq 0 \, , \quad \text{for all} \quad i, j \, .$$

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 and $A \subset S \times C$ (the only arcs are those which connect suppliers to consumers).



Lemma 9.1 *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 $\sum_{k\colon (i,k)\in A} M_{ik} - b_i$. 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 $c_{ij,j} = c_{ij}$.

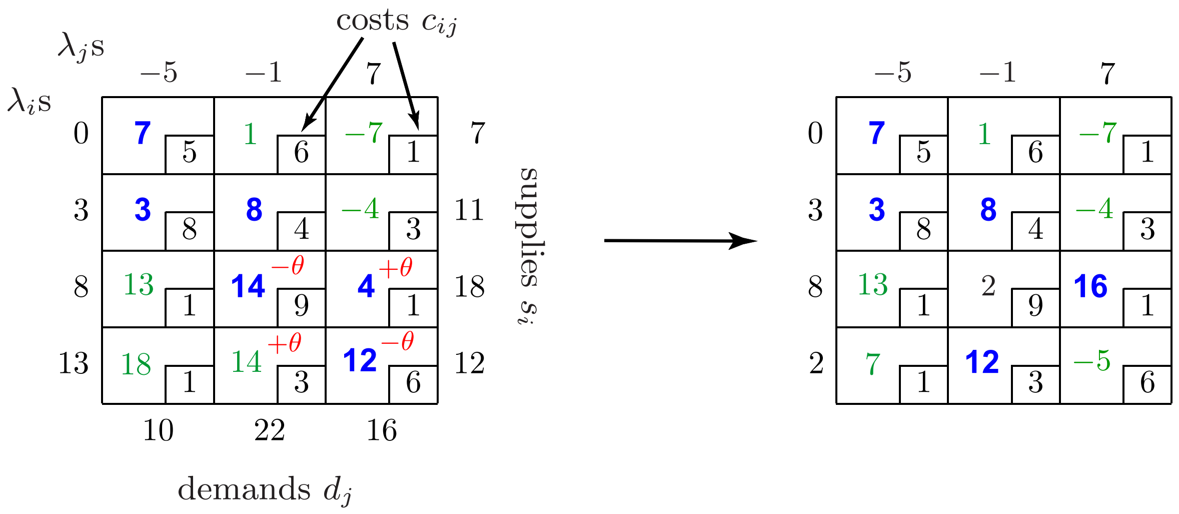


There is a 1-1 correspondence between feasible flows in the two problems and these flows have the same costs. To see this put a flow of f_{ij} on the arc from i, j to j, and a flow of $M_{ij} - f_{ij}$ on the arc from i, j to i. The total amount flowing into node i is then $\sum_j (M_{ij} - f_{ij}) + \sum_j f_{ji}$, 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**.

9.2 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

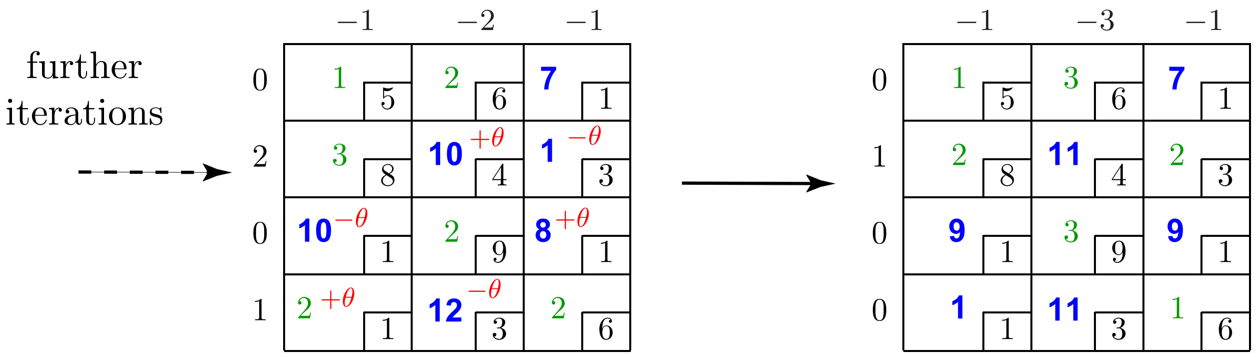

costs c_{ij}
λ_js
λ_is
supplies s_i
demands d_j

λ_is \ λ_js	−5	−1	7	supplies s_i
0	7 [5]	1 [6]	−7 [1]	7
3	3 [8]	8 [4]	−4 [3]	11
8	13 [1]	14 $-\theta$ [9]	4 $+\theta$ [1]	18
13	18 [1]	14 $+\theta$ [3]	12 $-\theta$ [6]	12
demands d_j	10	22	16	

→

	−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 λ_i are computed using the fact that we require $\lambda_i - \lambda_j = c_{ij}$ wherever $f_{ij} > 0$. At the first interation 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 θ around some cycle of cells in which $f_{ij} > 0$.



	-1	-2	-1
0	1 5	2 6	**7** 1
2	3 8	**10** $+\theta$ 4	**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.

9.3 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

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

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

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

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

9.4 Integer constraints

The problem in which the integer constraints are replaced with $0 \le f_{ij} \le 1$ 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.

9.5 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

$$\text{maximize } \delta$$

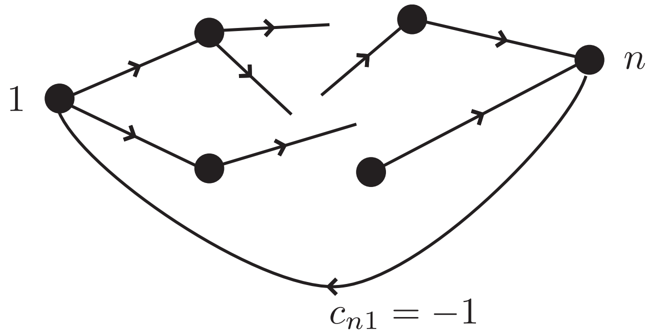
subject to

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

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

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.



10 Maximum Flow and Shortest Path Problems

10.1 Max-flow min-cut theorem

We return to the max-flow problem of Section 9.5. 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 10.1 (Max-flow min-cut theorem)

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

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:

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

Consider the Lagrangian in the usual way with dual variables, λ_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

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

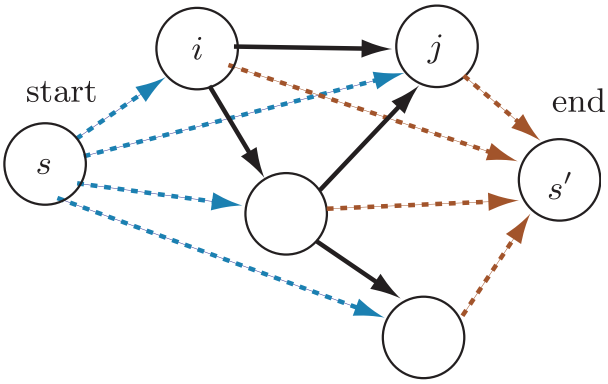
So $\lambda_i = 1$ for $i \in S$ and $\lambda_j = 0$, $j \in N \setminus S$.

10.2 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 τ_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

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



The dual of this problem is

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

subject to

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

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**.

10.3 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 ($b_1 = \cdots = b_{n-1} = 1$), 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**.

10.4 Bellman's equations

Consider the minimum cost flow problem formulation of the shortest path problem. Suppose that the λ_is 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.

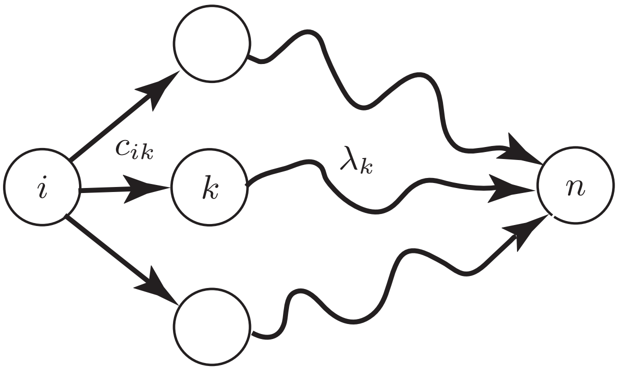
Moreover, as $b_1 = \cdots = b_{n-1} = 1$, the dual problem, with $\lambda_n = 0$, is

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

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

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

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**.

11 Algorithms for Shortest Path Problems

11.1 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

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

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 he 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 $v_i(t+1) = c_{is(i)} + v_{s(i)}(t)$.

11.2 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 11.1 *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 ℓ, 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 ℓ 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
   $c_{24} = \min\{7, 8 + 1\} = 7$.

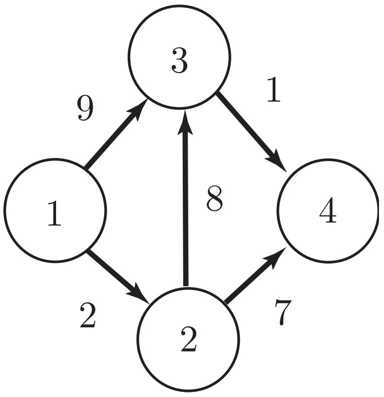
3. Eliminate node $\ell = 3$ from the graph.

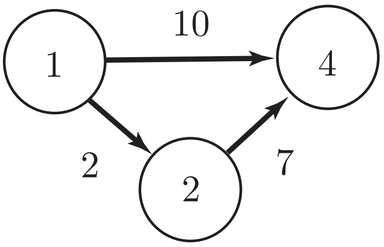
4. Iteration 2 gives $\ell = 2$ and $v_2 = 7$.

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
   $v_1 = c_{14} = 9$.





11.3 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

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

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

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

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.

11.4 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 11.1 (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). ■

11.5 Prim's greedy algorithm for MST

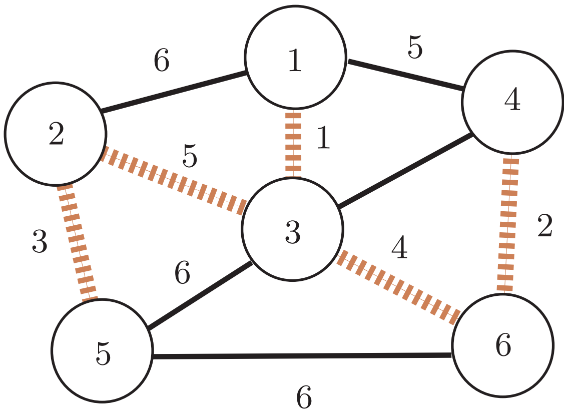
Labels the nodes, $N = \{1, 2, \dots, n\}$ and set $U = \{1\}$. Now construct U recursively using the property above.

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 $\{1, 3\}$, $\{3, 6\}$, $\{6, 4\}$, $\{3, 2\}$, $\{2, 5\}$.



12 Semidefinite Programming

12.1 MAX CUT problem

In the **MAX CUT problem** one is given a graph $G = (V, E)$ and wishes to partition the n vertices into two sets so as to maximize the number of edges crossing from one set to the other. This can be represented as

$$\text{maximize} \sum_{(i,j)\in E} \frac{1 - x_i x_j}{2}, \quad \text{s.t. } x_i \in \{-1, 1\} \text{ for each } i \in N.$$

It is the same as minimizing $x^\top Cx$, where $C_{ij} = 1$ or 0 as (i, j) is or is not an edge of G.

Note that $x^\top Cx = \operatorname{tr}(Cxx^\top)$. We may find an upper bound on the answer by replacing the **positive semidefinite matrix** $xx^\top$ with a general PDS matrix X, written $X \succeq 0$. We then consider the problem

$$\text{minimize } \operatorname{tr}(CX) \text{ s.t. } X \succeq 0 \text{ and } X_{ii} = 1,\ i = 1, \dots, n.$$

The constraint $X_{ii} = 1$ is the relaxation of $x_i x_i = 1 \iff x_i \in \{-1, 1\}$.

This is a special case of a **semidefinite programmming problem**.

12.2 Semidefinite programming problem

Let S_n be the set of symmetric $n \times n$ matrices. Let $C, A_1, \dots, A_n \in S_n$, and $b^\top = (b_1, \dots, b_m) \in \mathbb{R}^m$. Define the **semidefinite programming problem** as

$$\text{SDP}: \text{ minimize } \operatorname{tr}(CX) \text{ s.t. } X \in S_n,\ X \succeq 0 \quad \text{and} \quad \operatorname{tr}(A_i X) = b_i,\ i = 1, \dots, m.$$

(a) LP is a special case of SDP when $C = \operatorname{diag}(c_1, \dots, c_n)$, $A_i = \operatorname{diag}(a_{i1}, \dots, a_{in})$, and $X = \operatorname{diag}(x_1, \dots, x_n)$. SDP is then just 'minimize $c^\top x \ :\ x \geq 0,\ Ax = b$'.

Semidefinite programming has been called *linear programming for the 21st century.*

(b) The Lagrangian dual of SDP is

$$\begin{aligned} \text{DSDP}: \ \underset{y}{\text{maximize}} \min_{X\succeq 0} \operatorname{tr}(CX) + \sum_i y_i(b_i - \operatorname{tr}(A_i X)) \\ = \underset{y}{\text{maximize}}\ y^\top b \quad \text{s.t. } C - \sum_i y_i A_i \succeq 0. \end{aligned}$$

This is because $\operatorname{tr}(C - \sum_i y_i A_i)X) > -\infty$ for all $X \succeq 0$ iff $Z = C - \sum_i y_i A_i \succeq 0$. The complementary slackness condition is $\operatorname{tr}(ZX) = 0$.

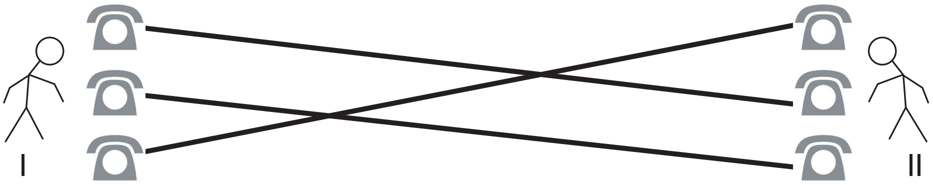
(c) Goemans and Williamson (1995) proved that the true value of MAX CUT is not less than 0.87856 of the value provided via the semidefinite programmign relaxation.

(d) Semidefinite programming problems can be solved efficiently using interior point methods, such as in 12.4 below.

(e) Many other interesting problems can either be formulated as SDPs, an SDPs can help to provide a bound (as with MAX CUT).

12.3 Symmetric rendezvous search game

Suppose that two players are placed in two rooms. Each room has 3 telephones, and these are pairwise connected at random, in a manner unknown to the players. On the stroke of each hour, $1, 2, \dots$, each player picks up one of his telephones and tries to communicate with the other player. They wish to minimize the expected number of attempts required until this occurs. Suppose that at time 1 each player has picked up a telephone at random, and they have failed to connect.



Suppose player I labels the telephone which he picked at time 1 as 'a'. He randomly labels the other two telephones as 'b' and 'c'. His possible pure strategies for time 2 are to pick up a, b or c. Suppose he adopts these with probabilities $x = (x_1, x_2, x_3)$. We assume the players are symmetric (perhaps reading what they should do in these circumstances from the same instruction manual), and so player II must use the same mixed strategy. The probability that the players fail to pick up commected telephones at the next attempt (i.e. fail to rendezvous) is

$$x^\top C_1 x = x^\top \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix} x.$$

The matrix C_1 is positive definite, and so the minimizer is easily found to be $x = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, with $x^\top C_1 x = 2/3$.

Similarly, thinking about both times 2 and 3, there are 9 pure strategies: aa, ab, ac, ba, bb, bc, ca, cb, cc. Suppose the players adopt these with probabilities $x = (x_1, \dots, x_9)$. One can show that the probability that the players have not yet managed to speak after the 3rd attempt is

$$x^\top C_2 x = x^\top \begin{pmatrix} 1 & \frac12 & \frac12 & \frac12 & \frac12 & 0 & \frac12 & 0 & \frac12 \\ \frac12 & 1 & \frac12 & 0 & \frac12 & \frac12 & \frac12 & \frac12 & 0 \\ \frac12 & \frac12 & 1 & \frac12 & 0 & \frac12 & 0 & \frac12 & \frac12 \\ \frac12 & 0 & \frac12 & 1 & \frac12 & \frac12 & \frac12 & \frac12 & 0 \\ \frac12 & \frac12 & 0 & \frac12 & 1 & \frac12 & 0 & \frac12 & \frac12 \\ 0 & \frac12 & \frac12 & \frac12 & \frac12 & 1 & \frac12 & 0 & \frac12 \\ \frac12 & \frac12 & 0 & \frac12 & 0 & \frac12 & 1 & \frac12 & \frac12 \\ 0 & \frac12 & \frac12 & \frac12 & \frac12 & 0 & \frac12 & 1 & \frac12 \\ \frac12 & 0 & \frac12 & 0 & \frac12 & \frac12 & \frac12 & \frac12 & 1 \end{pmatrix} x.$$

C_2 is not positive definite. (It's eigenvalues are $4, 1, 1, 1, 1, 1, 1, -\frac12, -\frac12$.) This means that the quadratic form $x^\top C_2 x$ has local minima. One such is given by $x^\top = (1/9)(1, 1, 1, 1, 1, 1, 1, 1, 1)$, which gives $x^\top C_2 x = 4/9$. But better is $x^\top = (1/3)(1, 0, 0, 0, 0, 1, 0, 1, 0)$, which gives $x^\top C_2 x = 1/3$. How might we prove this is best?

Let J_2 be the 9×9 matrix of 1s. Note that for x to be a vector of probabilities, we must have $x^\top J x = 9$. As with the MIN CUT problem we think of relaxing $xx^\top$ to a PDS matrix X and consider

$$\text{minimize } \operatorname{tr}(C_2 X) \text{ s.t. } X \in S_n,\ X \geq 0,\ X \succeq 0 \text{ and } \operatorname{tr}(J_2 X) = 9.$$

One can numerically compute that the optimal value of this SDP. It is $4/9$. This provides a lower bound on the probablity that the players do not rendezvous by the end of the 3rd attempt. This is achieved by $x^\top = (1/3)(1, 0, 0, 0, 0, 1, 0, 1, 0)$ — so this strategy does indeed minimize the probability that they have not yet met by the end of the 3rd attempt.

These ideas can be extended (Weber, 2008) to show that the expected time to rendezvous is minimized when players adopt a strategy in which they choose their first telephone at random, and if this does not connect them then on successive pairs of subsequent attempts they choose *aa*, *bc* or *cb*, each with probablity 1/3. Given that they fail to meet at the first attempt, the expected number of further attempts required is 5/2. This is less than 3, i.e. the expected number of steps required if players simply try telphones at random at each attempt. There are many simply-stated but unsolved problems in rendezvous search games.

12.4 Interior point methods for LP and SDP

SDPs are solved (numerically) using **interior point methods**. The ellipsoid algorithm is such a method. However, it is not the most effective method. We illustrate the **primal-dual path following method** for the LP and DLP

$$\text{minimize } c^\top x \text{ s.t. } Ax = b, \quad x \geq 0.$$

$$\text{maximize } y^\top b \text{ s.t. } y^\top A + s = c^\top, \quad s \geq 0.$$

Constraints of the form $x_i \geq 0$ and $s_i \geq 0$ are problematic. We drop those, and consider primal and dual objective functions, for $\mu > 0$,

$$c^\top x - \mu \textstyle\sum_i \log x_i \quad \text{and} \quad y^\top b + \mu \textstyle\sum_j \log s_j.$$

The term of $-\mu \log x_i$ provides a so-called **barrier function** which prevents x_i getting close to 0. However, its influence decreases as μ tends to 0. Similarly, for an SDP, the constraint $X \succeq 0$ is handled by the barrier $-\mu \log\det(X)$.

By considering the Lagrangian of

$$L = c^\top x - \mu \sum_i \log x_i + y^\top (b - Ax).$$

we can prove that x, y, s are optimal for the modified primal and dual problems if

$$Ax = b \tag{12.1}$$
$$x \geq 0 \tag{12.2}$$
$$A^\top y + s = c \tag{12.3}$$
$$s \geq 0 \tag{12.4}$$
$$x_i s_i = \mu \quad \text{for all } i. \tag{12.5}$$

Suppose that we have feasible solutions that satisfy (12.1)–(12.4). However, they are not optimal since (12.5) does not hold.

The key idea is to follow a path on which simultaneously we let μ tends 0 and try to ensure (12.5). At each iteration of the algorithm we use Newton's method to take a step $(x, y, s)_{k+1} \to (x, y, s)_k + (d_x, d_y, d_s)_k$ which seeks to solve (12.1), (12.3), (12.5), and is a small enough step that (12.2) and (12.3) are maintained. At iteration k we take $\mu = \mu_k$, where perhaps $\mu_k = (1/2)(s^\top x)_{k-1}/n$.

It is possible to prove that such an algorthm decreases the duality gap μ_k from ϵ_0 to ϵ in a time that is $O(\sqrt{n} \log(\epsilon_0/\epsilon))$.

13 Branch and Bound

13.1 Branch and Bound technique

A **branch and bound** technique can be useful in many problems. 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 ith 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 in 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 a single new 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

(a) $\ell(Y) \geq U$.

(b) Y contains no feasible solutions.

(c) 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.

13.2 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

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

1. 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 $\ell(X) = 1$.

2. 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 $L = \{X_0, X_1\}$.

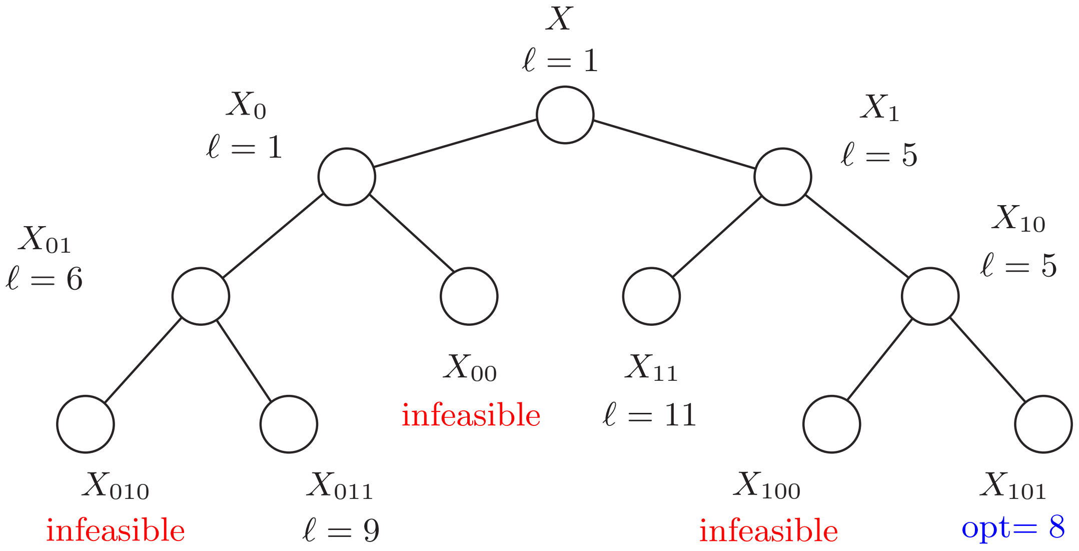
3. Break X_0 into the subproblems X_{01}, X_{00}, such that the backpack does not include item 1, and does or does not include item 2. X_{00} is infeasible and so we eliminate it. For X_{01} we have $\ell(X_{01}) = 6$. Now $L = \{X_{01}, X_1\}$.

4. 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 $L = \{X_{01}, X_{10}\}$.

5. 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 $L = \{X_{01}\}$.

6. Break X_{01} into subproblems X_{011} and X_{010}. Since $\ell(X_{011}) > U$, we eliminate X_{001} by test (a). As for X_{010} it is infeasible, and so elimnated 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. In general, the speed of branch and bound very much depends on the problem and what choices are made about how to bound and fathom nodes of the tree of solutions.

For example, if we are willing to do a bit more work we can compute better bounds. 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.

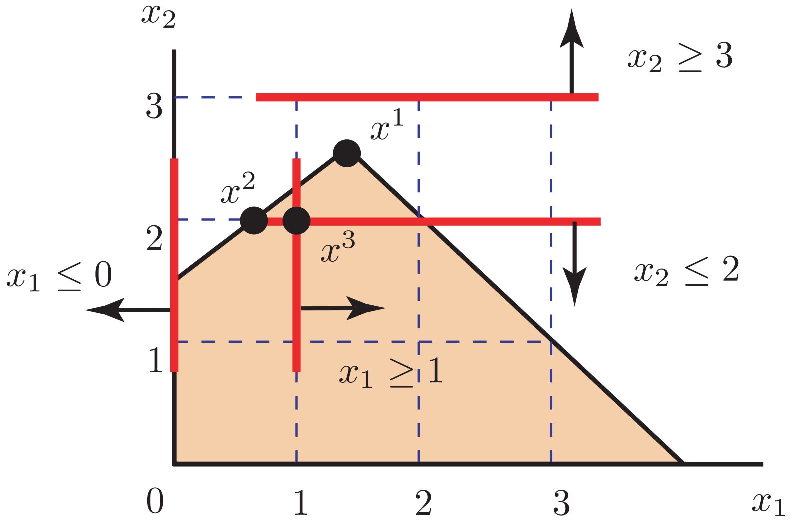
Note 2. Suppose we want all optimal solutions. In this case, we replace the fathom test (a) with $\ell(Y) > U$ and change fathom test (c) so that if $\ell(Y) = U$ we add additional incumbent solutions to the collection, and if $\ell(Y) < U$ we throw away all current incumbent solutions and replace by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

13.3 Dakin's method

Dakin's method is the application of a branch and bound approach to integer linear programming problem. This is called a pure integer LP is all variables constrained to be integers, or mixed integer LP is only a subset of the variables are constrainted to be integers. Let us illustrate it by an example.

$$\text{minimize } x_1 - 2x_2\,,$$
$$\text{subject to } -4x_1 + 6x_2 \le 9\,,\ x_1 + x_2 \le 4\,,\ x_1, x_2 \ge 0\,,\ x_1, x_2 \in \mathbb{Z}\,.$$

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 $x_2 \ge 3$ (subproblem P_1) or $x_2 \le 2$ (subproblem P_2). This is the general approach of Dakin's method: to introduce a partitioning using a constraint of the form $x_j \le \lfloor \widehat{x}_j \rfloor$ or $x_j \ge \lceil \widehat{x}_j \rceil$, where $\widehat{x}$ is the solution to a relaxed problem.
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 $x_1 \ge 1$ (subproblem P_3) or $x_1 \le 0$ (subproblem P_4).
7. The optimal solution to P_3 is $x^3 = (1, 2)$ which is integer and therefore er 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 $-3 \ge U$. So delete P_4.
9. There are no unfathomed subproblems so have optimal solution $x^3 = (1, 2)$.



14 Travelling Salesman Problem

14.1 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 algorithm**s 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.

14.2 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

$$\sum_{(i,j)\in\delta(\{i\})} x_{ij} = 2, \quad i \in N. \tag{14.1}$$

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

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 2, \quad \text{for all } S \subset N, \text{ such that } S \neq \emptyset \text{ or } N. \tag{14.2}$$

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

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

subject to (14.1), (14.2) and $x_{ij} \in \{0,1\}$. Notice that we have exponentially many constraints, since there are $2^n - 2$ constraints of type (14.2).

Alternatively, we can replace (14.2) by

$$\sum_{(i,j)\,:\,i,j\in S} x_{ij} \leq |S| - 1, \quad \text{for all } S \subset N \text{ such that } S \neq \emptyset \text{ or } N\,. \tag{14.3}$$

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).

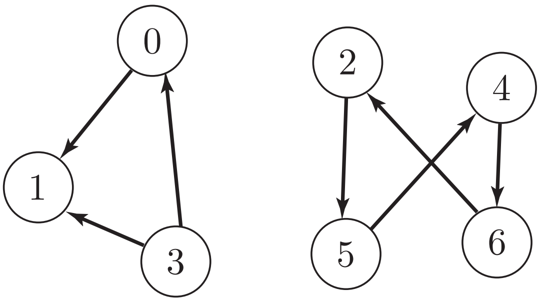
14.3 Polynomial formulation of TSP

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

$$\sum_{j\,:\,(i,j)\in\mathcal{A}} x_{ij} = 1, \quad i = 0,1,\ldots,n-1 \tag{14.4}$$

$$\sum_{i\,:\,(i,j)\in\mathcal{A}} x_{ij} = 1, \quad j = 0,1,\ldots,n-1\,. \tag{14.5}$$

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



Consider a tour $s_0 = 0, s_1, s_2, \ldots, s_{n-1}$. 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,

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

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

$$t_j \geq t_i + 1 - n(1 - x_{ij})\,, \quad i \geq 0,\ j \geq 1,\ i \neq j\,. \tag{14.6}$$

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 (14.4), (14.5), (14.6), $x_{ij} \in \{0,1\}$, $t_0 = 0$ and $t_i \in \{0,1,2,\ldots\}$.

14.4 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

$$\text{minimize } \sum_{i,j} c_{ij} x_{ij}\,, \quad \text{subject to (14.4), (14.5) and } x_{ij} \in \{0,1\}.$$

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.

14.5 Approximation algorithm for the TSP

Definition 14.1 *An* ***ϵ-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

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

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

15 Heuristic Algorithms

15.1 Heuristics for the TSP

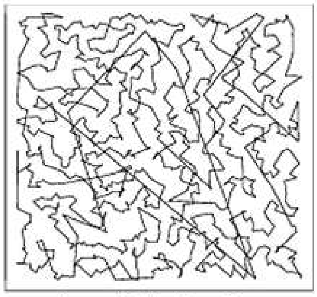
Nearest neighbour heuristic. Start at some city and then visit the nearest city. Continue to visit the nearest city that has not yet been visited, continuing until a tour is complete.

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

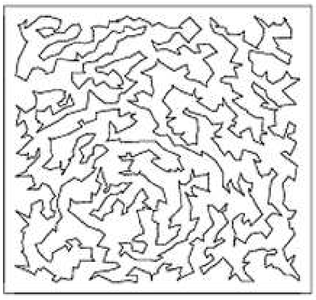
Cheapest insertion heuristic. This is also a greedy algorithm. Start with a single node and then, one by one, add the node whose insertion makes the smallest increase to the length of the tour.

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

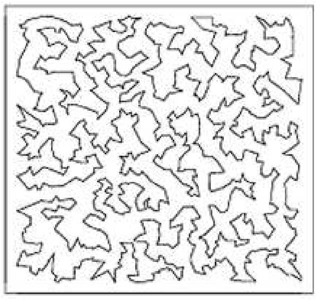
Savings heuristic. Rank the arcs in ascending order of cost. Add the arcs in this order, so long as they do not violate any constraints, and until all cities have been visited.



Nearest Neighbor Tour



Savings Tour



Optimal Tour

15.2 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.

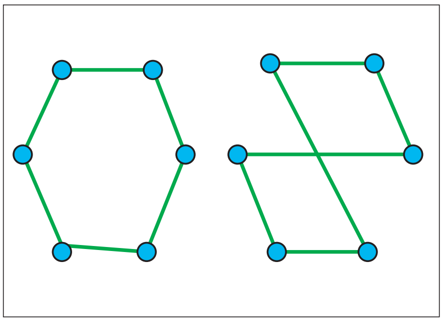
3. The simplex algorithm for linear programs is a local search method. We can say that two basic feasible solutions are neighbours if they are connected by an edge of the constraint set.

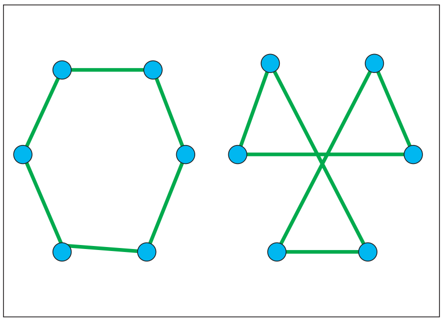
   In linear programming any local optimum is the global optimum.

15.3 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)	AEBCD (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.

15.4 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

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

As motivation for this claim, imagine that all solutions have k neighbours, and at each step of the algorithm one of these neigbours is chosen at random. Then x moves next to its neightbour 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 $\pi(Y)/(1-\pi(Y)) \to \infty$ as $T \to 0$.

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

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

15.5 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, analagous 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.

16 Two-person Zero-sum Games

16.1 Terminology

Game theory studies multi-player decision problems. The common theme is conflicts of interest between the players. We assume that each player plays the 'best way' he can. This may not happen in practice. The elements of a game are as follows.

- **Players.** Labelled $1, 2, 3, \ldots$, 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.)

16.2 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 $\mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_n$ and player II has strategies $\mathrm{II}_1, \mathrm{II}_2, \ldots, \mathrm{II}_m$.

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.

16.3 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, so 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)

$$\begin{array}{cc}
 & \begin{array}{cccc} \mathrm{II}_1 & \mathrm{II}_2 & \mathrm{II}_3 & \mathrm{II}_4 \end{array} & \text{min using } \mathrm{I}_i \\
\begin{array}{c} \mathrm{I}_1 \\ \mathrm{I}_2 \\ \mathrm{I}_3 \end{array} & \begin{pmatrix} 4 & 5 & 6 & 4 \\ 4 & 2 & 3 & 4 \\ 2 & 4 & 5 & 3 \end{pmatrix} & \begin{array}{c} 4 \\ 2 \\ 2 \end{array} \\
\text{max using } \mathrm{II}_j & \begin{array}{cccc} 4 & 5 & 6 & 4 \end{array} &
\end{array}$$

Thus, $v_L = \max\{4,2,2\} = 4$ and $v_U = \min\{4,5,6,4\} = 4$. 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**.

Sometimes we can simplify a game by making use of the idea of **dominating strategies**. Notice that 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

$$\begin{array}{cc}
 & \begin{array}{cc} \mathrm{II}_1 & \mathrm{II}_2 \end{array} \\
\begin{array}{c} \mathrm{I}_1 \\ \mathrm{I}_2 \\ \mathrm{I}_3 \end{array} & \begin{pmatrix} 4 & 5 \\ 4 & 2 \\ 2 & 4 \end{pmatrix}
\end{array}$$

and then, as I_1 dominates both I_2 and I_3, to

$$\begin{array}{cc}
 & \begin{array}{cc} \mathrm{II}_1 & \mathrm{II}_2 \end{array} \\
\mathrm{I}_1 & \begin{pmatrix} 4 & 5 \end{pmatrix}
\end{array}$$

and so there is a saddle point equilibrium at $(\mathrm{I}_1, \mathrm{II}_1)$.

16.4 Mixed strategies

Example (b)

$$\begin{array}{cccc} & \mathrm{II}_1 & \mathrm{II}_2 & \text{min using } \mathrm{I}_i \\ \mathrm{I}_1 & 2 & 0 & 0 \\ \mathrm{I}_2 & 1 & 2 & 1 \\ \text{max using } \mathrm{II}_j & 2 & 2 & \end{array}$$

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

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

$$e(p, q) = 2p_1q_1 + 0p_1q_2 + 1p_2q_1 + 2p_2q_2 = 3(p_1 - \tfrac{1}{3})(q_1 - \tfrac{2}{3}) + \tfrac{4}{3}.$$

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

$$e(p^*, q) = 4/3 \quad \text{for all } q\,, \quad \text{and} \quad e(p, q^*) = 4/3 \quad \text{for all } p\,.$$

It is clear that $v_L^M \le v_U^M$. Playing p^* guarantees I at least 4/3, so $v_L^M \ge 4/3$. If II plays q^* then I can obtain no more than 4/3, so $v_U^M \le 4/3$. Hence, we must have

$$v_L^M = v_U^M\,.$$

16.5 Minimax theorem

Theorem 16.1 (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.

16.6 Equilibrium pairs

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

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

It is possible for there to be more than one equilibrium pair. In Example (a) both $(\mathrm{I}_1, \mathrm{II}_1)$ and $(\mathrm{I}_1, \mathrm{II}_4)$ are equilibrium pairs. Indeed, $p^* = (1, 0)$ and $q^* = (q, 0, 1-q)$ are equilibrium pairs for any $q \in [0, 1]$. But in all cases $e(p^*, q^*) = 4$.

Lemma 16.1 *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

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

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

Theorem 16.2 *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

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

Then

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

So, since $v_L^M \le v_U^M$ 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

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

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

17 Solution of Two-person Games

17.1 LP formulation of a zero-sum game

We can solve a two-person zero-sum game using linear programming. We want to find probability vectors p^*, q^* and a v such that

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

for any p and q. The first inequality implies that

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

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

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

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

$$\text{maximize } \; Q_1 + Q_2 + \cdots + Q_m$$

$$\text{subject to } \sum_{j=1}^{m} e_{ij} Q_j \le 1\,, \quad i = 1, \ldots, n\,, \text{ and } Q_j \ge 0\,, \quad j = 1, \ldots m\,.$$

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

$$\text{minimize}\, P_1 + P_2 + \cdots + P_n$$

$$\text{subject to } \sum_{i=1}^{n} P_i e_{ij} \ge 1\,, \quad j = 1, 2, \ldots, m\,, \quad \text{and } P_i \ge 0\,, \quad i = 1, 2, \ldots, n\,.$$

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

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

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^*.

17.2 Two-person non-zero-sum games

The players are not completely antagonistic and might both be happier with one outcome than another. In a two-person non-zero-sum game (also called a **bimatrix game**, (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

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

Example: Prisoner's Dilemma

$$\begin{array}{c} \\ \text{Don't confess} \\ \text{Confess} \end{array} \begin{array}{cc} \text{Don't confess} & \text{Confess} \\ \left(\begin{array}{c} (\ 5,5) \\ (10,0) \end{array} \right. & \left. \begin{array}{c} (0,10) \\ (\ 1,1) \end{array} \right) \end{array}$$

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

$$\begin{array}{c} \\ \text{Opera} \\ \text{Football} \end{array} \begin{array}{cc} \text{Opera} & \text{Football} \\ \left(\begin{array}{c} (1,4) \\ (0,0) \end{array} \right. & \left. \begin{array}{c} (0,0) \\ (4,1) \end{array} \right) \end{array}$$

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**.

17.3 Nash's theorem

Theorem 17.1 (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

$$c_i(p, q) = \max\{0, e_1(\mathrm{I}_i, q) - e_1(p, q)\}, \quad i = 1, 2, \ldots, n$$

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

$$d_j(p, q) = \max\{0, e_2(p, \mathrm{II}_j) - e_2(p, q)\}, \quad j = 1, 2, \ldots, m$$

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

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

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

for all $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$. Then f is continuous so, by Brouwer's 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 such that $p^*_i > 0$, 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 for which $p^*_i > 0$ 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 $i' = 1, 2, \ldots, n$ we have that $c_{i'}(p^*, q^*) = 0$ and so

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

and hence

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

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. ■

17.4 Finding an equilibrium pair

Example. Consider the bimatrix game

$$\begin{array}{ccc} & \mathrm{II}_1 & \mathrm{II}_2 \\ \mathrm{I}_1 & (3,2) & (2,1) \\ \mathrm{I}_2 & (0,3) & (4,4) \end{array}$$

Suppose we fix q and find p to maximizes $e_1(p,q)$. If q is part of an equilibrium pair we have constructed it's 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

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

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

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

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 $(\mathrm{I}_2, \mathrm{II}_2)$ with payoffs $(4,4)$;
2. $p_1 = 1, q_1 = 1$, corresponding to $(\mathrm{I}_1, \mathrm{II}_1)$ 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.

18 Construction of a Nash Equilibrium

18.1 Symmetric games

In general, it is a difficult problem to find all equilibrium points. Indeed it is an NP-complete problem to decide if more than one Nash equilibria exists, if there exists an equilibrium in which each player mixes at least two pure strategies, or if there is an equlibrium in which the sum of I and II's expected payoffs can be at least some given value. We now look at a method for computing a Nash equilibrium. We begin with the special case of a symmetric game.

A **symmetric game** is one such that $e_1(i,j) = a_{ij}$, $e_2(i,j) = a_{ji}$. E.g.

$$A = \begin{pmatrix} 0 & 3 & 0 \\ 0 & 0 & 3 \\ 2 & 2 & 2 \end{pmatrix} \text{ gives the non-zero sum game: } \begin{pmatrix} (0,0) & (3,0) & (0,2) \\ (0,3) & (0,0) & (3,2) \\ (2,0) & (2,3) & (2,2) \end{pmatrix}$$

Lemma 18.1 *Suppose $x, z \geq 0$, $Ax + z = 1$, $x^\top z = 0$, and $1^\top x > 0$. Let $\bar{p} = x/1^\top x$. Then $(\bar{p}, \bar{p})$ is a (symmetric) equilibrium pair.*

Proof. Let p be any (mixed) strategy.

(i) $Ax \leq 1 \implies p^\top Ax \leq 1 \implies p^\top A\bar{p} \leq (1^\top x)^{-1}$, and $\bar{p}^\top A^\top p \leq (1^\top x)^{-1}$.

(ii) $x^\top z = x^\top(1 - Ax) = 0 \implies \bar{p}^\top A\bar{p} = (1^\top x)^{-1}$. ■

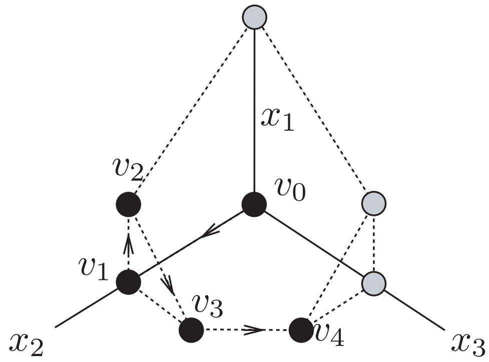
18.2 Lemke-Howson algorithm

The basic feasible solution $v_0 = (x,z) = (0,1)$ satisfies all conditions of the lemma except for $1^\top x > 0$. Let us say that strategy i is *represented* if $x_i z_i = 0$ and *twice-represented* if $x_i = z_i = 0$. It is *missing* if $x_i z_i > 0$.

Let $P = \{(x,z) : Ax + z = 1,\, x \geq 0,\, z \geq 0\}$. Let us assume no degeneracy, so that every vertex of P has exactly n neighbours. Fix a strategy, say i, and consider the set V, consisting of all vertices of P in which every strategy is represented or twice-represented, except possibly strategy i (which may be missing). Note that $v_0 \in V$ since in v_0 every strategy is represented. There is exactly one neighbouring vertex to v_0 that is also in V.

In the example above, $v_0 = (x,z) = (0,0,0,1,1,1)$. If we choose $i = 2$ then the unique neighbouring member of V which has strategy 2 missing is obtained by increasing x_2, to reach $v_1 = (0, \frac{1}{3}, 0, 0, 1, \frac{1}{3})$. In moving from v_0 to v_1, the variable z_1 has decreased to 0 and strategy 1 is now twice-represented. So let us now increase x_1 and move to $v_2 = (\frac{1}{6}, \frac{1}{3}, 0, 0, 1, 0)$. Now z_3 has been decreased to 0 and strategy 3 is twice-represented, so we increase x_3 to reach $v_3 = (0, \frac{1}{3}, \frac{1}{6}, 0, \frac{1}{2}, 0)$. Now x_1 has decreased to 0, strategy 1 is again twice-represented, so we increase z_1, to reach $v_4 = (0, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, 0, 0)$. Here, z_2 is decreased to 0, all strategies are represented, and the conditions of the lemma are satisfied. We have the equilibrium $\bar{p} = (0, \frac{1}{3}, \frac{2}{3})$.

As we move amongst vertices in V we are at each step increasing some variable x_i (or z_i) associated with an twice-represented strategy i, which is complementary to the variable z_i or (or x_i) that was decreased to 0 at the previous step.



	x_1	x_2	x_3	z_1	z_2	z_3
v_0 :	0	0	0	1	1	1
v_1 :	**0**	$\frac{1}{3}$	0	**0**	1	$\frac{1}{3}$
v_2 :	$\frac{1}{6}$	$\frac{1}{3}$	**0**	0	1	**0**
v_3 :	**0**	$\frac{1}{3}$	$\frac{1}{6}$	**0**	$\frac{1}{2}$	0
v_4 :	0	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{2}$	0	0

This algorithm (**Lemke-Howson**) always works! The reasoning is cute. Firstly, note that since P is bounded it must be the case that as we increase a nonbasic variable from 0, some basic variable must decrease and eventually reach 0, completing a pivot step from one vertex to another. Second, there is only one way to leave v_0 and remain in V; similarly, having reached v_i there is only one way to leave v_i and remain in V without returning to the vertex from which the path just arrived. Thus, the path never revisits a vertex. Thirdly, there are a finite number of vertices in V, so eventually the algorithm can go no further. The only way this can happen is to have reached a vertex at which no strategy is twice-represented. Since this is not v_0 it must be one for which $1^\top x > 0$.

18.3 Bimatrix games

Now consider the more general two-person zero sum game in which $e_1(i,j) = a_{ij}$ and $e_2(i,j) = b_{ij}$. Players I and II have n and m pure strategies, respectively. Matrices $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times m$. Similarly to Lemma 18.1, an equilibrium pair can be constructed from nonnegative and nonzero vectors x, z, y and w such that $x^\top B + z^\top = 1^\top$ and $Ay + w = 1$, for which $z^\top y = x^\top w = 0$.

We can also take a general bimatrix game and construct from it a symmetric game. This game has $n + m$ pure strategies and payoff matrices

$$\bar{A} = \begin{pmatrix} 0 & A \\ B^\top & 0 \end{pmatrix} \quad \bar{B} = \bar{A}^\top = \begin{pmatrix} 0 & B \\ A^\top & 0 \end{pmatrix}.$$

Suppose we can find $s = (x, y)$ such that $s \geq 0$, $\bar{A}s \leq 1$, $s^\top(1 - \bar{A}s) = 0$,

$1^\top x > 0$, and $1^\top y > 0$, then $s/1^\top s$ is a symmetric equilibrium for this symmetric game, and $\bar{p} = x/1^\top x$ and $\bar{q} = y/1^\top y$ provide an equilibrium pair for the original nonsymmetric game.

The above shows how to use the Lemke-Howson algorithm to find a Nash equilibrium of the bimatrix game with (A, B). We say that Player I's strategy i is twice-represented if $x_i = w_i = 0$ and Player II's strategy j twice-represented if $y_j = z_j = 0$.

Let $P = \{x : x^\top B \leq 1_m^\top,\ x \geq 0\}$ and $Q = \{y : Ay \leq 1_n,\ y \geq 0\}$. We start at $(x, z, y, w) = (0, 1, 0, 1)$ and arbitrarily pick one of the pure strategies, say Player I's first strategy. We increase x_1 from 0. If this makes $z_i = 0$ then Player II's ith strategy is now twice-represented, and the next step is to increase y_i from 0. We continue in this manner, alternately increasing from 0 a component of (x, z) and one of (w, y), until eventually reaching a vertex at which the missing strategy becomes represented, no strategy is twice-represented and both x and y are nonzero. There is an interesting corollary of this analysis.

Corollary 18.1 *A nondegenerate bimatrix game has an odd number of Nash equilibria.*

Proof. Let V be the set of vertices in which only Player I's first strategy might be missing (i.e. such that $x_1 w_1 > 0$). Every equilibrium of $P \times Q$ is a member of V (since equilibriums are vertices for which all strategies are represented). In the graph formed by vertices in V, each vertex has degree 1 or 2. So this graph consists of disjoint paths and cycles. The endpoints of the paths are the Nash equilibriums and the special vertex $(x, y) = (0, 0)$. There are an even number of endpoints, so the number of Nash equilibria must be odd. ■

Notice that we can write the conditions for an equilibrium as

$$\begin{pmatrix} w \\ z \end{pmatrix} - \begin{pmatrix} 0 & -A \\ -B^\top & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1_n \\ 1_m \end{pmatrix}, \quad \begin{pmatrix} x \\ y \end{pmatrix} \geq 0, \quad \begin{pmatrix} w \\ z \end{pmatrix} \geq 0, \quad \begin{pmatrix} x \\ y \end{pmatrix}^\top \begin{pmatrix} w \\ z \end{pmatrix} = 0.$$

This is an example of a more general problem.

18.4 Linear complementarity problem

The **linear complementarity problem** (LCP) unifies linear programing, quadratic programing and two-person non-zero sum games (**bimatrix games**). Let M be a $L \times L$ square matrix, and q a L-vector. LCP is to find two L-vectors, z and w, such that

$$w - Mz = q, \quad z, w \geq 0 \quad \text{and } z^\top w = 0.$$

Note that nothing is to be maximized or minimized in this problem.

18.5 Linear programming as a LCP

Consider the primal and dual linear programs

$$\begin{aligned} P: &\quad \text{minimize}\,\{c^\top x \,:\, Ax - v = b,\ x \ge 0,\ v \ge 0\} \\ D: &\quad \text{maximize}\,\{b^\top y \,:\, A^\top y + u = c,\ y \ge 0,\ u \ge 0\}\,. \end{aligned}$$

Recall that (x, v) and (y, u) are optimal solutions if and only if

$$x, y, v, u \ge 0, \quad \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} 0 & -A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c \\ -b \end{pmatrix}, \quad x^\top u = 0, \quad y^\top v = 0.$$

This is a LCP with the obvious choices of w, M, z and q.

18.6 Lemke's algorithm

If $q > 0$ then a solution to the LCP is $w = q$ and $z = 0$. So suppose $q \not> 0$. Let $d = -(1, 1, \dots, 1)^\top$ and consider

$$w - Mz + \lambda d = q.$$

For $\lambda > 0$ large enough, we have the solution $w = q - \lambda d \ge 0$, and $z = 0$. Imagine decreasing λ from a large value until some component of w becomes 0. At this point (where $\lambda = -\min_i q_i = -q_k$) we have $w_k = z_k = 0$, and there will be complementary slackness, i.e. $w_i z_i = 0$ for all i.

Our next step is to move to a neighbouring basic feasible solution by increasing z_k (i.e. we perform a pivot step that puts z_k into the basis). Suppose that as we increase z_k one of the existing basic variables decreases, eventually reaches 0 and so must leave the basis. (If this does not happen then the algorithm fails. But there are special cases for which this cannot happen, such as the bimatrix games.)

- If the departing basic variable is λ then we now have a solution to the LCP.
- If the departing basic variable is not λ then we are left with a new bfs in which $w_\ell = z_\ell = 0$ for some ℓ.

We continue in this fashion, moving amongst solutions that are always complementary slack, i.e. $w_i z_i = 0$ for all i, and such that $\lambda > 0$ and $w_\ell = z_\ell = 0$ for some ℓ. After each pivot, we note which which of the variables it was, w_ℓ or z_ℓ, that has just left the basis, and then increase the other one — continuing in this fashion until eventually it is λ that leaves the basis. At that point we have a solution to the LCP. Note that there are only a finite number of complementary bases and they do not repeat. Each bfs that we can reach has exactly two neighbouring bfs, one of which precedes it and one of which follows it. Thus no bfs can be reached more than once and so λ must eventually leave the basis.

19 Cooperative Games

19.1 Quadratic programming as a LCP

Let D be a positive definite symmetric matrix and consider

$$\begin{aligned} \text{QP}: \quad & \text{minimize } Q(x) = c^\top x + \tfrac{1}{2} x^\top D x \\ & \text{subject to } Ax \geq b, \quad x \geq 0. \end{aligned}$$

Consider minimizing over nonnegative x and v, the Lagrangian

$$L = c^\top x + \tfrac{1}{2} x^\top D x + \bar{y}^\top (b - Ax + v).$$

From the Lagrangian sufficiency theorem we see that (x, v) minimizes L if

$$b - A\bar{x} = \bar{v}, \quad \frac{\partial}{\partial x} L = c + Dx - A^\top \bar{y} = \bar{u},$$

$$\bar{y}^\top \bar{v} = 0, \quad x^\top \bar{u} = 0, \quad x,\, v,\, \bar{y},\, \bar{u} \geq 0.$$

So $\bar{x}$ is an optimal solution to QP if there exist vectors $\bar{y}$, $\bar{u}$ and $\bar{v}$ such that

$$\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} - \begin{pmatrix} D & -A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} = \begin{pmatrix} c \\ -b \end{pmatrix}$$

$$\begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} \geq 0 \quad \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} \geq 0 \quad \text{and} \quad \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}^\top \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} = 0.$$

This defines a LCP, whose solution is an optimal solution to QP.

19.2 Cooperative games

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 it is used to average between possible outcomes. Consider the game

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

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 β 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.

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 payuoff such that both can do as well and one can do better.

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

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

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

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

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

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

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

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

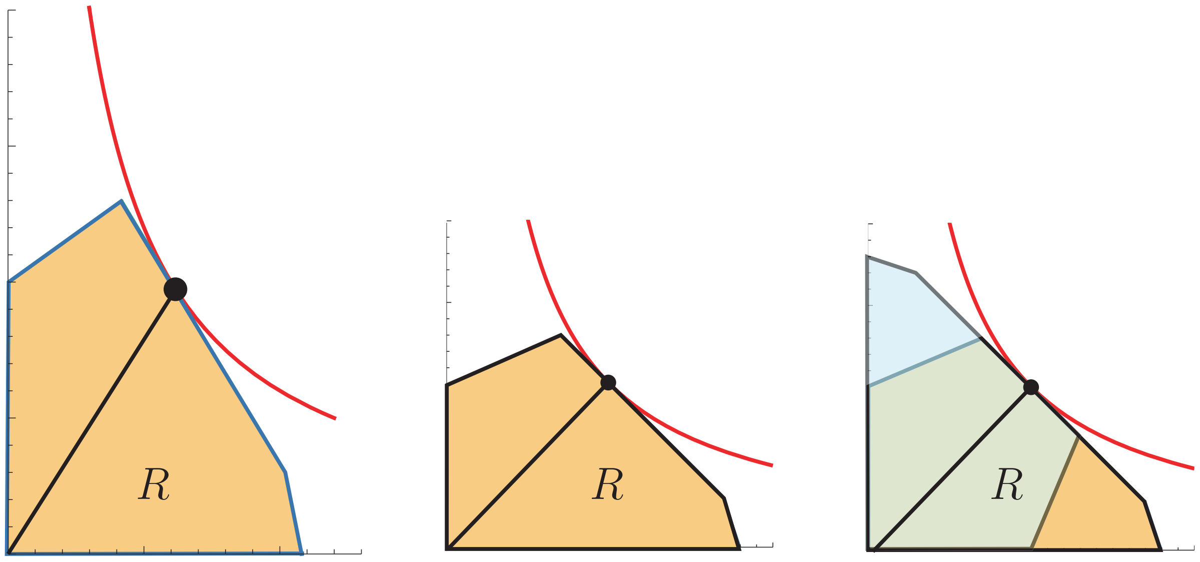
If there exists points $(u, v) \in R$ with $u > u_0$, $v > v_0$ then f attains a unique maximum at some point $(u^*, v^*) \in R$. Define

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

Nash showed that Ψ is the only function that satisfies the axioms (1–6).

"Proof"

Assume $u_0 = v_0 = 0$. Touch set R with $uv =$ constant; use axiom 5 to move the point of touching to $(1, 1)$; add the set found by reflection of R around 45 degrees; use 4 to argue $(1, 1)$ must be the solution; then use 6 to remove the extra set that was just added.



19.6 Maximin bargaining solutions

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

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

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

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

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

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

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

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 their relative bargaining strengths.

Label the players $1, 2, \ldots, n$. A **coalition** of players, S, is then a subset of $N = \{1, 2, \ldots, n\}$. 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 the 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 $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ satisfying

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

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

$$(3)\ y_i > x_i \text{ for all } i \in S\,; \quad \text{and} \quad (4)\ \sum_{i \in S} y_i \le v(S)\,.$$

(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*

$$(5) \quad \sum_{i=1}^{n} x_i = v(N)\,; \qquad (6) \quad \sum_{i \in S} x_i \ge v(S) \quad \textit{for all } S \subset N\,.$$

Proof. Let x satisfy (5) and (6). Putting $S = \{i\}$ for each $i = 1, \ldots, n$ 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)

$$\sum_{i \in S} y_i > \sum_{i \in S} x_i \ge v(S)\,,$$

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 ϵ by

$$\epsilon = \frac{v(S) - \sum_{i \in S} x_i}{|S|} > 0$$

Then the imputation

$$y_i = \begin{cases} x_i + \epsilon & i \in S \\ v(\{i\}) + \dfrac{v(N) - v(S) - \sum_{i \notin S} v(\{i\})}{|N \setminus S|} & i \notin S \end{cases}$$

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 £a per barrel. Country 2 wants to buy oil for use in farming, for a profit of £b per barrel. Country 3 wants to buy oil for use in manufacturing, for a profit of £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 £x per barrel, which is at least £b per barrel (otherwise, country 1 would be better off selling to country 2) and at most £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 $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in E(v)$ 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 $\theta(\mathbf{x}) = (\theta(\mathbf{x})_1, \theta(\mathbf{x})_2, \ldots, \theta(\mathbf{x})_{2^n})$ 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 $k = 1, 2, \ldots, i-1$ and $\theta(\mathbf{x})_i < \theta(\mathbf{y})_i$.

The **nucleolus** is the smallest imputation under this ordering, written

$$N(v) = \{\mathbf{x} \in E(v)\,;\, \theta(\mathbf{x}) < \theta(\mathbf{y}) \quad \text{for all } \mathbf{y} \in E(v)\}\ .$$

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, $C(v) = \{(x, 0, c-x) \,:\, b \le x \le c\}$.

Computing $v(S) - x(S)$ for all possible coalitions gives

$$\begin{array}{ll}
v(\emptyset) - x(\emptyset) = 0 & v(\{1,2\}) - x(\{1,2\}) = b - x \\
v(\{1\}) - x(\{1\}) = a - x & v(\{2,3\}) - x(\{2,3\}) = x - c \\
v(\{2\}) - x(\{2\}) = 0 & v(\{1,3\}) - x(\{1,3\}) = 0 \\
v(\{3\}) - x(\{3\}) = x - c & v(\{1,2,3\}) - x(\{1,2,3\}) = 0
\end{array}$$

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

$$\theta(\mathbf{x}) = \left(0, 0, 0, 0, \tfrac{1}{2}(b-c), \tfrac{1}{2}(b-c), \tfrac{1}{2}(b-c), a - \tfrac{1}{2}(b+c)\right).$$

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, $1, 2, \ldots, n$.

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***

$$\phi_i(v) = \sum_{S : i \in S} \frac{(|S| - 1)!(n - |S|)!}{n!} \left[v(S) - v(S \setminus \{i\}) \right]\,.$$

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

$$\phi_1(v) = \tfrac{1}{2}c + \tfrac{1}{3}a - \tfrac{1}{6}b\,, \quad \phi_2(v) = \tfrac{1}{6}b - \tfrac{1}{6}a\,, \quad \phi_3(v) = \tfrac{1}{2}c - \tfrac{1}{6}a - \tfrac{1}{3}b\,.$$

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

$$u_i(a,b)\,, \qquad i = 1,2,\ldots,M+N\,.$$

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,

$$u_i\left(\lambda(a_1,b_1)+(1-\lambda)(a_2,b_2)\right) \geq \lambda u_i\left(a_1,b_1\right)+(1-\lambda)u_i\left(a_2,b_2\right)$$

for $0 \leq \lambda \leq 1$ and $i = 1,2,\ldots,M+N$.

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,

$$v(S) = \max_{x_1,\ldots,x_{s_1+s_2},y_1,\ldots,y_{s_1+s_2}} \sum_{i=1}^{s_1+s_2} u(x_i,y_i)$$

where

$$\sum_{i=1}^{s_1+s_2} x_i = s_1 a\,; \qquad \sum_{i=1}^{s_1+s_2} y_i = s_2 b\,.$$

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

$$\mathbf{x}^* = \left((N+1)u\left(\frac{a}{N+1}, \frac{Nb}{N+1}\right) - Nu(0,b), u(0,b), \ldots, u(0,b)\right)$$

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

A market of two competing firms is known as **duopoly**; a market 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

$$u(p_1, p_2, \ldots, p_M, q_1, q_2, \ldots, q_M)$$

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 $p_1, p_2, \ldots, p_M$, so that, say,

$$q_i = f_i(p_1, p_2, \ldots, p_M) .$$

Firm i's utility is given by its profit

$$e_i(p_1, p_2, \ldots, p_M) = p_i q_i - c_i(q_i)$$

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 $(p_1^c, p_2^c, \ldots, p_M^c)$ such that

$$e_i(p_1^c, p_2^c, \ldots, p_M^c) = \max_{p_i} e_i(p_1^c, p_2^c, \ldots, p_i, \ldots, p_M^c)$$

for all firms $i = 1, \ldots, M$. 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

$$\begin{aligned} q_1 &= f_1(p_1, p_2) = \max\left\{1 + \tfrac{1}{3}p_2 - \tfrac{1}{2}p_1, 0\right\} \\ q_2 &= f_2(p_1, p_2) = \max\left\{1 + \tfrac{1}{4}p_1 - \tfrac{1}{2}p_2, 0\right\} \end{aligned}$$

and suppose, for simplicity, that $c_1(q_1) = c_2(q_2) = 0$.

We have that $0 \le p_1 \le 2 + \frac{2}{3}p_2$ and $0 \le p_2 \le 2 + \frac{1}{2}p_1$. The profit functions are then given by

$$\begin{aligned} e_1(p_1, p_2) &= p_1 + \tfrac{1}{3}p_1p_2 - \tfrac{1}{2}p_1^2 \\ e_2(p_1, p_2) &= p_2 + \tfrac{1}{4}p_1p_2 - \tfrac{1}{2}p_2^2 \,. \end{aligned}$$

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

$$\frac{de_1(p_1, p_2^c)}{dp_1} = 1 + \tfrac{1}{3}p_2^c - p_1^c = 0 \,, \qquad \frac{de_2(p_1^c, p_2)}{dp_2} = 1 + \tfrac{1}{4}p_1^c - p_2^c = 0 \,,$$

which gives the Cournot equilibrium as $p_1^c = \frac{16}{11}$, $p_2^c = \frac{15}{11}$, and so

$$e_1(\tfrac{16}{11}, \tfrac{15}{11}) = 1.06 \,; \qquad e_2(\tfrac{16}{11}, \tfrac{15}{11}) = 0.93 \,.$$

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,

$$p_1 = g_1(p_2) = 1 + \tfrac{1}{3}p_2 \,, \qquad p_2 = g_2(p_1) = 1 + \tfrac{1}{4}p_1 \,.$$

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 Auctions

22.1 Types of auctions

An auction is a type of multi-player partial-information game. Its rules 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. It is a **partial information game** because each bidder's valuation of an item is hidden from the auctioneer and other bidders. The 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 said to be **economically efficient**, in terms of maximizing social welfare, if it allocates the item to the bidder who values it most. Auction design is an art. There is no one auctioning mechanism that is efficient and can be applied in most situations.

Government contracts are often awarded through procurement auctions. They are used to sell oil drilling rights, or other natural resources, such as mobile telephone spectrum or satellite positions. 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).

In the **private values** model each bidder knows the value he places on the item, but does not know how it is valued 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 sealed-bids in writing). Popular types of auction are:

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).

22.2 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?

Let us begin with an intuitive result.

Lemma 22.1 *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

$$\pi^*(v) = \max_p [pv - e(p)] = p^*v - e(p^*) , \quad \left.\frac{\partial \pi}{\partial p}\right|_{p=p^*} = v - e'(p^*) = 0. \tag{22.1}$$

Note that $e(p)$ is convex in v, and so at the stationary point, $\pi^*(p^*) = e'(p^*)p^* - e(p^*) > e(0) = 0$. Now as $\partial\pi^*/\partial p^* = v - e'(p^*) = 0$, we have from (22.1)

$$\frac{d}{dv}\pi^*(v) = p^* + [v - e'(p^*)]\frac{d}{dv}p^*(v) = p^*. \tag{22.2}$$

Since $\pi^*(v)$ is the maximum of linear function of v, it is convex in v, which means that $d\pi^*(v)/dv$, 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 result. It is remarkable, as various auctions can have completely different sets of rules and strategies.

Theorem 22.1 (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.*

Suppose there are n bidders and all participate.

Proof. From (22.1) we have $v = e'(p) \implies vp'(v) = e'(p)p'(v) = de(p(v))/dv$. Integrating this gives

$$e(p(v)) = e(p(0)) + \int_0^v wp'(w)\,dw = vp(v) - \int_0^v p(w)\,dw\,, \tag{22.3}$$

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 22.1 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 22.1 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

$$x(p) = 1 - F(p)^n\,, \qquad \text{and} \qquad p - \frac{1 - F(p)^n}{nF(p)^{n-1}f(p)} = 0\,.$$

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 probabilty that a bidder with valuation v wins is v, i.e. $p(v) = v$. From (22.3) 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.

Example 22.2 Consider the 3-person game in which player 0, the government, solicits tenders from two contractors to do a piece of work. The contractors' costs are private values that are independently distributed $U[0,1]$. The goverment takes written bids and awards the contract to the low bidder. At the Nash equilibrium a contractor with cost c will bid $(1+c)/2$. This results in expected payment of $2/3$. If it had been possible for the government to know the low bidder's cost and pay just a bit above that then the expected cost would have been a bit over $1/3$. We say that the **price of anarchy** is $(2/3)/(1/3)=2$.

22.3 Risk aversion

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. 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.

23 Auction Design

23.1 The revelation principle

We have seen that some auctioning mechanisms incentivize truthful declaration of valuations (e.g. Vickery auction), whereas for others the bidders have the incentive to shade their bids (e.g. in a sealed bid first-price auction).

However, it is sufficient to restrict attention to mechanism that incentivize truthful revelation. These are called **direct revelation mechansims**.

Theorem 23.1 (Revelation principle) *A direct revelation mechanism can generally be designed to achieve the same Nash equilibrium outcome as any other mechanisms.*

Proof. Consider some mechanism, M, for which a Nash equilibrium involves the players (agents, bidders) making untruthful revelations about their items of private information. Consider now a mechanism M' which receives information from the players, and then inputs to M the information that bidders would have themselves submitted to M, and then takes appropriate action (such as awarding an item to a bidder). Then it is a Nash equilibrium of M' for players to truthfully report to M' their private information. So M' is a direct mechanism that achieves the same result as M. ∎

This is important because it means that if we wish to design an optimal mechanism (e.g. maximizing the seller's expected revenue), we may restrict attention to mechanism that incentivize truthtelling. This is called an **incentive compatibility condition**.

23.2 Optimal auctions

In the case of 2 bidders with private valuations that are $U[0,1]$ we have seen that the expected revenue obtained by the seller is 1/3 under English, Vickery, and any other auction mechanism satisfying certain conditions. Is there any auction mechanism that creates greater expected revenue for the seller?

Consider the SIPV model in which n bidders have private valuations. Suppose it public knowledge (of the auctioneer and other bidders) that i's valuation is a sample from a distribution with distribution function F_i, and probability density function f_i.

We ask the bidders to declare their valuations, $v_1, \dots, v_n$. As a function of these declarations, $v = (v_1, \dots, v_n)$, we award the item to bidder i with probability

$\phi_i(v)$, and make him pay $p_i(v)$. Suppose that we are designing a direct mechanism such that it is optimal for all bidders to declare their valuations truthfully.

Assuming that all other bidders are declaring their valuations truthfully, a bidder i who has valuation v_i will declare it to be v_i', where

$$\begin{aligned} v_i' &= \arg\max_{v_i'} \Big\{ v_i E_{v_{-i}} \phi_i(v_1, \dots, v_i', \dots, v_n) - E_{v_{-i}} p(v_1, \dots, v_i', \dots, v_n) \Big\} \\ &= \arg\max_{v_i'} \Big\{ v_i \Phi_i(v_i') - P_i(v_i') \Big\}, \end{aligned}$$

where $E_{v_{-i}}$ denotes expectation with respect to the other bidders' valuations, If bidder i is to be incentivized also to be truthful then we will need

$$\frac{d}{dv_i'} [v_i \Phi_i(v_i') - P_i(v_i')] \Big|_{v_i' = v_i} = v_i \Phi'(v_i) - P_i'(v_i) = 0$$

for all v_i such that it is optimal to participate. Suppose $v_i \Phi(v_i) - P_i(v_i) \geq 0$ iff $v_i \geq v_i^*$, with $v_i^* \Phi(v_i^*) - P_i(v_i^*) = 0$. Integrating the above we have

$$\begin{aligned} P_i(v_i) &= P_i(v_i^*) + \int_{v_i^*}^{v_i} w \Phi_i'(w)\, dw = P_i(v_i^*) + v_i' \Phi_i(w) \Big|_{v_i^*}^{v_i} - \int_{v_i^*}^{v_i} \Phi_i(w)\, dw. \\ &= v_i \Phi(v_i) - \int_{v_i^*}^{v_i} \Phi_i(w)\, dw. \qquad (23.1) \\ EP_i(v_i) &= \int_{v_i^*}^{\infty} \left[v_i \Phi(v_i) - \int_{v_i^*}^{v_i} \Phi_i(w)\, dw \right] f_i(v_i)\, dv_i \\ &= \int_{v_i^*}^{\infty} \left(v_i - \frac{1 - F_i(v_i)}{f_i(v_i)} \right) \Phi_i(v_i) f_i(v_i)\, dv_i, \end{aligned}$$

where the final line is by integration by parts, with $f_i(v_i) = -d(1 - F_i(v_i))/dv_i$.

Let

$$g_i(v_i) = v_i - \frac{1 - F_i(v_i)}{f_i(v_i)}.$$

The expected revenue obtained by the auctioneer is therefore,

$$\sum_i EP_i(v_i) = \sum_i \int_{v_i^*}^{\infty} g_i(v_i) \Phi_i(v_i) f_i(v_i)\, dv_i = E\left[\sum_i g_i(v_i) \phi_i(v_1, \dots, v_n) \right].$$

The quantity inside the final expectation is maximized by allocating the item (i.e. setting $\phi_i(v) = 1$) to the bidder with the greatest non-negative value of $g_i(v_i)$, but awarding it to no bidder if this is negative for all declared v_i. Suppose this

$g_i(v_i)$ is an increasing function of v_i (e.g. for the uniform distribution on $[0,1]$, $g_i(v_i) = 2v_i - 1$, $v_i^* = 1/2$. Then from (23.1)

$$P_i(v_i) = v_i \prod_{j \neq i} F_j(v_i) - \int_{v_i^*}^{v_i} \prod_{j \neq i} F_j(w)\, dw$$

In the i.i.d. $U[0,1]$ case, $g(v) \geq 0$ iff $v \geq 1/2$, and for $v_i > 1/2$,

$$P_i(v_i) = v_i v_i^{n-1} - \int_{1/2}^{v_i} w^{n-1} dw = \frac{n-1}{n} v_i^n + \frac{1}{n2^n}.$$

E.g., for $n = 2$ we might take $p_i(v_i) = v_i^2/2 + 1/8$. This could be the payment in an all-pay auction in which a participant who wishes to bid v_i must commit to paying $v_i^2/2 + 1/8$ (whether or not he wins). Thus a player with valuation v_i will declare $v_i = u$ to so as to maximize

$$v_i u - (u^2/2 + 1/8).$$

This is maximized by $u = v_i$ and has a positive maximum if $v_i > 1/2$. The expected sum of the payments is

$$2EP_1(v_1) = 2 \int_{1/2}^{1} (1/8 + v_i^2/2)\, dv_i = 5/12,$$

which exceeds the 1/3 we have in other auctions. The revenue equivalent theorem does not apply because the bidder participation is not always the same.

There are other ways to create an optimal auction (i.e. one that maximizes seller's revenue). We could conduct an English auction with **reservation price**, p_0. Bidder 1 in this auction pays p_0 if bidder 2 has valuation less than p_0, and otherwise pays Bidder 2's valuation if $v_1 > v_2 > p_0$. This makes his expected payment $p_0^2 + \int_{p_0}^{v_1} u\, du = (1/2)(v_1^2 + p_0^2)$. The seller's expected revenue is maximized by taking $p_0 = 1/2$.

Or we might make each bidder pay a **participation fee** c (which must be paid by a player if he wishes to submit a bid). Now bidder 1 will participate only if $v_1 > v^*$, and in that case if he bids v then his expected payment is be

$$c + \int_{v^*}^{v} v_2\, dv_2 = c + (v - v^*) \frac{v^* + v}{2}.$$

The fact that $v_1 = v^*$ has expected profit 0 means that $(v^*)^2 - c = 0$. One can check that the seller's expected revenue is maximized by $c = 1/4$, $v^* = 1/2$.

Note that all the optimal auction have the property that a bidder will participate only if his private valuation exceeds 1/2, and this is true for any n, as this critical value is where $g_i(v) = 2v - 1 = 0$.

Another way to construct payments in an optimal auction is to declare the winner as the one with the greatest positive $g_i(v_i)$ and make him pay the smallest value of v_i for which he would still be the winner. (This is not obvious, but by a calculation we can show that this creates the right value of $P_i(v_i)$, as given in (23.1).) This generalizes the mechanism of the second-price (or Vickrey) auction.

Example 23.1 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 $g_i(v_i) = v_i - (1 - v_i)/1 = 2v_i - 1$. 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 above.

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 $g_A(v_A) = 2v_A - 1$, $v_{0A} = 1/2$, and $g_B(v_B) = 2v_B - 2$, $v_{0B} = 1$. 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 should be declared the winner and pay 0.7 (which is the smallest v such that $g_A(v) = 2v - 1 \geq 2v_B - 2 = 0.4$).

23.3 Multi-unit 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 make closed sealed-bids, and the k objects are 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 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.

24 Games and Mechanism Design

24.1 Combinatorial auctions

Multi-item auctions 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.

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 $\sum_{i=1}^n v_i(S_i^*)$. Suppose bidder i is made to pay p_i, where

$$p_i = \max_{S \in P} \sum_{j \neq i} v_j(S_j) - \sum_{j \neq i} v_j(S_j^*)\,. \tag{24.1}$$

The first term on the right of (24.1) 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.

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. This is because the profit of bidder i is $v_i(S_i) - p_i$, which is $\max_{S \in P}[v_i(S_i) + \sum_{j \neq i} v_j(S_j)]$, minus a term that is independent of what bidder i says about his valuations. Thus it is in bidder i's interest to reveal information that allows $\sum_j v_j(S_j)$ to be maximized. 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.

24.2 Distributed decisions via price mechanisms

Suppose we wish to maximize

$$W(x_1, x_2, x_3) = V_1(x_1, x_2) + V_2(x_2) + V_3(x_3, x_2).$$

Rather than there being a single decision-maker who is capable of choosing all three of the decision variables x_1, x_2 and x_3, this is to be accomplished by three decision-makers (or agents), called 1, 2, and 3, who each choose one of the variables x_1, x_2 and x_3, respectively. The optimum can be represented as the Nash equilibrium of a five-person game in which, players 1, 2, 3, 4, and 5 have control over variables (x_1, v_1), x_2, (x_3, v_3), p_1 and p_3, respectively, and their payoffs are

$$\begin{aligned}
e_1 &= V_1(x_1, v_1) - p_1 v_1 \\
e_2 &= V_1(x_2) + (p_1 + p_3)x_2 \\
e_3 &= V_1(x_3, v_3) - p_3 v_3 \\
e_4 &= p_1(v_1 - x_2) \\
e_5 &= p_3(v_3 - x_2)
\end{aligned}$$

The fact that the Nash equilibrium maximizes $W(x_1, x_2, x_3)$ can be understood by looking at a Lagrangian formulation of the problem of maximizing $V_1(x_1, v_1) + V_2(x_2) + V_3(x_3, v_3)$ under the constraints $v_1 = x_2$ and $v_3 = x_2$, and in which p_1 and p_2 are Lagrange multipliers for these constraints. Analogous to the way that we search for a fixed point, one might imagine running an algorithm in which each agent updates his decision variables in an improving direction:

$$\begin{aligned}
\dot{x}_1 &= \partial V_1 / \partial x_1 \\
\dot{v}_1 &= \partial V_1 / \partial v_1 - p_1 \\
&\vdots \\
\dot{p}_3 &= v_3 - x_2
\end{aligned}$$

If all of the V_i are concave then this algorithm converges to the global maximum.

24.3 Mechanism design for shared infrastructures

Suppose that a library of size Q can be built at cost $c(Q)$. There are n users (or agents) who will potentially share the library. Agent i will obtain utility $\theta_i u(Q)$. However, the value of θ_i is his private information. It is public knowledge only that θ_i has an *a priori* distribution with density function f_i. For simplicity we take $f_i = f$ in what follows.

We desire to design a mechanism, implemented so that (i) agents are asked to reveal their θ_i, and then (ii) as a function of these revelations of $\theta_1, \dots, \theta_n$, a size Q is chosen, and agents are made to pay fees that cover its cost, $c(Q)$.

Making use of the revelation principle, we can restrict attention to direct mechanisms. Fix a mechanism and suppose that under this mechanism if agent i declares θ_i, then the expected value of $u(Q)$ is $V(\theta_i)$ and his expected fee is $P(\theta_i)$. Knowing this, agent i will declare θ_i to be t_i so as to maximize his net benefit of

$$\theta_i V(t_i) - P(t_i).$$

If this is to be made maximal (and stationary) by choosing $t_i = \theta_i$ then we need $\theta_i V'(\theta_i) - P'(\theta_i) = 0$. Integrating this gives

$$P(\theta_i) - P(\theta_i^*) = \int_{\theta_i^*}^{\theta_i} wV'(w)dw = \theta_i V(\theta_i) - \theta_i^* V(\theta_i^*) - \int_{\theta_i^*}^{\theta_i} V(w)dw,$$

where θ_i^* is the least value of θ_i for which it would be worthwhile for agent i to consider participating, and so $\theta_i^* V(\theta_i^*) - P_i(\theta_i^*) = 0$. As in Lecture 23, let $g(\theta_i) = \theta_i - [1 - F(\theta_i)]/f(\theta_i)$. The revenue obtained from agent i will be

$$\begin{aligned}
&\int_{\theta_i \geq \theta_i^*} P(\theta_i) f(\theta_i) d\theta_i = \int_{\theta_i \geq \theta_i^*} \left[\theta_i V(\theta_i) - \int_{\theta_i^*}^{\theta_i} V(w)dw\right] f(\theta_i) d\theta_i \\
&= \int_{\theta_1} \cdots \int_{\theta_n} \phi(\theta_i) \left(\theta_i - \frac{1 - F(\theta_i)}{f(\theta_i)}\right) u(Q(\theta_1, \dots, \theta_n)) f(\theta_1) \cdots f(\theta_n)\, d\theta_1 \dots d\theta_n \\
&= \int_{\theta_1} \cdots \int_{\theta_n} \phi(\theta_i) g(\theta_i) u(Q(\theta_1, \dots, \theta_n)) f(\theta_1) \cdots f(\theta_n)\, d\theta_1 \dots d\theta_n \\
&= E[\phi(\theta_i) g(\theta_i) u(Q(\theta_1, \dots, \theta_n))],
\end{aligned}$$

where $\phi(\theta_i)$ is 1 or 0 as $\theta_i \geq \theta_i^*$ or $\theta_i < \theta_i^*$.

Our aim is to maximize total welfare of $E[\sum_i \phi(\theta_i)\theta_i u(Q(\theta_1, \dots, \theta_n)) - c(Q)]$ subject to the fees covering the cost. Suppose we impose only a weaker constraint, that the expected fees cover expected cost, i.e.

$$\sum_i E[\phi(\theta_i) g(\theta_i) u(Q(\theta_1, \dots, \theta_n))] \geq \sum_i E[c(Q(\theta_1, \dots, \theta_n))].$$

This leads us to the Lagrangian of

$$L = \int_{\theta_1} \cdots \int_{\theta_n} \Big[\phi(\theta_i)(\theta_i + \lambda g(\theta_i))u(Q(\theta_1, \dots, \theta_n)) \\ - (1+\lambda)c(Q(\theta_1, \dots, \theta_n))\Big] f(\theta_1)\cdots f(\theta_n)\, d\theta_1 \dots d\theta_n.$$

To illustrate ideas, let us suppose that $\theta_i \sim U[0,1]$, so $g(\theta_i) = 2\theta_i - 1$. Then the maximum of L is achieved by taking $\phi(\theta_i) = 1$ if $\theta_i + \lambda g(\theta_i) \geq 0$, i.e. if $\theta_i \geq \lambda/(2\lambda+1) = \theta_i^*$. Otherwise $\phi_i(\theta_i) = 0$. Note that θ_i^* increases from 0 and 1/2 as λ increases from 0 to infinity. For a given $\lambda \geq 0$ we know how to find the ϕ_i. We also know that Q should be chosen to maximize the integrand, so

$$Q(\theta_1, \dots, \theta_n) = \arg\max_Q \Big\{ u(Q) \sum_i \max[\theta_i + \lambda g(\theta_i), 0] - (1+\lambda)c(Q) \Big\}.$$

The solution is completed by finding the value of λ such that the expected fees obtained from the agents match expected cost.

There is more we can say about this problem. For instance, it is possible, to take the above mechanism, in which expected fees cover expected cost, and strengthen it to a mechanism in which actual fees cover actual cost.

The optimal mechanism is complicated. However, as n becomes large, one can prove that nearly the same welfare can be achieved by a simple scheme that announces that the library will be of size Q^* and charges a fixed subscription fee p^* to any agent who then chooses to participate, which will be agents for whom $\theta_i u(Q^*) - p^* \geq 0$. The values of parameters Q^* and p^* are chosen so that $nP(\theta_i u(Q^*) \geq p^*) = c(Q^*)$ and $nE[\theta_i | \theta_i u(Q^*) \geq p^*]u(Q^*) - c(Q^*)$ is maximized.

24.4 The price of anarchy

The **price of anarchy** (PoA) is a measure of the inefficiency of selfish behavior. It is the quotient of the reward that can be obtained under centralized control to the lesser reward that is obtained at the worst Nash equilibrium of the game which results when agents all act selfishly. So in the section above the PoA is the quotient of the welfare that could be obtained if agents were to truthfully reveal their θ_i (without any need for incentivizing), to the expected welfare obtained when they must be incentivized to reveal their true θ_i.

In the case of costs, as in Example 22.2, the PoA is the ratio of the cost at the worst Nash equilibrium to the lesser cost that is possible under centralized control.

Index